Received 3 March 2008
Revised 7 July 2008
Accepted 14 July 2008

A heuristic solution to the university timetabling problem

Aderemi O. Adewumi
Faculty of Science, School of Computational and Applied Mathematics, University of Witwatersrand, Johannesburg, South Africa
Babatunde A. Sawyerr
Faculty of Science, Department of Computer Sciences, University of Lagos, Lagos, Nigeria, and
M. Montaz Ali
Faculty of Science, School of Computational and Applied Mathematics, University of Witwatersrand, Johannesburg, South Africa

Abstract

Purpose – The purpose of this paper is to consider the problem of university lecture timetabling. Timetabling deals with the problem of placing certain resources into a limited number of time slots, subject to given constraints, in order to satisfy a set of stated objectives to the highest possible extent. It is a well-known and established NP-hard problem. University timetabling is a major administrative activity especially in the third world universities. Solving the problem requires dynamic heuristics with predictable performance especially as the number of courses increases without corresponding increase in needed resources.

Design/methodology/approach – A genetic algorithm metaheuristic is designed to handle a real-life case study. Given the present structure of the case study, a modular approach to the design of the timetable schedules is adopted. The approach considers timetable in a bottom-up fashion at the various levels of department, faculty or entire university. Simulation study is conducted using the open source Java IDE, Eclipse® 3.0 in a window XP/vista environment running on a processor of 1.12 GHz.

Findings – Using the data sub-set from the case study, simulation experiments are conducted based on the proposed method and obtained promising results.

Research limitations/implications – Given the modular approach, the timetable system can easily be adapted to other various levels in the institution.

Originality/value – With reference to the case study, this is believed to be the first application of metaheuristics to a timetabling problem. The sensitivity analysis of the algorithm parameters is very valuable in guiding actual application development for the problem.

Keywords Universities, Lectures, Time-based organizations, Programming and algorithm theory

Paper type Research paper

1. Introduction

Timetabling problem (TTP) is an NP-hard optimization problem that involves the allocation of certain resources, subject to constraints, into a limited number of time slots with the aim of satisfying a set of stated objectives to the highest possible extent (Wren, 1996). TTPs arise in a wide variety of domain including education (e.g. university and school timetabling); healthcare institutions (e.g. nurse and surgeon rostering); transport (e.g. train and bus timetabling) and sport (e.g. league scheduling). University timetabling (UTT) is a major and regular administrative activity in most academic institutions. It is a special form of real-life optimization problem (Schaerf, 1999). Most institutions employ the knowledge and experience of expert personnel with regard to the production of good timetable that satisfy all given (and sometimes, conflicting) requirements. UTT involves the arrangement of students, lecturers, courses and lecture rooms in an optimal way so as to minimize the non-satisfaction of

Engineering Computations: International Journal for Computer-Aided Engineering and Software
Vol. 26 No. 8, 2009
pp. 972-984

0264-4401
DOI 10.1108/02644400910996853

the requirements of each of these entities. As in many optimization problems with large instances, exact methods have proved to be either inappropriate or inefficient thus the application of heuristic and metaheuristic algorithms is being adopted in recent time (White and Zhang, 1998; Dammak *et al.*, 2006; Zampieri and Schaerf, 2006). Genetic algorithm (GA) particularly has proved very efficient in handling timetabling and similar problems (Adewumi *et al.*, 2005, 2008). In a broad sense, UTT can be divided into two namely, lecture (course) timetabling and examination timetabling, each with its own sets of constraints and requirements. The focus of this paper is on lecture timetabling.

1.1 Background

An optimization problem can be mathematically defined as follows (Vesterstrøm, 2005):

> Let S be a search space and $F \subseteq S$ be the feasible part and f a fitness function. An optimization problem aims at finding an $x \in F$, such that $f(x) \leq f(y)$ (minimization problems) and $f(x) \geq f(y)$ (maximization problems) for every $y \in F$. The solution, x, is called the global optimum. The fitness function f is either numerical ($f : S \rightarrow R$) or ordinal ($f : S \times S \rightarrow S$). An element $x \in F$ is a local minimum if $f(x) \leq f(y)$ for all $y \in N(x)$ and a local maximum if $f(x) \geq f(y)$ for all $y \in N(x)$, where $N(x)$ is a defined neighbourhood function.

There are continuous or discrete (combinatorial) global optimization problems arising in different applications. Timetabling falls under the category of combinatorial optimization problems (COPs). Combinatorial optimization algorithms solve instances of problems that are believed to be NP-hard, by exploring the usually large solution space of these instances. This is achieved by reducing the effective size of the space and then exploring the space efficiently. A number of recent COPs are modeled and solved based on some real-life applications. Such is the case with TTPs especially in tertiary institutions. It is possible to solve a COP without having a model of it just as one can work directly on the physical system and try out different solutions based on the feedback derived from the system (Vesterstrøm and Riget, 2002). Exacts methods have been applied successfully but in most cases to smaller instances of COPs. Heuristic algorithms were later developed to provide faster solutions to more complex instances of COPs. A heuristic is a method of performing a minor modification, or a sequence of modifications, of a given solution or partial solution in order to obtain a different solution or partial solution (Kreher and Stinson, 1999). Each modification involves a neighbourhood search. A heuristic algorithm consists of iteratively applying one or more heuristics in accordance to a certain design strategy.

Metaheuristics, a term coined by Fred Glover in 1986 (Gendreau, 2003), have become a leading edge among heuristic approaches for solving COPs. In defining and explaining metaheuristics, Glover and Kochenberger (2003) in their preface state that "metaheuristics, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space. Over time, these methods have also come to include any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighbourhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes". Metaheuristics

are generally applied to problems for which there is no satisfactory problem-specific algorithm or heuristic; or when such an algorithm cannot be implemented for such problems.

2. Lecture timetabling problem

Lecture timetabling problem (LTTP) involves scheduling a number of students taking given course(s), lecturers and lecture rooms into a fixed set of timeslots per days of the week. As in general optimization problem with stated objectives, decisions, available resources and constraints, lecture timetabling (LTT) aims at allocating resources to the above-stated entities in an optimal schedule. The optimal schedule, in most cases, depends on the real-life domain under consideration as a result of diversity in constraints, resources and requirements of different real-life scenario. However, for a generic LTT, one would expect an optimal schedule to be, for example, a schedule where no lecturer, class of students or classroom is used more than once in any given period. Each instance of students, lecturer, and lecture room combination can thus be considered as a "schedule". The LTTP aims then at finding a schedule that minimize (if not completely eliminates) overlapping of schedules within the same period of time.

The constraints in most LTTP are introduced because of the given (dynamic or fixed) number of students taking a course, lecture room capacity, lecturers' preference, etc. Generally, all given constraints can be classified into two: hard and soft. Hard constraints are conditions that must be rigidly enforced within the generated timetable schedule while soft constraints might be slacked sometimes without much effect on the generated schedule. Soft constraints are desirable but not so-important qualities of the schedule. Most often, it is practically impossible to satisfy all soft constraints in real-life university LTTPs. In spite of the diversity in constraint definitions across domains, the kernel (main requirements) of the LTTP remains fixed. In all institutions, the constraints impose that a Lecturer will not teach two different courses at the same time; that a room can only contain a class of students at a time; and that a given group of students cannot attend two different classes at the same time remain the same. These are examples of hard constraints. Examples of soft constraints could include: Lecturer wishing to have non-consecutive class allotments; minimizing the distance lecturer/student walks to lecture rooms; an even distribution of classes over the week; or proximity of classes to the home department.

Solutions that do not violate hard constraints are referred to as feasible solutions. Though desirable, the large number of possible and different combinations of feasible solutions makes the search for an optimal solution a difficult task. The objective of an ideal LTT would then be to provide an optimal compromise to satisfy as many conflicting conditions as possible.

Problems relating to timetabling had been studied. White and Zhang (1998) employed tabu search hybridized with constraint logic program to solve the LTP for a small data sets of a university timetable. Results obtained from the hybridized algorithm were compared with those of the individual heuristics. In the words of White and Zhang (1998, p. 2), "an automated timetabling system should formulate complete descriptions of which students and which teachers should meets, at what locations, at what times and should accomplish this quickly and cheaply while respecting the traditions of the institutions and pleasing most of the people involved most of the time". Dammak *et al.* (2006) formulated the lecture and tutorial TTP of a university in Tunisia as a zero-one integer linear programming problem. A three-stage heuristic was developed in solving the problem taking into account students, professors and

proximity constraints. Zampieri and Schaerf (2006) modeled and solved the examination timetabling problem (ETTP) which is a variant of the generic TTP with stricter constraints. Non-overlapping of examination courses, spreading of examinations for students as much as possible, allocation of spacious classrooms are some strict requirements of the ETTP. A case of an Italian university ETTP was considered and the solution obtained by Tabu Search was presented.

2.1 Problem description

This paper employs a case of the LTTP as obtained in a Nigerian University. We concentrate on a module of the LTT from one of the faculty. The university is structured around nine faculties and a college of medicine, each having various departments under it. Courses are offered by each department for students. Generally, courses within the university are designated as compulsory, elective or optional. On few occasions, students are permitted to "borrow" approved courses from other departments within and/or outside their faculty in order to make up the required minimum number of course units. On a uniform note, a course has a designated and unique course code, course title, number of units (a measure of the workload of the course) and allotted lecturer(s) who is/are to teach in any assigned classroom depending on the timetable schedule. To reduce multiplicity of courses across various departments, the university designates some courses taken by most students within one or more faculties as faculty or general courses. For example, FSC102 is a Faculty of Science (FSC) course that is compulsory for all first year students within the faculty. The first three letters of a course code identifies either the department or faculty where the course is being offered while the first digit of the course code identifies the level (year) at which it is being offered. FSC101 is thus a FSC course for 100-level students. The last two digits ("01" in this case) serve as a unique identity for each course. In the same vein, CSC322 is a 300-level computer science course. Thus FSC101 is different from FSC102.

In terms of space allocation for lectures, most lecture rooms are located around each faculty and are meant to be utilized by all departments within that faculty. Exemptions to this arrangement are few separate lecture theatres (usually larger in capacity) designated for most large inter-faculty and/or general courses. Given this arrangement of the lecture rooms in our case study, we adopt a modular design of the LTP in which a bigger LTTP is broken down into modules for departmental, faculty and general courses. The entire LTT can thus be built in a bottom-up fashion. Hence, an LT designed for a department can be expanded to incorporate more departments within the faculty for department/faculty courses. This is made possible since factually the same lecture room facilities are available for them with the exception of designated laboratories. Laboratory allocation is the sole responsibility of individual department. The current work does not incorporate laboratory allocation. However, the adopted modular design can allow a laboratory class to be scheduled on the departmental LTT as a course entity if and when necessary. The general inter-faculty courses meant for the faculty-independent large lecture theatres can also be treated separately as a department/faculty timetable with this design strategy.

Current challenges of LTTP in the university emanate from the increasing number of student intakes (due to increasing number of applicants seeking admission), introduction of new courses and restructuring of older ones, shortage or inadequate numbers of lecture rooms/laboratories, and increasing number of inter-departmental/faculty courses.

These make the LTP a harder problem to tackle as a result of large number of conflicting constraints.

In this paper, we present a generic model solution of the university LTTP based on GA metaheuristic. The GA implementation is defined such that it can be used by individual departments and/or incorporated into the entire faculty or the university at large. All that is required is to incorporate necessary additional requirements/ constraints into the implementation by expanding the data files specified for courses, classes and lecturers. To handle the constraints on non-overlapping of courses for a class, priority is given to the students of the same level as that designated for the course. Thus during allocation, 200-level students offering CSC202 are taking into consideration more than others. Little or no priority is given to students carrying over the course or lecturers who wish to teach in a particular classroom. Based on this premise, satisfaction of given hard constraints are more essential requirements for an optimal timetable than soft constraints in the case study. In the university, soft constraints introduced by students (e.g. proximity to faculty) are almost always neglected given arrangement of most classrooms around the faculty. It is thus assumed that requirements introduced by student entity are generally flexible as they can adjust their schedules to attend necessary lectures. The major hard constraint introduced by students is that of non-overlapping schedule for related classes. For example faculty courses for all 100-level students must not be scheduled together. Given this premise, the constraints considered in our experiment are as follows:

- Lecturers should not be doubly booked – that is a lecturer taken two or more courses must not have the courses fixed in the same time slot for the same day.
- The number of registered students for a course must not exceed the maximum capacity of a room. As much as it is important to observe this constraint, the present situation in the university with increased in admission quota allows for a little slacking of this requirement especially with regard to the large lecture theatres. Thus, the classroom capacity may be exceeded slightly, may be by not more than 5 per cent of its maximum capacity. However, the present study considers this requirement a strict hard constraint.
- Lecture rooms should not be doubly booked – that is two different courses must not be assigned to the same timeslot in the same lecture room simultaneously.
- Related classes should not be booked simultaneously. This implies that the same set of students taking two different courses must not have the courses allotted in the same timeslot. Generally, similar courses at the same level (e.g. CSC201 and CSC202) offered the same students must not be doubly booked to avoid clashes.

3. GA metaheuristic

GA (Goldberg, 1989) metaheuristic is employed to explore solutions to real-life LTP. GA is generally used in finding approximate solutions to many difficult problems arising in various fields of applications. GAs use biologically inspired techniques such as inheritance, mutation, natural selection and recombination (or crossover) in search of an optimal solution within the solution space. It is a probabilistic search algorithm that iteratively transforms a set (called a *population*) of mathematical objects, each with an associated fitness value, into a new population of offspring objects using the Darwinian principle of natural selection and operations that are patterned after naturally occurring genetic operations – that is crossover (recombination) and

mutation (Koza, 2007). One essential feature of GA is the encoding of potential solution to a specific problem in form of chromosome-like data structure. Recombination operators are then applied to this structure in such a way as to generate better offspring while still preserving critical information. Using this approach, variables are represented as genes on a chromosome. Through natural selection and the genetic operators, chromosomes with better fitness are found. Natural selection guarantees that chromosomes with best fitness will propagate in future populations. Recombination or crossover operator combines genes from two parent chromosomes to form a new chromosome with a high probability of having better fitness than the parents. Mutation alters one or more gene values in a chromosome from its initial state with the aim of finding a better solution. It can help to prevent population from stagnating at any local optima. This gives GA its characteristics generational improvement in the fitness of chromosomes with the eventual aim of creating chromosomes that contain optimized variable settings. As stated earlier, one essential aspect in the application of GA is the choice of representation of the chromosome otherwise referred to as the data structure. We present the data structure/representation adopted for the current problem in Figure 1. In the representation, Ci stands for Course i, $i = 1, 2, \ldots, m$, where m is the number of courses to be allocated and 0 indicates a free slot.

3.1 Solution representation

This work is basically concerned with the allocation of various courses into the given lecture rooms. As earlier stated, the main objective is to ensure that all stated hard constraints are met. To realize this objective, the data about the capacity of the various lecture rooms, the number of students registered for a course, lecturers' allocation to

Room 1					
Time	Mon	Tue	Wed	Thu	Fri
8-10	C1	0	0	0	0
10-12	C5	0	0	0	C2
12-1	C12	0	0	0	0
…	…	…	…	…	…
4-6	0	0	C9	0	0

……..

……..

Room N					
Time	Mon	Tue	Wed	Thu	Fri
8-10	0	0	C3	C6	0
10-12	C15	0	0	0	C16
12-1	C4	0	0	0	0
…	…	…	…	…	…
4-6	C17	0	C18	0	C20

Figure 1.
Solution representation for individuals in the population

courses, and other useful information are essential. In the case study, lectures are scheduled majorly within some timeslot in 5 days of the week, with 10 h per day. Our representation (see Figure 1) takes a timetable for each classroom as a chromosome. Each chromosome represents a potential timetable schedule (solution) for a particular classroom thus we have a population with chromosome length of N, where N is the number of rooms. Each gene within the chromosome contains information on what courses are scheduled in the given room for a particular timeslot. For the sake of convenience, we assume an even distribution of timeslots starting from 8 am to 6 pm. The entire timetable can thus be considered as an array of rooms. The timetable for a room is represented as a matrix. The timetable for an entire department, faculty or university is a collection of timetables for all the rooms. This representation ensures that a lecture room is not doubly booked since only one course can be allocated to a particular room thus satisfying one of the hard constraints.

The data sets needed to generate an LTT are stored in three different input files – *course, classroom* and *lecturer*. The file structures allow major hard constraints to be specified therein. For example, the course file contains the course code, class size, lecturer ID and related class number (a single digit specifying the level at which the course is offered) while the classroom file specifies the class name and the capacity. Other interactive inputs expected from the user include the number of rooms, lecturers and courses.

3.2 Fitness evaluation

The fitness function is designed to measure the degree of violation of given hard constraints as the iterative process of generating the timetable schedule progresses. Thus the fitness evaluation determines the number of lecturer doubly booked errors (when the same lecturer teaching different courses are allotted the same period for at least two courses), room too small errors (the lecture room capacity is smaller than the size of the class for a given course) and related class errors (students at the same level are assigned the same timeslot for at least two compulsory courses). Weights, which penalize the violations of the constraints, are assigned to each constraint. All hard constraints are allocated the same weight to ensure that none takes pre-eminence over the other during the experiment. We employed a linear combination of number of constraints violations as the measure of the fitness of individual chromosome within a population. The procedure examines and counts the number of constraint violations and compute the fitness value as follows:

$$f_c = \begin{cases} \dfrac{1}{z_c} & \text{if } z(c) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where

$$z(c) = \sum_{i=1}^{p} w_i g_i, \tag{2}$$

c represents the individual chromosome.

In Equation (2), g_i represents the number of constraint violations for a given constraint i while w_{i} is the weight associated to that constraint. The total number of

constraints is p while $z(c)$ is the summation of all the weighted constraint violations for chromosome c. The fitness of a timetable schedule (represented by chromosome c), given as f_c, is computed as in Equation (1). Since hard constraints are the major concern in the study, each of these is assigned equal weight. The current study set the value of w_i to 10. This fitness evaluation approach makes room for flexibility in case of any need to incorporate any soft constraint which can easily be added once appropriate, relatively lower weight, for the constraint has been assigned.

3.3 GA operators

The initialization procedure creates a random population of solutions for the GA operators to iteratively work with. This procedure checks for each course and room, if a given timeslot has a course fixed already. If so, a boolean variable is set otherwise the variable is reset. At the end, if the boolean variable remains false, a course is picked at random and allocated to the timeslot and the boolean variable is set. This process is repeated until a feasible initial population is generated.

Roulette wheel scheme is used in the experiment to select two parents for crossover operation. A single-point crossover strategy is used based on the user-defined probability and a generated random number, $r \in [0, 1]$. If the random number is less than the crossover probability, the selected parents are crossed. Mutation is carried out based on a user-defined probability to modify some genes in a given chromosome. The mutation is implemented by randomly generating another gene and swapping the random gene with the current gene. Where necessary, a repair algorithm is invoked to remap the generated offspring that are outside the search space back to the search space as defined by the given constraints. The algorithm ensures that the offspring produced by crossover and/or mutation do not have multiple bookings of classes.

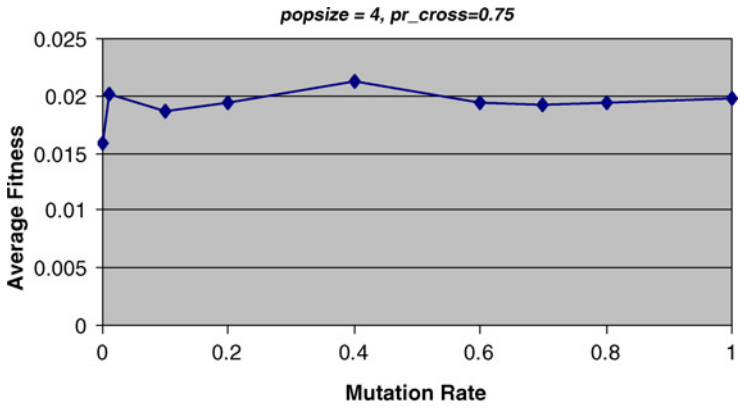
Once started, the GA runs until the specified number of generation is reached or an optimal solution is found with fitness value 0.0.

4. Experimental study

GA possesses explorative features, characterized by the population size and the rate of mutation; and exploitative features characterized by the type of selection used, the probability of crossover and the crossover operator itself (Petrovski, 2007). The effects of varying these parameters on the efficiency of GA are different. In our simulation experiment, we varied the major GA parameters in order to identify the factors that most significantly affect the performance of the algorithm.

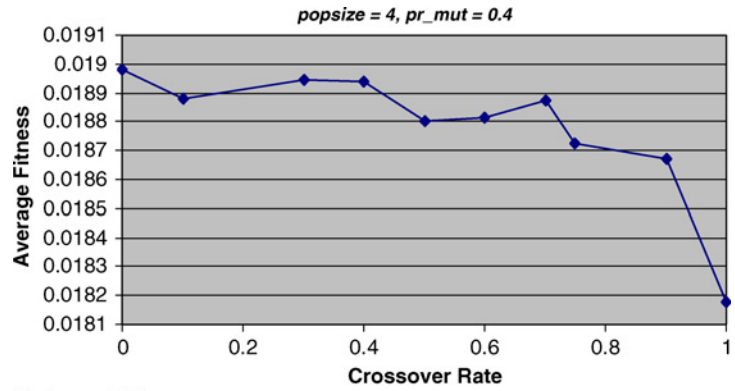
We conducted series of experiments to find out the combination of the parameters like *popsize* (population size), *pr_cross* (crossover rate) and *pr_mut* (mutation rate) that consistently give an optimal result when the algorithm is executed over a number of generations N. The test was conducted based on simulated data set for computer science programme with 13 lecturers, 4 classrooms and 24 courses in the institution. Note that the data set can be expanded, as stated earlier, by making additions to the respective input files. For each combination of parameters, the algorithm was executed a number of times and the average fitness value of returned solutions computed. We present some of the experimental results in graphical form in Figures 2-4. Sample input data are presented in Tables I and II.

After running the experiment for ($n = 200$) with $popzise = 4$; $pr\_cross = 0.75$, the optimal mutation rate was found to be between 0.3 and 0.4 (Figure 2). With the



Note: $n = 200$

Figure 2.
Average fitness vs mutation rate



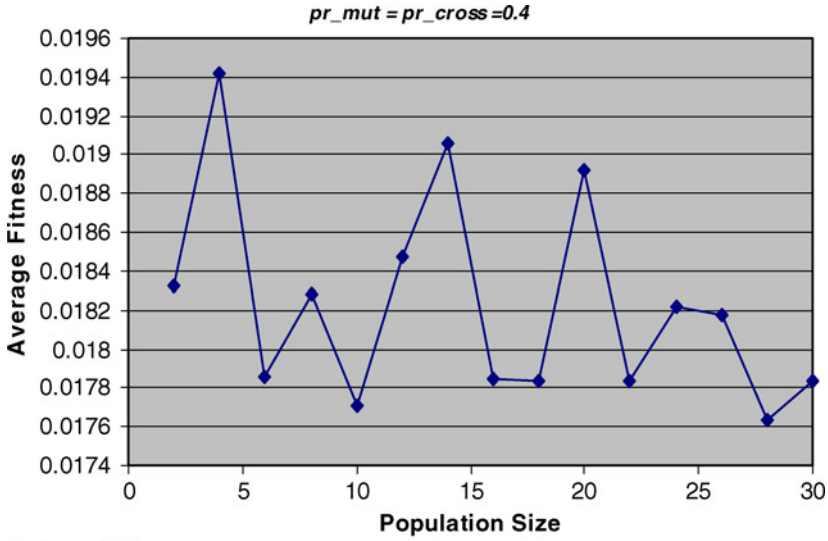
Note: $n = 200$

Figure 3.
Average fitness vs crossover rate

mutation set to zero the cost of the population stagnated. This was because without mutation no new alleles will be introduced into the population. It was observed that higher mutation rates permit too much exploration of the search space with the resultant slow convergence rate. High mutation rates also bring about too much diversity in the population set thus prolonging the execution time to optimality. Mutation rates that are too low tend to miss some near-optimal points.

Experimenting with crossover rate in a similar vein, we kept *popsize* at 4 and *pr_mut* at 40 while *pr_cross* was varied differently from 0.0 to 0.1. Crossover rate is found to have little effect on the rate of evolution of the algorithm for this study (Figure 3). This is because after some generations, all chromosomes tend to have almost the same fitness values. This might be as a result of the selection strategy. Roulette wheel selection tends to select chromosomes with high fitness thereby causing the population to possess chromosomes with almost the same fitness value. Crossover rate between 0.3 and 0.4 gives near-optimal results (Figure 3).

For the population size, we kept *pr_mut* and *pr_cross* at 0.4 and varied population size from 2 to 30. The experiment was repeated thrice ($n = 200$) and the average fitness for each generation calculated. An optimal population size was found at 4 (Figure 4). Actually, large population sizes result in a long waiting time for significant



Note: $n = 200$

Figure 4.
Average fitness vs population size

Index	Code	Class size	Lecturer ID	Class related no.
1	CSC100	120	1	1
2	CSC201	70	4	2
3	CSC202	70	7	2
4	CSC204	70	8	2
5	CSC203	70	3	2
6	CSC301	50	1	3
7	CSC302	50	2	3
8	CSC303	50	3	3
9	CSC304	60	8	3
10	CSC305	60	6	3
11	CSC306	60	4	3
12	CSC307	60	11	3
13	CSC308	60	11	3
14	CSC311	50	5	3
15	CSC321	70	1	3
16	CSC401	50	12	4
17	CSC402	50	2	4
18	CSC433	50	12	4
19	CSC431	50	5	4
20	CSC430	50	10	4
21	CSC421	50	6	4
22	CSC411	30	13	4
23	CSC422	20	9	4
24	CSC423	50	10	4

Table I.
Sample course file

improvement in the algorithm thus higher population sizes were seen to evolve chromosomes with lower fitness values.

A generated sample timetable schedule in one of the generations is presented in Figure 5. A more detailed outline in another generation is presented in Table III.

5. Conclusion and further work

A GA metaheuristic for a module of the LTTP as obtained in a Nigerian university was presented. Results obtained in some of the simulation experiments were presented. The design of the fitness function makes room for expansion of the LTT module to incorporate other departments within a faculty and/or university. This can be done by modifying the input constraints files. Fitness evaluation was based on the degree of violation of given constraints hence incorporation of other soft constraint could be possible as the university deem it fit by simply adding them to the function and assigning appropriate weight. The size of the weights must be such that determine whether selection pressure in the direction of the constraints should be less or more (depending on whether the constraint is soft or hard). Soft constraints can easily be included whenever it is necessary but allocated lower weight.

In a recent work, we are applying some probabilistic adaption and hybridization with other metaheuristics as well as integrated crossover rule in order to improve performance of the algorithm. Also application of already proposed niching method (Mahfoud, 1995) has been explored. Furthermore, another set of selection and crossover strategies can be used in search of an enhanced performance and better parameter setting. Other alternative representation might be to consider the timetable as an array of courses or lecturers. Set-based representation could also be explored.

Index	Class name	Capacity
1	E303	70
2	E304	70
3	E203	50
4	L026	150

Table II.
Sample classroom file

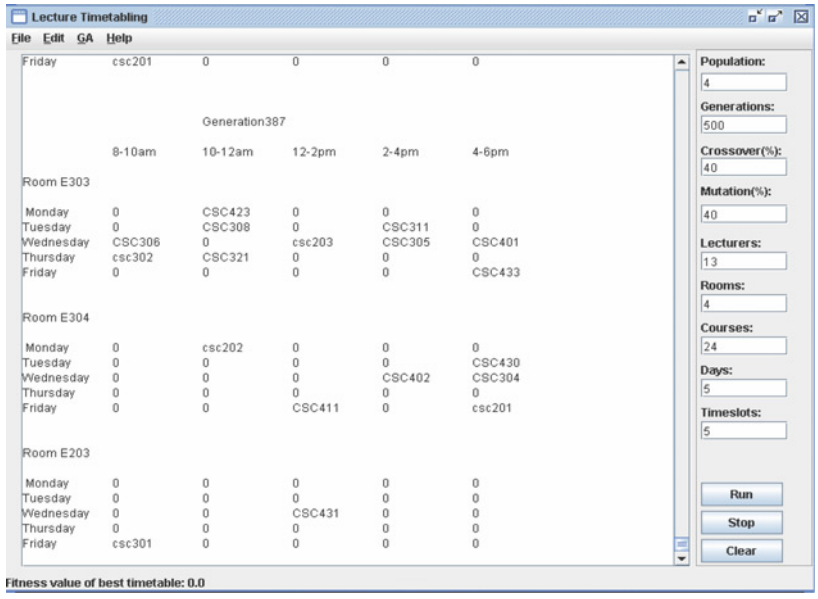


Figure 5.
Sample of generated LTT at generation 387

	8-10 am	10-12 am	12-2 pm	2-4 pm	4-6 pm
Room E303					
Monday	0	0	0	0	CSC305
Tuesday	CSC401	CSC321	0	0	CSC301
Wednesday	0	0	0	CSC423	0
Thursday	0	0	0	0	CSC201
Friday	0	0	0	0	0
Room E304					
Monday	CSC202	0	0	CSC304	0
Tuesday	0	CSC431	0	0	0
Wednesday	CSC204	0	0	CSC303	0
Thursday	0	0	0	CSC308	0
Friday	0	CSC307	0	CSC203	0
Room E203					
Monday	CSC433	0	0	0	0
Tuesday	0	0	0	0	0
Wednesday	CSC302	CSC422	0	0	CSC421
Thursday	0	CSC411	0	0	0
Friday	0	0	0	0	0
Room L026					
Monday	0	CSC311	CSC402	0	CSC430
Tuesday	0	0	0	0	0
Wednesday	0	0	0	0	CSC100
Thursday	0	0	0	0	0
Friday	0	0	0	0	CSC306

Table III.
Generated sample schedule in one of the generations

References

Adewumi, A.O., Ihemedu, N. and Ayeni, J.O.A. (2005), "An evolutionary-based approach to university course time-tabling problem", *First Annual Research Conference and Fair, Poster No. 9.11, University of Lagos, 26 October*.

Adewumi, A.O., Fasina, E.P., Ayeni, J.O.A. and Ali, M.M. (2008), "A genetic algorithm metaheuristic for a multi-stage hostel space allocation problem", *Proceedings of the 18th Triennial International Federation of Operation Research Society Conference (IFORS 2008)*, *Sandton*, pp. 12-13.

Dammak, A., Elloumi, A. and Kamoun, H. (2006), "Lecture and tutorial timetabling at a Tunisia University", in Burke, E.K. and Rudova, H. (Eds), *Proceedings of the of the Sixth International Conference on Practice and Theory of Automated Timetabling (PATAT 2006)*, *Masaryk University, Brno*, pp. 384-90.

Gendreau, M. (2003), "An introduction to Tabu Search", in Kochenberger, G. and Glover, F. (Eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers, Dordrecht, pp. 37-54.

Glover, F. and Kochenberger, G.A. (Eds) (2003), *Handbook of Metaheuristics*, Kluwer Academic Publishers, Dordrecht.

Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison Wesley, Boston, MA.

Koza, J.R. (2007), "Genetic algorithms and genetic programming", presentation at the Department of Electrical Engineering, School of Engineering, Stanford University, Stanford, CA, available at: www.smi.stanford.edu/people/koza/ (accessed August).

Kreher L. Donald and Stinson R. Douglas (1999), *Combinatorial Algorithms: Generation, Enumeration, and Search*, CRC Press, Boca Raton, FL.

Mahfoud, S.W. (1995), "Niching methods for genetic algorithms", PhD thesis, Illinois Genetic Algorithm Laboratory, University of Illinois at Urbana-Champaign, IL.

Petrovski, A., Wilson, A. and Mccall, J. (2007), "Statistical identification and optimisation of significant GA factors", Technical paper, The Robert Gordon University, School of Computer and Mathematical Sciences, Aberdeen.

Schaerf, A. (1999), "A survey of automated timetabling", *Artificial Intelligence Review*, Vol. 13, pp. 87-127.

Vesterstrøm, J. (2005), "Heuristic algorithms in bioinformatics", PhD thesis. Bioinformatics Research Center (BiRC), Department of Computer Science, Faculty of Science, University of Aarhus, Aarhus.

Vesterstrøm, J. and Riget, J. (2002), "Particle swarms: extensions for improved local, multi-modal, and dynamic search in numerical optimization", master's thesis. Department of Computer Science, University of Aarhus, Aarhus.

White, G.M. and Zhang, J. (1998), "Generating complete university timetables by combining tabu search with constraint logic", Burke, E. and Carter, M. (Eds), *Lecture Notes in Computer Science*, vol. 1408, Springer Verlag, Berlin and Heidelberg, pp. 187-98.

Wren, A. (1996), "Scheduling, timetabling and rostering – a special relationship?", in Burke, E.K. and Ross, P. (Eds), *The Practice and Theory of Automated Timetabling: Selected Papers from the First International Conference on the Practice and Theory of Automated Timetabling PATAT*, (Lecture Notes in Computer Science, vol. 1153), Springer, Heidelberg, pp. 46-75.

Zampieri, A. and Schaerf, A. (2006), "Modeling and solving the Italian examination timetabling problem using tabu search", in Burke, E.K. and Rudova, H. (Eds), *Proceedings of the Sixth International Conference on Practice and Theory of Automated Timetabling (PATAT 2006)*, *Masaryk University, Brno*, pp. 487-91.

Corresponding author
Aderemi O. Adewumi can be contacted at: laremtj@gmail.com