International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 3, March - 2014

Use of Genetic Algorithms in Rigorous University Timetabling

[1] Chikwiriro Hilton, [2]Chaka Pharaoh, [3]Mavhemwa Prudence M , [4]Ndumiyana David
Computer Science Department
Bindura University of Science Education, Zimbabwe
Bindura, Zimbabwe

Abstract—**The timetable problem is a class of computationally NP-complete decision problems; this means that no polynomial-time algorithm or method of solving it is in a reasonable amount of time is known. Manual courses timetabling is a complex administrative and time consuming task for the timetablers in the faculty, it is sometimes impossible to produce totally feasible solution given considering the tightness of the timetable constraints. The research seeks to assess user perception on automated course timetabling system which use multi-phase genetic algorithm, in solving their respective departments' course timetable problems. This research explores case studies of others institutions timetabling/scheduling problem as well as other approaches of scheduling and optimisation problems which are linked to the timetable problem. A course timetable system was designed and implemented using the multi-phase genetic algorithm and knowledge about the timetable constraints was incorporated into the genetic operators, fitness function and in the repair strategy.**

Keywords—Course class, Fitness function, hard constraints: Multi-phase, Scheduling, Soft Constraints, Timetabling

I. INTRODUCTION

Sometimes, the words timetable and schedule are loosely used as if were synonymous, but, there can be certain distinctions between these terms observed in the literature [1][2]. A timetable shows when particular events are to take place. For example a class timetable shows when particular events are to take place and in such a way try satisfying as nearly as possible a set of desirable objectives. A schedule will normally include times at which activities are to take place, statements as to which resources will be assigned where and work plans for individual personnel or machines. It is in such a way as to minimise the total cost of some set of the resources used. The margin between the two words may be somehow trivial as in their broadest sense they solve practical problems relating to the allocation, subject to constraints, of resources to objects being placed in space-time, using or developing whatever tools may be appropriate. According to [20], a timetable is an optimum allocation of activities, actions or events to a set of objects in space-time matrix to satisfy a set of desirable constraints. A typical constraint is a restriction that activities which are using the same resource such as a room, machine and an operator can not overlap in time. This is because a resource maybe of a certain capacity, restricting the number of activities which can use it at the same time.

The nature of the timetable problem may vary from one university to another and for the past years researches have been conducted to consider an approach suitable for the institutions' problem context. For instance, the Purdue University, in Indiana for course timetabling problem of **Fall 2004 Semester** [7] and [8] had different demand from that of the institution under study. Their lecture times are usually one hour and at most one hour thirty minutes per week and lectures extend from day time to evening classes this increases their timetable slots to accommodated many lectures, also each department owns several home rooms which are not heavily interleaved. All these differences reflect that although Purdue offers a wide range of degree programs and enroll students in several folds compared to Bindura University; their academic resources demand is not tightly restrictive.

This research study is an investigation on university course timetabling problem and an attempt to give an optimal approach for the Bindura University of Science Education (BUSE), faculty of science timetabling problem.

II. THE TIMETABLING/ SCHEDULING PROBLEM

Timetabling is a widely studied area and many potentially useful algorithms have been developed for solving the university course timetabling problem, as evidenced by several surveys[15][21]. The major differences between many of the timetable problems studied and their real life counter parts are the additional complexity imposed by course structures, the variety of constraints considered and the distributed responsibility for information needed to solve such problems at an institution level.

A. *CASE STUDY 1: PURDUE UNIVERSITY COURSE TIMETABLE PROBLEM*

The university is located in West Lafayette, Indiana. In the institution, a timetable for large lecture classes was constructed by a central scheduling office in order to balance the requirements of many departments offering large classes that serve students from across the university. Smaller classes, usually focused on students in a single discipline, were timetabled by "schedule deputies" in the individual departments. Such a complex timetabling process, included subsequent student registration, took a rather longtime.

The data set in consideration is for the **Fall 2004 Semester** [7] and [8] and the case study in consideration is a real-life large scale problem that includes features of over-constrained as well as optimisation problems. The goal was to timetable about 830 classes, forming almost 1800 meetings, having a high density of interaction that were required to fit within 50 lecture rooms with capacities up to 474 students. Room availability was a major constraint for Purdue. Overall utilization of the time available in rooms exceeded 78%; moreover, it was around 94% for the four largest rooms. About 90,000 courses requested by almost 30,000 students were also to be considered. 8.4% of class pairs had at least one student enrolment in common.

To minimize potential time conflicts, Purdue had historically subscribed to a set of standard meeting patterns. With few exceptions, 1 hour * 3 day per week classes met on Monday, Wednesday and Friday at the half hour (7:30, 8:30, 9:30, ...) one and half hours each two days per week classes met on Tuesday and Thursday during set time blocks. Two or three hours per day each week, classes were also expected to fit within specific blocks. Generally, all meetings of the class were to be taught in the same location. Such meeting patterns were of interest to the problem solution as they allowed easier changes between classes having the same or similar meeting patterns.

Another aspect of the timetabling problem that was to be considered was the need to perform student sectioning. Most of the classes in the large lecture problem (about 75%) corresponded to single-section courses. Here there was exact information about all students who wished to attend a specific class. The remaining courses were divided into multiple sections. In this case, it was necessary to divide the students enrolled in each course into sections that would constitute the classes.

B. *The proposed algorithm: Iterative Forward Search*

The proposed algorithm was based on ideas of local search methods [18]. However, in contrast to classical local search techniques, it operates over feasible, though not necessarily complete solutions. In such a solution, some variables can be left unassigned. Still all hard constraints on assigned variables must be satisfied. Similarly to backtracking based algorithms, this means that there are no violations of hard constraints.

The framework based on the IFS algorithm was written in Java and was also extendable to be used for solving lecture timetabling problems as well as for other constraint-based problems. In order to present the general purpose of this algorithm, it is described for solving general finite constraint satisfaction and optimisation problems.

C. *Analysis of IFS algorithm*

During each step, a variable A is initially selected. Typically an unassigned variable is chosen like in backtracking-based search. An assigned variable may be selected when all variables are assigned, but the solution found so far is not good enough for example, when there are still many violations of soft constraints.

Once a variable A is selected, a value *a* from its domain DA is chosen for assignment. Even if the best value is selected, its assignment to the selected variable may cause some hard conflicts with already assigned variables. Such conflicting assignments are removed from the solution and become unassigned. Finally, the selected value is assigned to the selected variable.

The algorithm attempts to move from one (partial) feasible solution to another via repetitive assignment of a selected value *a* to a selected variable A. During this search, the feasibility of all hard constraints in each iteration step is enforced by unassigning the conflicting assignments η (computed by function conflicts). The search is terminated when the requested solution is found or when there is a timeout expressed, for example, as a maximal number of iterations or available time being reached. The best solution found is then returned.

D. *CASE STUDY 2: UNIVERSITY KEBANGSAAN MALAYSIA (UKM) EXAMINATION TIMETABLE PROBLEM*

Schedulers at the UKM were chief decision makers who applied the examination assignment procedure, based on their experience with a little guidance from a computer application programmed to aid timetable clash avoidance. By then, they did not consider students sitting two/three consecutive exams in a day instead they only took into account that exams are spread evenly and fairly throughout the timetable. In trying to achieve this, the size/complexity of the problem made it unrealistic therefore, they became only concerned with the constraint of not assigning a student to sit for more than one exam in a given timeslot. Even this seemingly easy procedure usually could take the manual schedulers more than two weeks. After circulating the exam timetable to students, the schedulers would invariably receive many complaints from students and lecturers. When students complained about sitting three consecutive exams in a day they were scheduled, a new timeslot would be added on a Saturday and the middle exam is scheduled to this new timeslot with an emphasis on making as few adjustments to the rest of the timetable as possible. This incurred extra overhead costs.

The following was the approach and experience incurred in solving the examination timetabling problem for Semester one of the year 2006 at UKM. The dataset (UKM06-1) was preprocessed based on the supplied data which contained 818 exams, 14,047students, 75,857 enrollments, 42 timeslots and 15 exam days this excluded weekends. The UKM06-1 dataset was held in four text files: UKM06-1.stu, UKM06-1.slt, UKM06-1.rom and UKM06-1.isl, which represented student enrollment, slot, room and isolated exams definition, respectively.

The dataset had three weeks examination period. Each week having five exam days Monday to Friday and each day having three timeslots morning, afternoon and evening, except Fridays which had two timeslots morning and evening only.Due to the complexity, the problem was partitioned into two sub problems. That is, it assigned exams to timeslots the same as the capacitated examination timetabling problem and the room assignment problem.

Before assigning exams to timeslots the supplied data required some pre-processing. Firstly, solved anomalies in the data set such as removal of courses with no exams and combined exams that have to be scheduled together into a single exam. The output of the first stage will be used as input to subsequent stages.

E. The proposed heuristic procedure for the examination timetabling problem

The solution to the UKM examination timetable problem presented a heuristic procedure which was called Greedy Least Saturation Degree (G-LSD). It used the term "greedy" because the heuristic attempted to assign each exam to the best timeslot which satisfied all the hard constraints. There was need to randomly assign exams to timeslots when the exam has no conflict with the exams that had already been scheduled. While assigning exams to timeslots, it also ensured that all hard constraints were satisfied, this was possible through an adaptation of the least saturation degree heuristic for classical graph colouring problem.

F. Analysis of the G-LSD heuristic procedure

In the initialisation step, all exams in B were reset. That is, the number of available timeslots for each exam were set to the maximum available slot and the exam's status changed to unscheduled and was copied into the unscheduled exam set B′={E1',E2′….,EN′}. The heuristic first arranged the unscheduled exams in B′ in non-decreasing order of the number of available timeslots, then in non-increasing order of the number of conflicts they have with other exams (in B) and, finally, by non-increasing order of the number of student enrollments. The then heuristic chooses the first exam in B′, Ei′ and assigns it to the best timeslot and subject to the total number of students assigned to the timeslot that did not exceed the maximum seat capacity. However, when all slots were available for Ei′, that is the exam had no conflict with the exams that had already been scheduled, the heuristic randomly chose a timeslot for Ei′. The idea was to allocate the best timeslot for Ei′, so as to obtain different solution for each run. While assigning exams to timeslots, there was also a need to ensure a clash free schedule and larger exam (student enrollment > = 400) were assigned to earlier timeslots the first two weeks if it were possible. This was done based on discussions with UKM registry officers as they usually assigned larger exams to earlier timeslots in order to give longer time for marking larger exams.

After assigning exam Ei′ to the timeslot, the algorithm would update the appropriate exam details such as timeslot index, number of available timeslots. In B, reduce the number of available timeslots for exams in B′ accordingly. Then eliminate Ei′ from B′ and repeated step 2.1 to 2.6 until all the exams were scheduled, or until Ei′ could not be assigned to any available timeslot. If this occurs, the algorithm stops the process and start again (steps 1 to 3). After assigning all exams to timeslots, it would verify that all hard constraints are satisfied. If the solution is feasible, the process ends.

In many cases, the algorithm produced solutions indicating no students sitting for three consecutive exams in a day. However, the solution was bound to be rejected (infeasible) if there were students sitting three consecutive exams in a day. Since this was a constructive heuristic, it would stop the process when it obtains a feasible solution. The algorithm could extend this procedure to produce many feasible solutions and return the best solution found by repeating step 1 to 3 for a given number of iterations.

III. GENETIC ALGORITHMS

Unlike most methods of combinatorial optimisation, GAs did not initially have an underlying mathematical model. As such, they were limited to demonstrating a number of famous mathematical problems such as the travelling sales person problem and the k-armed bandit problem before tackling more practical issues [9]. By 1989 when David E Goldberg released the seminal "GAs in search, optimisation and machine learning", the field had begun the brightest phase of its career that of being applicable to real world problems [9].

In today's GA application, typical problems can be phrased so as to require the minimising or maximising of some function. In particular, where this function is dependent upon a great many variables, such that more conventional methods are out of their depth, evolutionary methods become attractive [10]. Particularly noteworthy applications of GAs include the solving of pipe network optimisation problems [11] transportation problems [12] conformational analysis of DNA [9] image processing and machine learning [13] and of course, scheduling problems [14][13].

GAs are advancing by containing less of a close metaphor with natural evolution instead they are becoming more conforming only to that essence of evolution which allows them to function. For example, data structures are replacing binary numbers as the most common form of representing genetic material. In modern GAs, chromosomes are rarely fully encoded [9].

The algorithm was developed by Professor John Holland at the University of Michigan in the 1960s. The GA is a probabilistic search algorithms that iteratively transforms a set (a population) of mathematical objects typically fixed-length binary character strings, each with an associated fitness value, into a new population of offspring objects using the Darwinian principle of natural selection. The GA belong to the larger class of evolutionary algorithms, essentially, they are a method of searching problems for a solution by means of simulated evolution. The algorithm uses the biological principles of selection, crossover and mutation to perform a search in often complex and big search spaces and it attempts to find global solutions, while avoiding local optimal solutions [15].

The processes loosely based on natural genetic operators which are repeatedly applied to a population of binary strings representing potential solutions. Over the time, the number of above-average individual's increases and highly-fit building blocks are combined from several fit individuals to find good solutions to the problem at hand. In the process, a population of candidate solutions referred to as phenotype which is involved in an optimisation problem is evolved towards better solutions. Each candidate solution has a set of properties referred to as chromosomes or genotype which can be altered through the evolution process in search of a fit solution.

The evolution starts from a population of randomly generated individuals and occurs throughout the next generations. In each generation, the fitness of every individual in the population is evaluated, fit individuals are stochastically selected from the current population and each individual's genome is modified or recombined to form a new pattern. The GA pseudo-code is shown below.

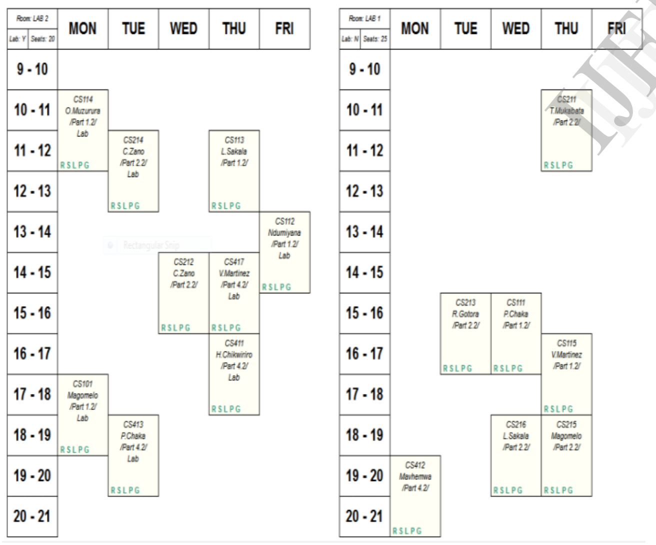
<u>The genetic algorithm (GA) pseudo-code</u>

```
Choose initial population
 Evaluate each individual's fitness
 Determine population's average fitness
 Repeat { Select best-ranking individuals to reproduce
             Mate pairs at random
             Apply crossover operator
             Apply mutation operator
             Evaluate each individual's fitness
             Determine population's average fitness
Until terminating condition (until at least one individual
has the desired fitness or enough generations have passed)
```

IV. RESEARCH METHODOLOGY

A. *SYSTEM DESIGNS*

The system design describes technical details of the used solution approach and its implementation. There was need to design a course timetable system for the BUSE, faculty of science to address the problem mentioned in chapter one. Figure 1 below is a screen-shot showing the layout which was used as a guide in the design of the multi-phase GA timetable system.



Room: LAB 2 Lab: Y Seats: 20	MON	TUE	WED	THU	FRI
9 - 10					
10 - 11	CS114 O.Muzurura /Part 1.2/ Lab RSLPG				
11 - 12		CS214 C.Zano /Part 2.2/ Lab RSLPG		CS113 L.Sakala /Part 1.2/ RSLPG	
12 - 13					
13 - 14					CS112 Ndumiyana /Part 1.2/ Lab RSLPG
14 - 15			CS212 C.Zano /Part 2.2/ RSLPG	CS417 V.Martinez /Part 4.2/ Lab RSLPG	
15 - 16					
16 - 17				CS411 H.Chikwiriro /Part 4.2/ Lab RSLPG	
17 - 18	CS101 Magomelo /Part 1.2/ Lab RSLPG				
18 - 19		CS413 P.Chaka /Part 4.2/ Lab RSLPG			
19 - 20					
20 - 21					

Room: LAB 1 Lab: N Seats: 25	MON	TUE	WED	THU	FRI
9 - 10					
10 - 11				CS211 T.Mukabata /Part 2.2/ RSLPG	
11 - 12					
12 - 13					
13 - 14					
14 - 15					
15 - 16		CS213 R.Gotora /Part 2.2/ RSLPG	CS111 P.Chaka /Part 1.2/ RSLPG		
16 - 17				CS115 V.Martinez /Part 1.2/ RSLPG	
17 - 18					
18 - 19			CS216 L.Sakala /Part 2.2/ RSLPG	CS215 Magomelo /Part 2.2/ RSLPG	
19 - 20	CS412 Mavhemwa /Part 4.2/ RSLPG				
20 - 21					

Figure 1: A timetable system design screen-shot

B. *MULTI-PHASE GA DESIGN FRAMEWORK*

Multi-phase concept was used to break down the timetabling problem into manageable chunks which were assumed easier to work with. The algorithm could then be applied to solve the reduced problem in separate distinct phases.

By nature no standard GA can take constraints into account [17], but According to Dowsland (2000), incorporation knowledge about the constraints into GA operators, the algorithm, can exploit the power of the algorithm and could be used for solving constraint satisfaction problems. This research study adopted the concept used by [19], which proposed a multi-stage operation-based GA in dealing with the flexible job scheduling problem.

At a high conceptual level the algorithm combines two methods, one, a direct and the other and indirect GA constraint handling. The timetabling process was broken into logically distinct phases, as the name multi-phase GA suggests.

Direct constraint handling: leaves the timetable object variables as they are to 'adapt' and the GA to enforce them. This implies that violating them is not reflected in the fitness function, but in the solution generation completeness.

Indirect constraint handling: circumvents the problem of satisfying constraints by incorporating them in the fitness function (*f*) such that the *f* optimally implies that the constraints are satisfied through the use of GA to find an optimal solution.

The aforementioned modification was for the purpose of splitting the process into four phases which were conveniently categorised as phases one (1), two (2), three (3) and four (4). The breakdown allowed the algorithmic design and implementation of a solution for a complex problem to be of a manageable size at each time instance.

NB: Note that direct and indirect constraint handling methods were both used to solve the course scheduling in separate phases.

Before assigning a course to timeslots the supplied data required some pre-processing. Firstly, by solving anomalies in the data set such as removal of classes without defined lecturers, associated students or with incorrect instantiation from the input.

V. IMPLEMENTATION

The multi-phase GA timetable system was developed using Microsoft Visual C++ integrated development environment. The course scheduling system is a desktop application program which executes on a Microsoft Windows platform.

Visual C++is an object oriented programming language supporting object oriented principles and paradigms (encapsulation, inheritance and polymorphism).

The timetable system design was through interweaving different loosely coupled C++ classes, following software engineering principles to keep fewer dependencies that is loose coupling between modules. This enabled easy design modifications of the source code. Figure 2 below is a diagram showing the GA schedule system architecture.

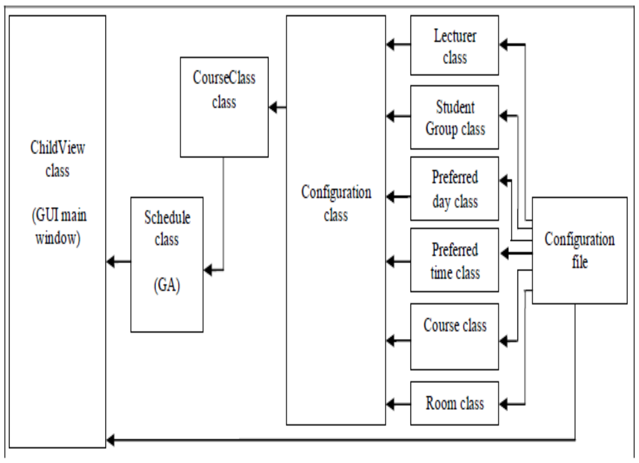


Figure 2: Multi-phase GA schedule system architecture

The course scheduling system has the following object classes and which do the following:

- Lecturer class- handles data about the lecturer object attributes
- Pdayclass- handles data about week days object attributes
- Ptimeclass- handles data about day time object attributes
- Student Group class- handles data about the student groups object attributes
- Course class- handles data about the course object attributes
- Room class- handles data about lecture room object attributes

The *Configuration* class obtains timetable data objects (rooms, courses, day, time, lecturers and student groups)from the configuration file and parse it into the system.

The '*CourseClass'* object class is a logical representation of a class which is ready to be scheduled to an event day and time. The class is a controller of the timetable objects data.

The *Schedule* class performs the assignment of course classes today and time slots in the timetable vector. The `ScheduleObserver class` contained in the *Schedule* class handles events that are triggered by the algorithm during execution such as: when the algorithm finds new best chromosome or when state of execution of algorithm is changed. It also functions to block caller's thread until the execution of the algorithm is finished using the `WaitEvent()` method.

When the configuration file is parsed, a timetable vector frame is loaded by the *ChildView* class into the main window. The *Schedule* class invokes the GA and the course scheduling process commences. The algorithm execution stops when the stopping criterion is met and this frees the system resources. A timetable solution is displayed in the main window showing the computed fitness function and the number of generations produced during the process.

A. THE MULTI-PHASE GA ALGORITHM

A complete cycle of genetic operations (selection, crossover and mutation) forms a generation of partial timetable solutions and for each generation, the algorithm performs two basic operations:

1. Randomly selects N pairs of parents from the current population and produces N new chromosomes by performing a crossover operation on the pair of parents.
2. Randomly selects N chromosomes from the current population and replaces them with new ones. The algorithm does not select chromosomes for replacement if it is among the best chromosomes in the population.

The two operations are repeated until the best chromosome reaches a fitness value equal to 1,this indicates that all classes in the schedule meet the requirements. The algorithm keeps track of the best chromosomes in the population and guarantees that they are not going to be replaced while they are among the best chromosomes. Below is the pseudo code for the Multi-phase GA algorithm.

B. THE MULTI-PHASE GA PSEUDO CODE

```
PHASE 1(Preprocessing)
Load all constraint data from a configuration file.
While the population size is less than the maximum:
PHASE 2 (Initialisation)
{
Create a new timetable with no classes booked to it.
Repair the new timetable by using the constraint data.
Evaluate the cost of the new timetable by using the constraint data.
Enter the new timetable into the population.
    }
PHASE 3 (Scheduling)
(Apply genetic operators)
Selection -> while the cost of the best timetable is greater than zero:
            {
Discard a portion of costly timetables.
Repeat until the population size is maximum:
{
                Breed a new timetable.
Crossover -> Apply the crossover operator to the new timetable
Mutation -> Apply mutation operator to the new timetable.
PHASE 4(Heuristic repair strategy)
Repair the new timetable by using the heuristic repair strategy based on constraint data.
Evaluate the cost (average fitness) of the new timetable by using constraint data.
Enter the new timetable into the population.
                }
            }
```

C. POPULATION INITIALISATION AND SELECTION

The fitness-based selection is used in the population selection process. This is a kind of parent selection where each chromosome has a chance of being selected that is directly proportional to its fitness value.

When using the GA to solve a combinatorial optimization problem, it is important to map out how to represent a solution of the problem as a chromosome. Figure 3 below is a diagram showing the chromosome representations in a vector data structure.

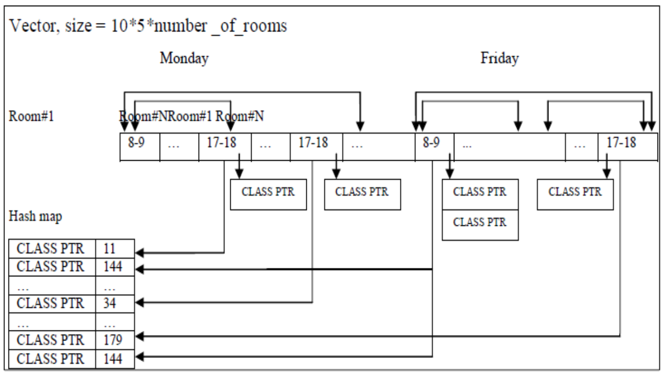


Figure 3: Chromosome representation in a vector

The diagram above shows chromosome representation, there is a time-slot for each hour for every room in each working day. Assuming that classes start at 0800hours and finish at 1600hours, that is a total of 10 hours and the working days under consideration are Monday to Friday, thus 5. We can use a standard vector `(std::vector)` with a size `10*5*number_of_rooms`. The vector slot must be a standard list `(std::list)` as during the system execution, the algorithm requires temporal storage of multiple course classes allocated in the same time-slot.

The system uses hash maps for storage of vector data variables during execution. Hash maps are data structures used to implement associative arrays which are structures that map strings to values. During the scheduling process, the algorithm uses hash maps for rapid access of vector positions (chromosomes). There is also a hash map that is used to store the first time-slot at which a class begins, that is its position in the vector from the address of the class' object. Each hour of a class has a separate entry in the vector, but there is only one entry per class in the hash map, this intuitively resolve clashes. In cases where the key of a new item match the key of an old item, a hash collision occurs, typically this erases the old item and overwrites it with a new item in order to maintain consistency so that every item in the table have a unique key. Chromosomes are represented in the C++ `Schedule` class, the class stores the representation of a class schedule in following two attributes:

1. Time-space slots, one entry represents one hour in one lecture room
   vector<list<CourseClass>> _slots;*
2. The class table for chromosome is used to determine the first time-slot used by a course class
   hash_map<CourseClass, int> _classes;*

The chromosome parameters used by the GA are as follows:

1. The number of crossover points of parent's class tables
   int _numberOfCrossoverPoints;
2. Number of classes that are moved randomly by single mutation operation
   int _mutationSize;
3. Probability that crossover will occur
   int _crossoverProbability;
4. Probability that mutation will occur
   int _mutationProbability;

The above parameters are used to make a prototype of chromosomes also for making new global instance of an algorithm.

D. CROSSOVER OPERATOR

The crossover operation combines data in the hash maps of two selected parents and creates a vector of slots according to the content of the new hash map. The operation splits hash maps of both parents in parts of random sizes which are defined by the number of crossover points which are defined in the chromosome's parameters. Then, it alternately copies parts from parents to the new chromosome and forms a new vector of slots. Figure 4 below is a diagram showing abstract representation of the crossover operation in the hash maps.

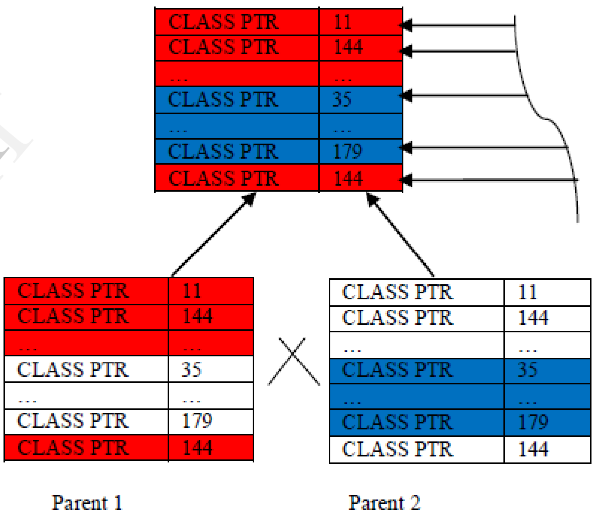


Figure 4: Crossover operation in the hash map

E. MUTATION OPERATOR

The mutation operation randomly takes a class and moves it to another chosen slot. The number of classes which are to be moved in a single operation is defined by the mutation size in the chromosome's parameters.

F. THE GA FITNESS FUNCTION

Chromosomes store their fitness values and parameters which are used by genetic operators as shown in the below C++ implementation.

- Fitness value of chromosome
  float _fitness;
- Flags raised for class requirements satisfaction
  vector<bool> _criteria;

The fitness value is represented by a single-precision floating point number in the range of 0 to 1. The closer the fitness value to the fitness upper bound, the better the optimal solution is.

Calculation of fitness isbased on score value and is represented as:

`schedule_score/maximum_score`, *and* `maximum_score` is `number_of_classes*5.`

`The C++ implementation is shown below:`

```
_fitness = (float)score / (Configuration::GetInstance().GetNumberOfCourseClasses() * DAYS_NUM );
```

The criterion employed by the system for the fitness function evaluation is as follows:

- Each class can have 0 to 5 points.
- If a class uses a spare lecture room, its score is incremented.
- If a class requires academic resources and is located in the room with them, or it does not require them, the score of the class is incremented.
- If a class is located in a lecture room with enough available seats, the score is increment.
- If a lecturer has no other classes at that time, the class's score is incremented.
- If any of the student groups that attend a class do not have any other classes at the same time, the class score is incremented.
- If a class breaks a rule at any time-space slot that it occupies, its score is not incremented for that rule.
- If a class is scheduled for a particular empty slot day or time, a class score is incremented.
- The total score of a class schedule is the sum of points of all classes.

A timetable under consideration has a set of constraint to be satisfied during the timetable processing and these are divided into two: hard and soft constraints. The two are defined as follows:

Hard constraints: Are restrictions, of which, in any working timetable, there will be no breaches. For example, a lecturer cannot be in two places at once. A list of hard constraints considered in the system timetabling is shown below:

- A class can be placed only in a spare classroom.
- No lecturer or student group can have more then one class at a time.
- A classroom must have enough seats to accommodate all students.
- To place a class in a classroom, the classroom must have laboratory equipment (computers, in our case) if the class requires it.

Soft constraints: These are limitations which may be broken, but of which breaches must be minimised. For example, classes should be booked at a preferred day and time. Lists of soft constraints under consideration is below:

- Preferred time of class by lecturers.
- Preferred classroom by lecturers.
- Distribution in time or space of classes for student groups or lecturers.

The timetable input data sets consisting of all information about lecturers, lecture rooms, student groups, courses and preferences were obtained from the timetable committee. The timetable system requires these constrained data to be loaded into the system also that the user should be able to modify them for example by adding a new course or change date and time preferences for a particular course schedule.

Each time the system executes it loads all the constraint data from a configuration file which is a text file with a ".cfg" extension. Each different *.cfg file describes a unique department such as Chemistry, Developmental Studies or Biology.

G. CONSTRAINT DATA ENCODING

The C++ *Configuration* class reads the config file and storethe parsed objects into the system using the '`ParseFile`' method. It reads timetable object data by searching for object tags then calling appropriate objects. After parsing, the method clears previously parsed objects to allow taking in of new objects into the next iteration.

The config file stores objects data and their attributes. Each object is delimited between an opening tag and finishing tag. In the body of an object, each line contains only one key and value pair (attribute) separated by an '=' character. Each attribute is specified just once except for the #group object which can have multiple attributes.

VI. RESEARCH DESIGN

The research study was carried out at BUSE, faculty of science which was the domain of the problem under study. To overcome the timetable problem aforementioned in chapter one, a course timetable system using multi-phase GA was designed and implemented.

The system was put under a test by users (timetablers) in the faculty of science in order to determine their perception towards automated course scheduling in timetable fabrication.

The process involved sampling from the target population and then visiting each users within the sample space to carry out the investigation.

The procedure was as follows:

1. Randomly choose participants by using simple random sampling.
2. Concertising the participants (timetablers)about the scope of the research study and the relevant information about the University Course Scheduling desktop application program.
3. Installation and configuration of the application program on their computers.
4. Engaging the timetabler in the timetabling process, modifying the configuration file to suit the timetabler choices or requests then perform the timetabling using the automated system.
5. Verified the timetable drafts and re-computed to obtain timetable solution variations.
6. Administering of questionnaires.
7. Collecting the questionnaires and performing data analysis.

A. POPULATION AND SAMPLING

A population is generally a large collection of individuals of concern under a scientific query.

The university course timetable has three main stakeholders who represent the population and each as their own purpose these are:

1. The timetable committee (administration) sets the minimum standards that the timetable must conform to. For example, lectures start at 800hours and end at 1800hours.

2. The departments under the faculty of science, their concerns is the course schedule to be consonant with the development of the courses taught as well as making more specific demands for particular lecture rooms or laboratory.

3. The third group of stakeholders are the students, whose view of the timetable will be restricted to the part that affects them although given the number of students involved it is difficult to obtain specific criteria as to what is the best timetable for them.

B. RESEARCH INSTRUMENTS

Observation, questionnaires and the timetable system which we developed were used.During direct observations, the researcher was interested in observing the timetable system behavior and or events in real-time then consequently administered questionnaires for the purpose of gathering data from timetablers.

C. QUESTIONNAIRES

A questionnaire is a document containing questions to solicit information for appropriate analysis. Questionnaires present information in writing to respondents who in turn provide written responses in form of comments, ticks, rating or other response form. The questionnaires were administered to the population sample of forty people.

VII. DATA PRESENTATION, ANALYSIS AND INTERPRETATION

A. INTRODUCTION

Forty respondents participated from the five departments under the faculty of science with eight representatives for (Biological-Sciences, Chemical-Technology, Computer-Science, Developmental-Studies and Nursing Sciences).

B. ANALYSIS AND INTERPRETATION OF RESULTS

Both qualitative and quantitative data was used. The raw data was first preprocessed before use, thus, was checked for errors such as: consistency, completeness and duplication, SPSS (Statistical Package for Social Sciences) data analysis software was used for validation.

C. AUTOMATED TIMETABLING PROCESS ASSESMENT RESLUTS

This section acquired data from the population sample about timetabling using the multi-phase GA timetable system. Questionnaires contained closed-ended questions with a three point Likert response scale such that: (0-Not solved, 1-Partially solved, 2-Solved) in response to questions about: lecture room double bookings, equipment availability errors, lecturer double booking errors, student group double booking errors, day and time preference errors (misalignment).

12.5% of the respondents' perceived that the automated system partially solved the room errors and 87.5% perceived that the system solved the errors. 5% of the respondents' perceived that the automated system did not solve for equipment availability errors, 37.5% perceived that the system partially solved the errors and 57.5% perceived that the system solved the errors. 7.5% of the respondents' perceived that the automated system was partial in solving and 92.5% respondents perceived that the system solved the double booking errors. 2.5% of the respondents' perceived that the automated system was partial in solving double bookings and 97.5% respondents perceived that the system solved the student group double booking errors. 10.0% of the respondents' perceived that the automated system was partial in solving day preferences and 90.0% respondents perceived that the system solved the day preference errors. 92.5% of the respondents' perceived that the automated system was partial in solving time preferences and 7.5% respondents perceived that the system solved the time preference errors.

VIII. CONCLUSIONS, FUTURE WORK AND RECCOMENDATIONS

Respondents from department which had almost half of their courses as practical and heavily demanded use of academic equipment which is in shortage within the institution had a slight negative perception whilst the majority was positive, which showed that as the number of constraints increased and also as the constraints became tighter, the scheduling becomes less satisfactory. The future scope based on this study includes the need to amalgamate the departments timetabling into one unit for it to be solved as if it's a single department and per faculty level, so as to improve the system's ability to schedule university wide shared resources. Also try to cater for the fast growing dynamics of the institution enrolments, new building facilities for lectures, equipment and other changes which are possible in the future as quickly as possible.

REFERENCES

[1] Werra. D. (1985). "An Introduction to Timetabling", European Journal of Operation Research 19, pp151-162
[2] Wren. A. (1996), "Scheduling, Timetabling and Rostering – A Special Relationship?", In the Practice and Theory of Automated Timetabling, ed. E.K. Burke and P. Ross, pp46-75
[3] Burke. E, Kingston. J, Jackson. K and Weare. R (1997), "Automated University Timetabling: The State of the Art", The Computer Journal 40 (9) pp565-571
[4] Abdennadher. S and Marte. M (1999), "University timetabling using constraint handling rules", Journal of Applied Artificial Intelligence, Special Issue on Constraint Handling Rules
[5] Barták. R (2000), "Dynamic Constraint Models for Planning and Scheduling Problems", In New Trends in Constraints, LNAI1865, Springer, pp237-255
[6] Kocjan.W (2002), "Dynamic scheduling: State of the art report", Technical Report T2002:28
[7] Rudová. H and Murray. K (2003), "University course timetabling with soft constraints", In Edmund Burke and Patrick

DeCausmaecker, editors, Practice And Theory of Automated Timetabling, Selected Revised Papers, pp310–328

[8] Müller. T and Rudová. H (2004), “Minimal Perturbation Problem in Course Timetabling”, Proceedings of the 5th international conference on the Practice And Theory of Automated Timetabling, pp283-303

[9] Davis. L. E (1991), “Handbook of Genetic Algorithms”, New York: Van Nostrand Reinhold

[10] Corne D and Ross P (1995), “Peckish Initialisation Strategies for Evolutionary Timetabling”, In Burke E and Ross P (Eds): Lecture Notes in Computer Science 1153 Practice and Theory of Automated Timetabling First International Conference, Edinburgh, U.K., August/September 1995, Selected Papers. New York: Springer-Verlag Berlin Heidelberg, pp 227-240

[11] Anderson. A and Simpson. A. R (1996), “Genetic Algorithm Optimisation Software in FORTRAN Research Report No. R136”, Department of Civil and Environmental Engineering, The University of Adelaide

[12] Gen M and Cheng R (1997), “Genetic Algorithms and Engineering Design”, New York: John Wiley & Sons, Inc.

[13] Buckles. B.P and Petry. F.E (1992), “Genetic Algorithms”, Los Alamitos: The IEEE Computer Society Press

[14] Burke. E and Ross. P. E (1996), “Lecture Notes in Computer Science 1153 Practice and Theory of Automated Timetabling First International Conference, Edinburgh, U.K., August/September 1995, Selected Papers”, New York: Springer-Verlag Berlin Heidelberg

[15] Petrovic, S. and Burke E. K (2004), “University timetabling”, Handbook of scheduling: algorithms, models and performance analysis

[16] Thanh. N. D (2007), “Solving Timetabling Problem Using Genetic and Heuristic Algorithms”, Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp472-477

[17] Jong. De. K (1993), “Genetic Algorithms are NOT Function Optimisers.” In D. Whitley (ed.), Foundations of Genetic Algorithms, vol. 2, San Mateo, CA: Morgan Kaufmann, pp5-17

[18] Michalewicz. Z and Fogel. B.D (2000), “How to Solve It: Modern Heuristics”

[19] Gen. M and Zhang H. (2006), “Effective designing chromosome for optimizing advanced planning and scheduling”, In: Dagli, C. H., Buczak, A. L., Enke, D. L., Embrechts, M., and Ersoy, O. (ed.), Intelligent Engineering Systems through Artificial Neural Network, ASME Press, New York, pp61-66

[20] Norberciak, M. (2006). Universal Method for Timetable Construction based on EvolutionaryApproach.

[21] Schaerf, A. (1999, April). A Survey of Automated Timetabling. Artificial Intelligence Review,87-127.