An Enhanced Genetic Algorithm-Based Timetabling System with Incremental Changes

Ahmed F. AbouElhamayed, Abdarhman S. Mahmoud, Tarek T. Shaaban, Cherif Salama, Ahmed H. Yousef
Computers and Systems Department
Faculty of Engineering, Ain Shams University
Cairo, Egypt
ahmed.abouelhamayed@hotmail.com, abdarhman_taha@hotmail.com, tarektalaat93@hotmail.com,
cherif.salama@eng.asu.edu.eg, ahassan@eng.asu.edu.eg

***Abstract*— Constructing a timetable is a widespread problem. Computers can be employed to solve this problem faster and to produce better solutions. Software solutions for this problem already exist and are used by some universities. However, some universities have complex types of constraints that make it hard to use most of the available software solutions. This paper introduces a software solution for the curriculum based timetabling problem with flexibility to include some complex types of constraints. In addition, the solution provided here allows for incremental changes when new constraints are added after generating the timetable. The solution is based on the genetic algorithm with some modifications to some of the operators to enhance the algorithm. The preliminary results show that it is possible to represent some complex constraint types. Also, that incremental changes can be implemented to reach a close solution faster. It is also shown that the operators of the genetic algorithm can be modified to better suit the timetabling problem and produce better results.**

Keywords—Timetabling; Genetic Algorithm; Scheduling; Incremental Changes; Local Search; Selection; Genetic Diversity.

I. INTRODUCTION

The weekly timetable of a class is a set of time slots assigned to a specific number of subjects where each combination of one subject and one time slot is given a specific unshared room. The process of constructing a timetable is not an easy one. The timetable needs to comply with several constraints, imposed by university rules, teaching staff constraints, capacity of rooms, and many other constraints. Constructing the timetable usually starts by gathering all the constraints, then trying to put each class in a specific time and detecting conflicts, then trying to resolve them just to find other conflicts and so on until finally reaching a timetable that complies with as much constraints as possible without conflicts.

It's better to solve this problem using software solutions[1]. The problem in practice is an NP-complete problem[2]; So, Approximate approaches are used to find a feasible solution. Techniques used to tackle that problem include tabu search[3], constraint logic programming[4], SAT-based solvers[5] and evolutionary algorithms[6]. More than one technique are usually combined while solving the problem as used in [7] and [8]. The solution presented is based on the genetic algorithm[9]. A brief description of the whole timetabling system is presented along with a focus on the core solver that solves the timetabling problem.

A. Problem Statement

A formal representation of the problem is College: **< R, S, P, H, T >** Where; **R** is a specific set of rooms $<R_1, R_2 \dots R_t>$ inside the college where t is the total number of rooms in the college; **S** is a specific set of groups of Subjects $<G_1: <S_1, S_2 \dots S_m>, G_2: < S_1, S_2 \dots S_n> \dots G_t>$ given to classes where m and n are the number of subjects assigned to each group or class and t is the total number of classes are there in the college; **P** is a set of Professors $<P_1, P_2 \dots P_n>$ where n is the total number of professors; **H** is a set of slots that represents the total number of Hours $<H_1, H_2 \dots H_n>$ (time intervals) per week; **T** is the timetable generation function that computes an evaluation of the current timetable that to be minimized. So, it is required to find *min* <**T**> where **T**: < **I, C, D** > Where **I** is a set of infeasibilities that are:

- Making two classes attend different lectures in the same room at the same time.
- Making the same teacher deliver two different lectures at the same time.
- Excluding one or more of the subjects assigned to a certain class.

C is a set of sets of Constraints $<P_1: <C_1, C_2 \dots C_n>, P_2: < C_1, C_2 \dots C_n > \dots P_m: < C_1, C_2 \dots C_n >>$ each set belongs to a specific professor regarding their:

- Unavailability time slots.
- Max hours per day.
- Max days per week.
- Max and min interval between slots.
- Max continuous working hours.

D is a set of didactic constraints, those that assure the quality of the timetable that all the factors <Students, Staff and College> are subject to the optimum educational process like:

- Having two days off per week for students.
- Avoiding long distance between successive lectures in different rooms.

B. Motivation

Existing software solutions for the timetabling problem lack some important features that make them unsuitable for use in many universities. There's a need to represent some complex types of constraints; Some universities have complex types of constraints such as having different programs that share the staff members and rooms but differ in number of time slots occupied by a lecture. After generating the timetable, some new constraints appear but that would probably happen after the timetable had been announced and accepted by most of the people. Generating a totally new timetable would not be desirable in such case; That is why a solver that supports incremental changes is needed.

A comparison between some existing software solutions:

	TimeTabler	Mimosa
Incremental changes	Supported	Supported
Support multiple systems	Not supported	Not supported
Decentralization	Not supported	Supported

So we decided to design our system to support all these features.

C. Related work

Genetic algorithm has been in use to solve the timetabling problem for a while. The genetic algorithm is usually modified or used in conjunction with other techniques to arrive at a good solution. Encoding is usually the first step in forming a solution[10]. The genetic operators are usually modified to solve the problem[11]. A local search is sometimes used[12-14]. For a comparison between a genetic algorithm version with and without local search, see[15]. Some solutions combine the genetic algorithm with other techniques such as tabu search[16]. Some solutions include an initial population generation function such as [17] and [10]. Due to the nature of the genetic algorithm which makes it expensive but parallelizable, some solutions introduced a parallelized genetic algorithm[18].

D. Contribution

- Selecting an encoding that allows some complex constraint types to be represented without enforcing extra constraints.
- Adding a modification to the algorithm to allow incremental changes to the timetable after it's generated.
- Selecting a set of operators that improve the performance of the algorithm.

E. Outline

The introduction section provides an overview of the contents of the paper and its purpose. It is followed by the methodology section where the details of the solver and the approach are explained. Some experiments along with their results and a brief discussion are presented in the results and discussion section. Finally, a conclusion of the paper and suggestions for future work are presented.

II. METHODOLOGY

The solver is the core part of the timetabling system but a method to easily enter the data makes the timetabling system more usable. A web interface was implemented to help with collection of data into the database. This data represents the problem that is to be solved. Instead of letting the solver access the database whenever needed, to check for violations for instance, data from the database was compiled into some constant numbers and constant arrays to represent the problem. The core solver uses these constants and solves the problem then the solution is imported into the database for the web interface to show the resulting timetable. The core solver is explored in more details.

A. Encoding

Encoding is the way the solution is represented. It's an important part of solving the timetabling problem. A good encoding would enforce as many constraints as possible by just putting the solution in the given representation. One encoding that's common in the literature[10, 16] is having a 2D array where the indices represent the time slot and the room; Values represent the lecture that is to be given in that combination of time slot and room. This encoding enforces many constraints which makes it pretty popular; However, for some scenarios, it enforces some constraints that don't really exist. An example of that is when 2 lectures are given once every 2 weeks. You can expect that they are scheduled together in the same time slot and room. Using the mentioned encoding, a solution containing that arrangement can't be represented, thus can't be reached. Another encoding[13] was selected which doesn't have that problem. It still enforces many constraints but allows representing all possible solutions. See Figure 1. The timetable is mapped into an array **T** of size $2\times\mathbf{L}$, where **L** is the total number of lectures for all courses taught in the college. Each index in **T** represents a specific lecture for a specific course. Each lecture of each course is given a unique number **i** where $0<=\mathbf{i}<\mathbf{L}$; $\mathbf{L_i}$ means the lecture number **i**. Let **S** be the total number of time slots available in a week. $\mathbf{S} = d\times t$ where d is the number of working days for the college and t is the number of time slots per day. Each time slot is given a unique number **j** where $0<=\mathbf{j}<=\mathbf{S}$; $\mathbf{S_j}$ means the time slot number **j**.

Value in $\mathbf{T[L_i]}$ represents the number of the time slot at which $\mathbf{L_i}$ is to be given. If we say $\mathbf{T[L_i]=S_j}$, this means that lecture number **i** is to be given in time slot number **j**.

Let **R** be the total number of rooms available in the college. Each room is given a unique number **k** where $0<=\mathbf{k}<=\mathbf{R}$; $\mathbf{R_k}$ means the room number **k**.

Value in $\mathbf{T[L+L_i]}$ represents the number of the room at which $\mathbf{L_i}$ is to be given. If we say $\mathbf{T[L+L_i]=R_k}$, this means that lecture number **i** is to be given in room number **k**.

Figure 1 - Encoding

B. Fitness Function

The fitness function is the function that when given some solution, specifies how good or bad it is. As the return value of the fitness function decreases, the solution is better. The fitness function could simply be checking if there's any violated constraint and return infinity if that's the case. If there are no violated constraints, it returns 0. A problem with that approach is that we get no information from the fitness function about how good or bad the solution is. Consider a case where the fitness function is given 2 solutions; One of the solutions has all the lectures scheduled at the same time slot in the same room and the other one has all the lectures scheduled in different time slots and rooms with the exception of 2 lectures that are scheduled at the same time slot and room. That simple fitness function would give the same value for both schedules, infinity. Using the genetic algorithm, we can't know which of the solutions are the best in the population to increase their probability of being selected into the next generation.

The fitness function needs to be returning a value that is proportional to the number of constraints violated so as to get a sense of how good or bad a solution is. We have 2 types of constraints; Hard constraints and soft constraints. Violations to hard constraints can't be tolerated. So, violating a single hard constraint while violating any number of soft constraints is worse than satisfying all soft constraints with 1 violating one hard constraint. Think for example of a schedule where all groups have no gaps between their lectures. That's great but if the schedule has one of the groups needing to attend two lectures at the same time then it is infeasible. It's better to have a gap than to have to attend two lectures at the same time.

The existence of two types of constraints doesn't mean that there're two values of penalty only. It turns out some of the soft constraints are more critical than others. The seniority of the staff members for example means that those with higher seniority should be comfortable first, then those with lower seniority are to be considered. Soft constraints then need to have different penalties to account for the severance of the violation. Hard constraints on the other hand all have the same penalty as violating any single hard constraint means that the solution is infeasible and unacceptable anyways. This is different from the simple fitness function described earlier where any violation returns infinity. Here, violation of two hard constraints returns a value that is double the value returned by violation of one hard constraint. So, the fitness function still shows how good or bad a solution is. Figure 2 shows the mathematical model of the fitness function.

$$fitness\ value = P_h * N_h + \sum_{n=1}^{N_s} P_s(n);\ where\ P_h > \sum_{i=1}^{N_s} P_s(i)$$

P_h *is penalty of violating a hard constraint.*

$P_s(i)$ *is the penalty of violating soft constraint number i.*

N_h *is the number of hard constraints.*

N_s *is the number of soft constraints.*

Figure 2 – Mathematical model of the fitness function

C. Incremental Changes

It is not unusual that new constraints appear after the timetable is generated and accepted by many. It is desired to support incremental changes by generating a new timetable that satisfies new constraints but is as close as possible to the old existing timetable. There are other approaches that handle incremental changes in constraints[19]. However, up to our knowledge, no proposed method for supporting incremental changes using the genetic algorithm was suggested till now. Incremental changes is implemented by having the existing timetable defined as a constant inside the fitness function and adding a penalty for every difference in the values of the fields between the existing timetable and the suggested one. Additionally, the mutation operator is modified to inject the existing timetable to be part of the population. The results of this implementation can be seen in experiment 2.

D. Crossover Operator

There are several crossover operators. Only basic ones were explored in our experiments. Single-point crossover and double-point crossover produced best results with the rest of the operators we defined. Single-point crossover was selected. The single-point crossover chooses a break point at random then takes the part of the chromosome before the break point from the first parent and the other part from the other parent.

E. Mutation Operator

Our mutation operator is chosen to serve multiple roles. Each role is accomplished by a specific operation. So, the mutation operator is divided into a set of operations where each operation has a certain probability of being the chosen operator to occur. The roles and operations of the mutation operators are described next.

The first role is to help delay convergence and explore other areas of the search space rather than getting stuck at a local minimum. The corresponding operation is to completely ignore the incoming chromosome and output a chromosome that is constructed by putting random values within the bounds in each index of the array.

Another role is to help make the resulting timetable as close as possible to the existing timetable if provided (incremental changes). The corresponding operation is to completely ignore the input timetable and return the existing timetable so it is just injecting the existing timetable into the generation.

Another role is to help make the current solution better than it is. The corresponding operation is a local search operation which starts with the suggested timetable and looks for the best value in each of the values of the array while assuming all other values constant as shown in the pseudocode.

```
Set best_fitness to fitness of chromosome
Set best_chromosome to chromosome
Loop i from 1 to size of chromosome in a random order
   Loop j from minimum to maximum possible values of cell i
      Put j in cell i of chromosome
      If fitness of chromosome < best_fitness
         Set best_fitness to fitness of chromosome
         Set best_chromosome to chromosome
Return best_chromosome
```

F. Selection Operator

There are several factors that determine the performance of the genetic algorithm and its ability to find the best solution. One major factor is the diversity of the population. Diversity of a population is a measure of how much the population is not similar; it measures the variety of solutions in the population. If the average distance between the individuals in the population is large we say that the population has high diversity. In contrast, if the average distance between the individuals in the population is small, we say that the population has low diversity.

This means that if individuals of the population have similar or near similar fitness values then the population has low diversity or no diversity at all. If the individuals of the population have different fitness values, then the population has a high diversity.

There are several factors that cause the loss of population diversity[20]. One of them is the selection pressure which is the tendency to select only the best members of the generation to survive to the next generation and is required to direct the genetic algorithm to the optimum. Too much selective pressure lowers the genetic diversity and causes the algorithm to converge to a local optimum. Too little selective pressure prohibits genetic algorithm from converging to an optimum in a reasonable time. So, a proper balance is needed between selective pressure and diversity for the algorithm to converge in a reasonable time to a global optimum.

Convergence of a population to a solution means that no better offspring are generated as a result of selecting parents form the current population. Populations with a lack of diversity cause the pre-mature convergence of the genetic algorithm. Pre-mature convergence is the convergence to an undesirable non-global suboptimal solution. This occurs as a result of the population having individuals of the same fitness value which is not the best fitness value. The reason for the occurrence of this case is the traditional selection mechanism of the genetic algorithm which tends to favor the best individuals in the current population and replicates them while discarding the less fit individuals. These best individuals do not have necessarily the best fitness values and may not result in finding the global optimum solution. So, over multiple successive generations the population tends to have similar individuals having suboptimal fitness values which results in convergence to a suboptimal solution. On the other hand, if we keep the less fit solutions for some more generations, the crossover and mutation operations between the more and less fit individuals may result in finding an even better solution than both. So, the algorithm performs better if the population is kept as diverse as possible for as many generations as possible which increases the exploration so a better single solution can be found. Techniques that delay the formation of homogenous populations (having individuals of similar fitness values) result in better solutions[21].

Another purpose of maintaining population diversity is to locate multiple final solutions. It should be noted that techniques that locate multiple solutions are also highly effective in delaying convergence. But techniques which only delay population convergence are not useful for locating multiple solutions[20].

There are many techniques that are used to maintain population diversity. Mutation is an example of diversity promoting process. Other examples are a selection method based on the gender concept[21] and stochastic universal sampling[20]. The chosen selection operator in our system is the gender selection operator. Gender selection is a selection function based on the gender concept as explained in [21].

As stated earlier, the traditional selection mechanism of the genetic algorithm tends to discard the less fit individuals and select the best fit individuals and replicates them either by passing the best of them to the next generation without change (elite count in Matlab) or by choosing them as parents for the children that will result from the crossover operation and form the next population. If we restrict the mating process between parents such that it occurs between a parent of high fitness and another of low fitness (the ones getting discarded), we can delay the population convergence and increase the diversity of the population. This is the basic idea of the gender selection operator. The implementation follows.

The population is sorted by its score, the first half of the population has individuals of higher fitness values and the second half has individuals of lower fitness values. The first half is considered to be one gender (females) and the second half is the other gender (males). Pairs of population members are selected from the two halves of the population for crossover to reproduce an offspring. One candidate is selected from the first half and the other candidate is selected from the second half, since the first half has individuals of high fitness and the second half has individuals of lower fitness, selection of opposite genders is ensured and one parent is ensured to be of high fitness value while the other parent is ensured to be of low fitness value.

The selection from each half is done using tournament selection. The default tournament size is 4 but it can be changed. In tournament selection a number of candidates equal to tournament size are chosen from each half and they compete based on their fitness values. The fittest individual gets the chance to be selected for reproduction. The tournament size is important to control selection pressure and to keep diversity in the population. Large tournament size means weak individuals have a smaller chance of being selected. Small tournament size means that weak individuals have greater chance of being selected.

See experiment 3 for the efficiency of the operators.

III. Results & Discussion

Some experiments were carried out to test the ability of the system to handle complex constraints and to test the efficiency of the solver. This section contains these experiments, their results and a discussion of these results. The second instance from ITC 2007 Curriculum-based course timetabling[22] is used as a test data in some of the experiments.

The problem statistics are as follows: courses: 82, rooms: 16, periods per day: 5, days: 5, curricula: 70, max daily lectures: 4, min daily lectures: 2, room occupation per room: 70.76%,

conflict density per course: 7.11%, conflict density per lecture: 7.97%, teachers' availability per course: 74.98%, teachers' availability per lecture: 76.90%, rooms' suitability per course: 57.93%, rooms' suitability per lecture: 58.75%.

A. Experiment 1: Complex Constraint Types

Some universities have some lectures that can be only once per 2 weeks (alternating lectures). If the constraints enforce that 2 of those lectures be scheduled at the same time slot and at the same room, then the encoding selected here can handle it. This was tested by constructing a problem that has no solution but to place 2 of the alternating lectures in the same time slot and room. The solution reached by the solver can be seen in Figure 3.

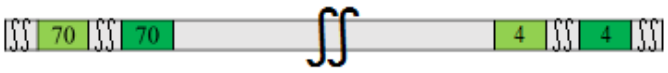

Figure 3 – Valid solution chromosome for experiment 1

It can be seen that two periods were scheduled to be given at time slot 70 in room 4 which is valid if the two periods are alternating. An encoding where the indices of the array represent a given time slot and a given room can't be used to represent such a solution as only one value can be present.

B. Experiment 2: Incremental Changes

Only hard constraints were considered in this experiment. The second instance from ITC2007 was used after removing some of the unavailability constraints. The solver was run and an initial timetable was obtained. 9 unavailability constraints were added to the system and the solver was run once using the incremental changes feature and once without using the incremental changes feature as if it's solving a new problem. The results of the runs are presented in Table 1.

Solver	Number of generations	Periods with changed assignments
Without incremental changes	76	279
With incremental changes	23	3

Table 1 – The effect of using incremental changes

Table 1 shows that using incremental changes, the solver was able to arrive at a solution in less than third the number of generations taken by the solver without incremental changes. It also shows that without incremental changes, 279 out of 283 periods had different assignments in the initial timetable and the new timetable. Using incremental changes, the number of periods that have different is reduced to 3.

These results show that the incremental changes helped arrive at a solution in a less number of generations and the obtained timetable was very close to the original one.

C. Experiment 3: Improved Genetic Algorithm Operators

This experiment was carried out to see the effect of modifying the genetic algorithm operators on the efficiency of the solver and the speed of finding a solution. The crossover operator was selected to be single-point crossover as it produced best results in our experiments. Only hard constraints were considered. The solver was run with different operators. To increase the accuracy of the results, for each set of operators, the experiment was repeated five times and the average of the numbers is plotted in Figure 4.

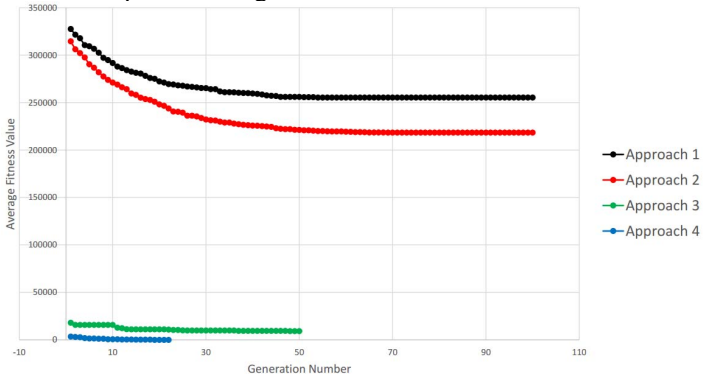

Figure 4 - Fitness value vs Generation number for different set of operators

Approach 1 uses gaussian mutation operator and stochastic uniform selection operator. Approach 2 uses gaussian mutation operator and the gender selection operator. Approach 3 uses the local search mutation operator and stochastic uniform selection operator. Approach 4 uses the local search mutation operator and the gender selection operator.

It is clear that without modifying the operators, the genetic algorithm converges quickly into an infeasible solution. When using the gender selection operator, the solver reaches a better solution but it's still an infeasible one. Using the local search function as the mutation operator causes the solver to be able to reach better solutions in a smaller number of generations. However, it still converged before finding a feasible solution in that experiment. Combining the local search operator with the gender selection method, the solver managed to reach a feasible solution in a small number of generations. It can be concluded that using modifications to the genetic algorithm operators is needed to build a better solver.

IV. Conclusion & Future Work

It is shown that a system that holds complex constraint types is doable which gives some hope that in the future, most of the universities would be using software to construct their timetables. The incremental changes feature was successfully implemented and tested. The results are very promising. This resolves one big problem that faces universities when using software that doesn't provide support for that.

The future steps include more enhancement to the genetic algorithm operators to achieve better results. In addition to testing the incremental changes on larger instances.

ACKNOWLEDGMENT

The work published in this paper was partially performed as a graduation project in the Faculty of Engineering, Ain Shams University. We'd like to thank Aya Mahmoud, Esraa Rabie, Mohammad Yahia, Mona Matar and Tareq Elgafy for their work in the project. We'd like to also thank all individuals who provided help to the project.

REFERENCES

[1] E. Burke, D. Elliman, and R. Weare, "The automation of the timetabling process in higher education," *Journal of Educational Technology Systems,* vol. 23, pp. 353-362, 1995.

[2] T. B. Cooper and J. H. Kingston, "The complexity of timetable construction problems," in *Practice and Theory of Automated Timetabling: First International Conference Edinburgh, U.K., August 29–September 1, 1995 Selected Papers*, E. Burke and P. Ross, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 281-295.

[3] D. Costa, "A tabu search algorithm for computing an operational timetable," *European Journal of Operational Research,* vol. 76, pp. 98-110, 1994.

[4] P. Boizumault, Y. Delon, and L. Peridy, "Constraint logic programming for examination timetabling," *The Journal of Logic Programming,* vol. 26, pp. 217-233, 1996.

[5] R. Asín Achá and R. Nieuwenhuis, "Curriculum-based course timetabling with SAT and MaxSAT," *Annals of Operations Research,* pp. 1-21, 2012.

[6] P. Adamidis and P. Arapakis, "Evolutionary algorithms in lecture timetabling," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 1999.

[7] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *European Journal of Operational Research,* vol. 176, pp. 177-192, 1/1/ 2007.

[8] S. Abdullah, E. K. Burke, and B. McCollum, "A hybrid evolutionary approach to the university course timetabling problem," in *2007 IEEE congress on evolutionary computation*, 2007, pp. 1764-1768.

[9] Z. Michalewicz, "GAs: What are they?," in *Genetic algorithms+ data structures= evolution programs*, ed: Springer, 1994, pp. 13-30.

[10] E. Yu and K.-S. Sung, "A genetic algorithm for a university weekly courses timetabling problem," *International transactions in operational research,* vol. 9, pp. 703-717, 2002.

[11] B. Sigl, M. Golub, and V. Mornar, "Solving timetable scheduling problem using genetic algorithms," in *Proc. of the 25th int. conf. on information technology interfaces*, 2003, pp. 519-524.

[12] S. Abdullah, H. Turabieh, B. McCollum, and E. K. Burke, "An investigation of a genetic algorithm and sequential local search approach for curriculum-based course timetabling problems," in *Proc. Multildisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2009), Dublin, Ireland*, 2009, pp. 727-731.

[13] S. Abdullah and H. Turabieh, "Generating university course timetable using genetic algorithms and local search," in *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, 2008, pp. 254-260.

[14] S. Yang and S. N. Jat, "Genetic algorithms with guided and local search strategies for university course timetabling," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 41, pp. 93-106, 2011.

[15] A. Colorni, M. Dorigo, and V. Maniezzo, "A genetic algorithm to solve the timetable problem," *Politecnico di Milano, Milan, Italy TR,* pp. 90-060, 1992.

[16] S. Abdullah and H. Turabieh, "On the use of multi neighbourhood structures within a Tabu-based memetic approach to university timetabling problems," *Information Sciences,* vol. 191, pp. 146-168, 2012.

[17] R. Weare, E. Burke, and D. Elliman, "A hybrid genetic algorithm for highly constrained timetabling problems," *Department of Computer Science,* 1995.

[18] D. Abramson and J. Abela, *A parallel genetic algorithm for solving the school timetabling problem*: Citeseer, 1991.

[19] H. Cambazard, F. Demazeau, N. Jussien, and P. David, "Interactively solving school timetabling problems using extensions of constraint programming," in *International Conference on the Practice and Theory of Automated Timetabling*, 2004, pp. 190-207.

[20] S. W. Mahfoud, "Niching methods for genetic algorithms," *Urbana,* vol. 51, pp. 62-94, 1995.

[21] K. Tahera, R. Ibrahim, and P. Lochert, "A Comparative Study of Gender Assignment in a Standard Genetic Algorithm," in *Trends in Intelligent Systems and Computer Engineering*, ed: Springer, 2008, pp. 167-174.

[22] L. Di Gaspero, B. McCollum, and A. Schaerf, "The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3)," Citeseer2007.