# Developing Class Scheduler using Genetic Algorithm

Dave E. Marcial
College of Computer Studies,
Silliman University
Dumaguete City
63 – 035 - 4226033
demarcial@su.edu.ph

Albert Geroncio Rivera
College of Computer Studies,
Silliman University
Dumaguete City
63 – 035 - 4226033
alrivs@yahoo.com

## ABSTRACT

This paper discusses the applicability of Genetic Algorithm as a solution in developing an automated class scheduler system. Constraints like time conflicts, room conflicts, and professor information were organized, considered and solved in the study.

Three types of iterations were done to investigate how much looping the algorithm would need to get the lowest amount of conflicts possible. For the 500 iterations, there was an average of 10.43 conflicts. These conflicts were the total of conflicts from the faculty and the room. For the 1000 iterations, an average of 9.28 conflicts, and the 1500 iterations resulted in an average of 12.78 conflicts. These findings suggest that the higher the iteration rate, the more conflict is acquired. The same occurrence has happened on an iteration that is too low. Thus, an iteration of 1000 is embedded since it has the lowest conflict rate among the three tests.

The resulting schedules varied since the data was generated randomly. Also, there wasn't an occurrence when both of the faculty and room did not have a conflict at the same time. This just proves that for a college with many courses to generate, a 100% conflict-free schedule may not be possible.

**Keywords:** Genetic Algorithm, Class Scheduler

## 1. INTRODUCTION

A class schedule is one of the most important things in any university. The schedule would determine the tasks of the faculty, and it will be carried on for a whole semester. The process of scheduling is difficult and intricate, since many factors are to be considered. A schedule would usually take one to two weeks to finish by the secretary, since it is very complex. It is interesting to know if a scheduling process that is usually done by a person could really be solved by using an algorithm.

An algorithm is defined as a step-by-step problem solving procedure, used in order to solve a problem in a finite number of steps. However, many problems still exist wherein no algorithms can solve in a reasonable number of steps. Many interesting problems known are classified as Nondeterministic Polynomial-complete (NPC).

One of the features that make NP-complete problems so difficult is that the search space for potential solutions is sometimes so large that not every possible solution can be checked. A genetic algorithm narrows down this potentially large search space by using iteration and evolution techniques to jump over large patches of unlikely candidate solutions.

This presents the applicability of Genetic Algorithm (GA) as a solution in the development of a class scheduler. Accordingly, GA can be applied with great success on a wide range of problems, including class scheduling problems. Organizing a class schedule for a college or a department is a difficult endeavor. Time conflicts, room conflicts, and professor preferences all have to be organized and solved. In many cases an ideal solution may not exist, but in genetic algorithm, settling for suboptimal solution may be the best option.

### 1.1 Theoretical Background

Genetic Algorithms (GA) are intelligent heuristic methods that follow a process that simulates evolution in the computer. For a specific problem the solution is represented as a chromosome, which generally contains a sequence of 0s and 1s, indicating the values of a vector of decision variables. For this string of chromosomes, the objective value can be completed. A genetic method starts with a randomly generated population of solutions, and randomly combined portions of chromosomes together to form new solutions with an occasional notation. The new solutions are tested for feasibility. The best feasible ones from the previous and current generations are selected to survive to reproduce. After several combination iterations, the best solution is typically near an optimal solution to the decision making problem. Genetic algorithms have been applied to many large scale combinatorial mathematical programming problems such as large scale scheduling problems. Further, the "basic goal of genetic algorithms (also known as evolutionary algorithms) is to develop systems that demonstrate self-organization and adaptation on the sole basis

of exposure to the environment, similar to biological organisms. Attaining such a goal would provide special capabilities in pattern recognition, categorization, and association; that is, the system would be able to learn to adapt to changes." (Turban, Efraim and Aronson, Jay E. (1998).

Genetic algorithm is described as an iterative procedure maintaining a population of structures that are candidate solutions to specific domain challenges. Like in biological systems, a chromosome can make a copy of itself. The copy might be slightly different from its parent. During each generation, the structures in the current population are rated for their effectiveness as domain solutions, and on the basis of these evaluations, a new population of candidate solutions is formed using specific 'genetic operators' such as reproduction, crossover, and mutation. (Grefenstette, 1982)

Genetic algorithm systems start with a fixed size population of data structures which are used to perform some given tasks. After requiring the structure to execute the specified tasks some number of times, the structures are rated on their performance and a new generation of data structures is then created. The new generation is created by mating the higher performing structures to produce offspring. These offspring and their parents are then retained for the next generation while the poorer performing structure is discarded. (Patterson, 1990)

## 1.2 Definition of Terms

The following terms are described for a better understanding in this paper:

- *Chromosome* – this represents a string of data which contains the solutions. For example, a chromosome could represent the string 14324 or 42153. Its length will depend on the constraints.
- *Candidate Solution* – refers to a possible solutions
- *Fitness* – this represents the value which is assigned to a chromosome. The value is based on how far or close the chromosome is to the solution. A greater fitness value would represent a better solution.
- *Gene* – it is considered as a part of a chromosome, which also contains a part of the solution. For example, in the chromosome 14324, the genes are 1, 4, 3, 2 and 4.
- *Mutation* – this refers to the process of randomly changing a gene in a chromosome.
- *Nondeterministic Polynomial-complete(NPC) problems*- a problem is said to be NP if its solution comes from a finite set of possibilities
- *Optimization* – is deciding for the best alternative among a given set of choices
- *Population* – this represents a group of chromosomes, all having the same length
- *Search Space* – a space for all possible solutions.
- *Selection* – this refers to the process of selecting two new chromosomes for creating the next generation

# 2. THE RESEARCH COMPONENT

## 2.1 Complexity and Its Merits

Making a class schedule from scratch is possible by systematically going through every single combination of faculty, time, days, and room choices for every single class (course) and evaluating them to a large list of constraints. Then picking the choice with the smaller number of constraint violations. However for even a small number of classes, faculties, and rooms, the combinations to choose from rapidly ascend into the hundreds. And each of those hundreds of schedule needs to be checked for all violations. As additional classes are added, the search space increases exponentially. But by using genetic algorithm techniques these problems can be solved.

Genetic Algorithm (GA) is considered as the best-fitting algorithm to solve the class scheduler system since there are many existing schedules which use GA. A scheduler has many constraints to be considered and there are various possible candidate solutions. Therefore, it produces a large search space. GA is a helpful tool in solving problems with a large search space since it has the ability to narrow it down to be able to find the best solution for a problem. Additionally, the GA approach gives the ease of handling random kinds of constraints which is present in a scheduling problem. The chromosomes containing the constraints can be assigned with a fitness value, making it easy for the GA scheduler to produce a wider range of better alternative solutions.

The Class Scheduler System has the ability to automatically produce alternative class schedules based on the faculty, course, and room information. Everything will be handled by the application; all that the secretary would do is to supply all the necessary information. In that case, it lessen the burden of the college secretary in assigning schedules and checking if there are classes that conflict with each other.

## 2.2 Constraints

The variables in generating alternative class schedules include the field of expertise of the faculty, the time preference, unit load of the faculty, part or full-time faculty, class size, class section, availability of the room, room capability, and service units requested by other colleges.

Only the College of Computer Studies was used as the pilot for the development of the system in proving the applicability of genetic algorithm.

## 2.3 Search Techniques

Genetic Algorithm is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic Algorithm could be applied in a class scheduler system in order to create a system that would be able to translate human thinking into an effective program. Also, this algorithm is a way of mimicking the same process of a human intelligence.

The general algorithm that is being used is as follows:

**START**
Generate initial population.
Assign fitness function to all individuals.
**DO UNTIL** best solution is found
Select individuals from current generation
Create new offspring with mutation and/or breeding
Compute new fitness for all individuals
Kill all the unfit individuals to give space to new offspring
Check if best solution is found
**LOOP**
**END**

The basic idea of GA is to randomly generate an initial population which consists of possible solutions. Each of these solutions is represented by chromosomes, and is assigned with a fitness function. The fitness determines how far or close a chromosome is from a solution. The greater the fitness value, the better solution it contains. At this point a couple of solutions are chosen from among the relatively better ones in the population, and a form of gene-splicing occurs. In other words a random part of one solution is grabbed and placed within another. With every iteration, the algorithm creates new solutions from parts of existing ones. These new solutions are then placed into the population replacing some of the worst solutions that currently reside there, thereby slowly giving favor to better and better fitness values.

The generation of successors in a GA is determined by a set of operators that recombine and mutate selected members. These operators correspond to idealized version of genetic operations found in biological evolution. The two most common operators are crossover and mutation. Crossover is a genetic operator used to vary the programming of a chromosome from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. This operator produces two new offspring from two parent strings, by copying selected bits from each parent. On the other hand, mutation operator is used to maintain genetic diversity from one generation of a population of chromosomes to the next. It is analogous to biological mutation. Mutation operator produces small random changes to the bit string by choosing a single bit at random, then changing its value. (Mitchell, 1997)

## 2.4 GA's Pseudocode

Presented below are the pseudocodes of the GA applied in the generation of a class schedule:

START
Generate initial population.
Assign fitness function to all individuals.
DO UNTIL best solution is found
Select individuals from current generation
Create new offspring with crossover and mutation
Compute new fitness for all individuals
Kill all the unfit individuals to give space to new offsprings
Check if best solution is found
LOOP
END

**Pseudocode for the operation *testfitness*()**

BEGIN testfitness() operation
DO UNTIL all Schedules checked
IF Schedule Violates Constraints THEN
INCREMENT Fitness_value
END IF
LOOP
END testfitness() operation

The following are the specific constraints applied in the making of the class scheduling system, which is the basis in the genetic algorithm:

- A faculty can only teach one class at a time.
- A room can only hold one class at a time.
- A faculty can only teach classes that he/she is capable of.
- A class may only be held in a room that has the correct equipment.
- A faculty may prefer not to teach MWF, TTH, in the morning, afternoon or evening classes.
- A class may only be held in a room that has enough number of seats.

Pseudocode for the operation *useGenOperators*()

BEGIN crossover operation
FIND a random crossover point
FOR i = 0 to crossover point 1
child A gene[i] = parent A gene[i]
child B gene[i] = parent B gene[i]
END FOR
FOR i = crossover point 1 to crossover point 2
child A gene[i] = parent B gene[i]
child B gene[i] = parent A gene[i]
END FOR
FOR i = crossover point 2 to gene length
child A gene[i] = parent A gene[i]
child B gene[i] = parent B gene[i]
END FOR
RETURN children
END crossover operation
BEGIN mutation operation
FOR i=1 to gene length
IF random number is less than the mutation rate THEN
child A gene[i] = random value
child B gene[i] = random value
END FOR
END mutation operation

## 2.5 GA'S Operation

### *7.4.1 Crossover Operation*

Crossover operation generates two random numbers in the range of the total number of classes. Then every course between those two values will be swapped and all the rest stays

the same. The class with the low fitness value is the one that is likely to be picked.
EXAMPLE of a crossover operation
Parent A - a b c d e f
Parent B - g h i j k l
random number's = 2,5
Child C - a b | i j k | f
Child D - g h | c d e | l

### *7.4.2 Mutation Operation*

Crossover alone is not enough to create a good schedule. Mutation operation is simple but important because it introduces new population that may have been made impossible with the initial population. Mutation just takes class randomly and moves it to another randomly chosen slot. The simplest way to do mutation is when it is encoded in binary because its just a matter of flipping the bits.

EXAMPLE of a mutation operation: 1111001001 => 1101001001

## 3. DESIGN ARCHITECTURE

Figure 1 shows the design architecture of the proposed system. It starts with the adding of the needed data. The Course Information (which is composed of the course code, course title, course type, class size, units and course category), Room Information (which is composed of the room number, room capability, room capacity, name of the building) and Faculty Information (composed of the faculty's first name, last name, subjects to teach, status, type and time preference) will be used to create the schedules, which would make up the population. Each schedule in the population will be tested for its fitness value.
A schedule with the fitness value of zero would state that the schedule does not violate any of the given constraints, thus giving it a non-conflicting schedule. Otherwise, if the schedule would have a higher fitness value, crossover, and mutation operators (also known as genetic operators) will be applied to it. That makes up the reference population After passing through the crossover and mutation operators, the bad schedule in the reference population will be deleted, and the good schedules will be retained. After the execution of the genetic operators, the new schedule will be tested again for its fitness value. This process will be repeated until the system finds a schedule which is equal to zero. The final schedule will be displayed. The manual override feature allows the secretary or the college dean to modify the generated schedule.

## 4. ACTIVITY DIAGRAM

### 4. 1 creatSchedules() operation

First is to put all of the data needed for a class scheduler into look up tables. Then create an empty object array with the size of total number of courses offered for a particular semester. Get all courses from the look up tables and put them all in the object array according to their course category. The first ones to get in the object array are the course classified as the 'core' then the course classified as 'foundation'.

Then one by one, the **course** stored in the object array will be assigned to a **faculty** that will have an expertise that is equal to that course. Their priority number and unit overload will also be prioritized. If the most prioritized faculty will exceed with his/her the maximum overload then the course will be assigned to the next prioritized faculty with his expertise equal to that course and his unit overload will not reach the maximum overload. If there are more than one faculty that is prioritized for that subject it will randomly select from that faculty given that their units overload has not yet reached the maximum. If there are no more faculties to be assigned to that course, then it will be assigned as "TTBA" meaning "Teacher to be assigned". After that, it will repeat the process of assigning courses to a faculty until all courses are assigned. These data will also be placed into the object array.
It will now randomly pick a **room** and a **time slot** and will be placed into the object array.
Testing the fitness of a particular schedule is done by simply checking if a faculty will have a conflict in his/her time. The same checking is done for the room. If there will be a conflict, the fitness value of that particular schedule will be added with 1. After the testing, if the schedule has a fitness value of zero, the program will automatically stop, and the final schedule will be displayed. But, if there are no schedules that have a fitness value of equal to zero, the Genetic Operators will be applied. See figure 2 how the activity diagram was generated.

### 4.2 testfitness() Operation

Testfitness() operation's activity diagram is shown in figure 3. Testing the fitness of a particular schedule is done by simply checking if a faculty will have a conflict in his/her time. The same checking is done for the room. If there will be a conflict, the fitness value of that particular schedule will be added with 1. After the testing, if the schedule has a fitness value of zero, the program will automatically stop, and the final schedule will be displayed. But, if there are no schedules that have a fitness value of equal to zero, the Genetic Operators will be applied.

### 4.3. useGenOperators() Operation

This operation as shown in figure 4 will randomly select 4 schedules from the population and will be paired. The first and second will compete with each other. The schedule with the lesser fitness value will be placed in an object array called the 'best_fit'. There is also a 'worst_fit' object array. It is for the schedule that will lose the competition. This means that their fitness value is greater than the other. Same goes for the third and fourth picked schedule.

The two schedules that are placed in the best_fit array will be the parents for creating two new offspring or schedule.
The creation of new schedule will involve two genetic operators called the crossover and mutation.

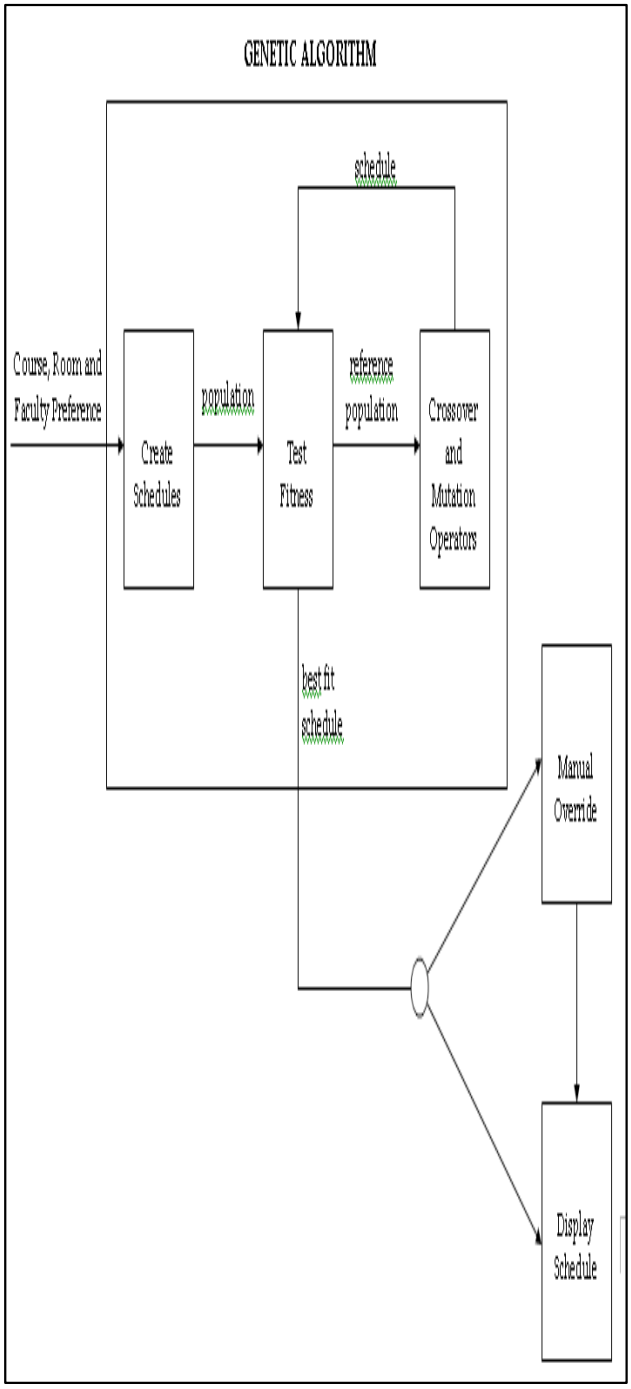


**Figure 1. Class Scheduler Design Architecture**

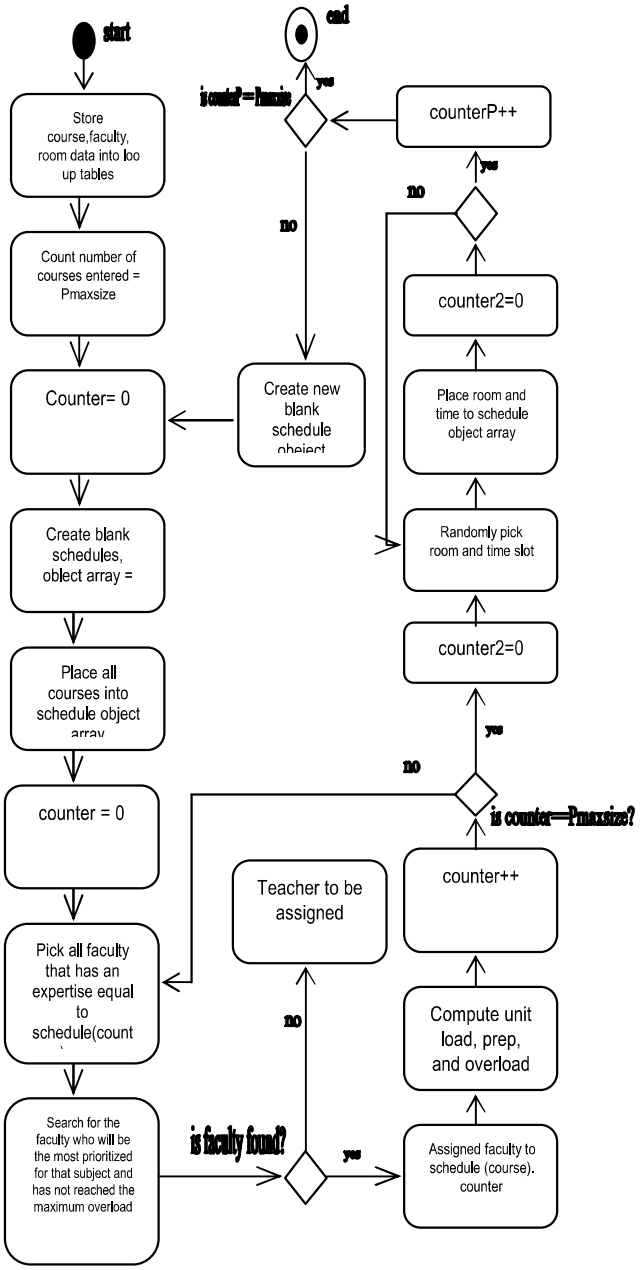


**Figure 2. createSchedules() Activity Diagram**

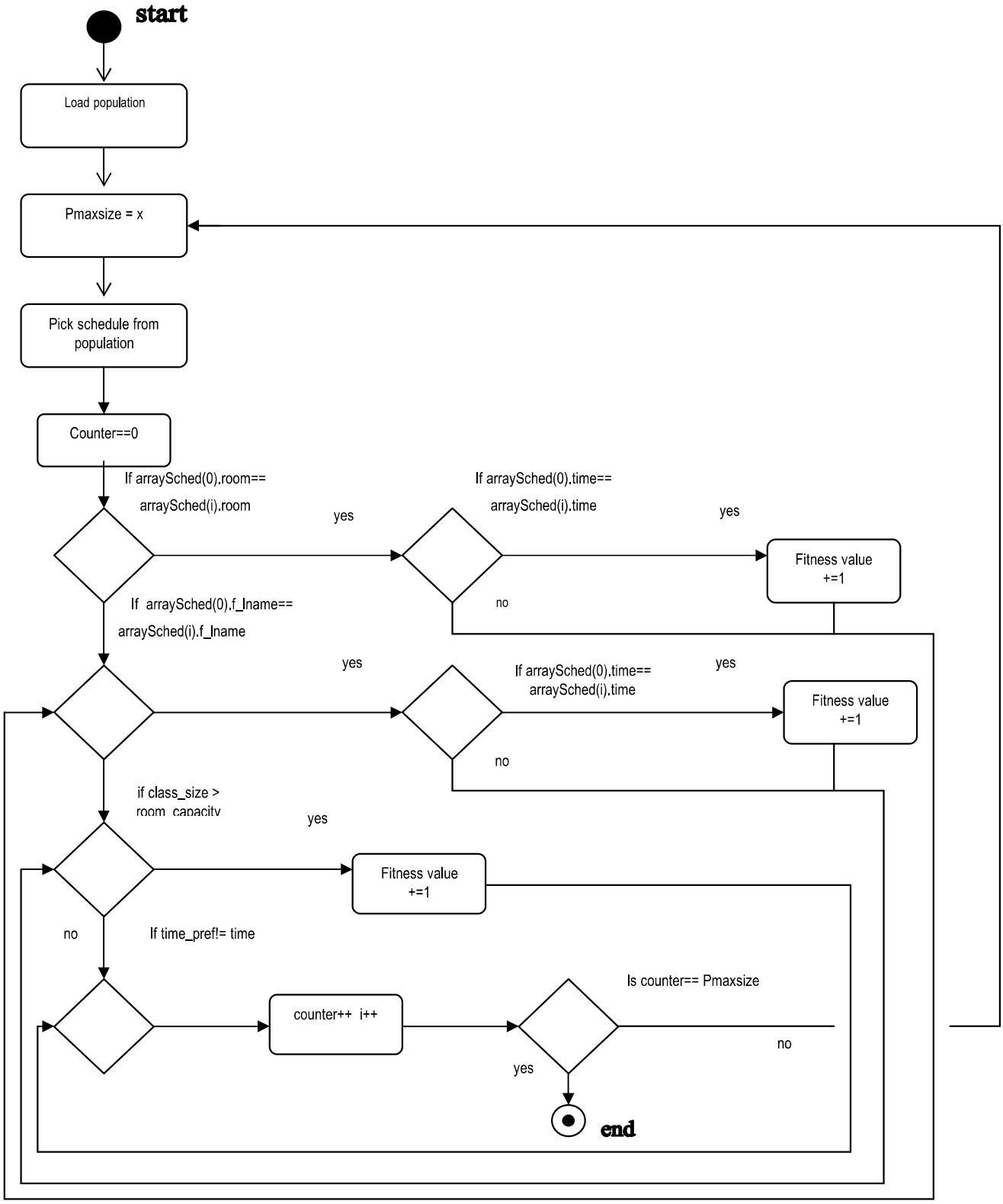


**Figure 3. testfitness() Operation Activity Diagram**

In crossover, it will randomly generate two random numbers in range of total number of courses offered for a particular semester. So if there are 60 courses, two numbers will be randomized in that range. Every **time slot** and **room** of the parents between those two randomly generated numbers will be swapped. No crossover operation will occur if for example the numbers generated is 4 and 5 or 10 and 11, this is because there are no data to be swapped in between them.

In mutation, it will again randomly generate two random numbers from 1 to 50. The first number will be for the first new schedule. The randomly generated number is for determining if that number is greater than to the mutation rate which is equal to 20. If it is less than the mutation rate, there will be a mutation to the first new schedule. The process of this is to randomly pick new time slot and room. If the number is greater than the mutation rate, no mutation will occur. Then again same goes for the second new schedule.

Now, the two new schedules created will have their fitness value tested and checked if their fitness value is equal to zero. If one of the two new schedules has fitness value of zero it will stop the execution. If not, the two new schedules will be placed back to the population table. But before doing that, the schedules that were placed in the worst fit array will be deleted, replacing those two new schedules created.

The operation here in **useGenOperators**() is that it will only stop if it finds a fitness value equal to zero or if the maximum iteration has been reached. If the maximum iteration has been reached, the schedule that has a fitness value closer to zero will be the final schedule.

## 5. THE CLASS SCHEDULER

The design of the proposed system intends to create graphical interface that users can easily understand. Security is of utmost concern, and the developers have taken steps to ensure that critical data is only accessible by authorized personnel. Authorized users are given freedom to manipulate data and interaction is encouraged. This allows for information exchange that has the user at the center of consideration. In many cases an ideal solution may not exist, but in genetic algorithm, settling for suboptimal solution may be the best option. See figure 5 for the main page of the proposed system.

## 6. TESTING & EVALUATION

Table 1 shows the results of the testing and evaluation of the generation of class schedule highlighting the number of iterations and the number of conflicts in class schedule created. In average, generation of class schedule will be finished 5.4 minutes with about 1 conflict in the faculty to a class time and a room to a class time.

The time it takes to generate the schedule is not stable since the data was randomly generated, and because of this some data sets may actually have had no possible solutions. Some schedule may have conflicts and others don't have. With these findings, it clearly states that with the fluctuating results of the time and conflicts occurred in the schedule, it does not give out a 100% schedule but closer to what perfection is. The final schedule will also vary on the inputted information. For a large amount of data, the time it took to get to the lowest fitness value was extensive.

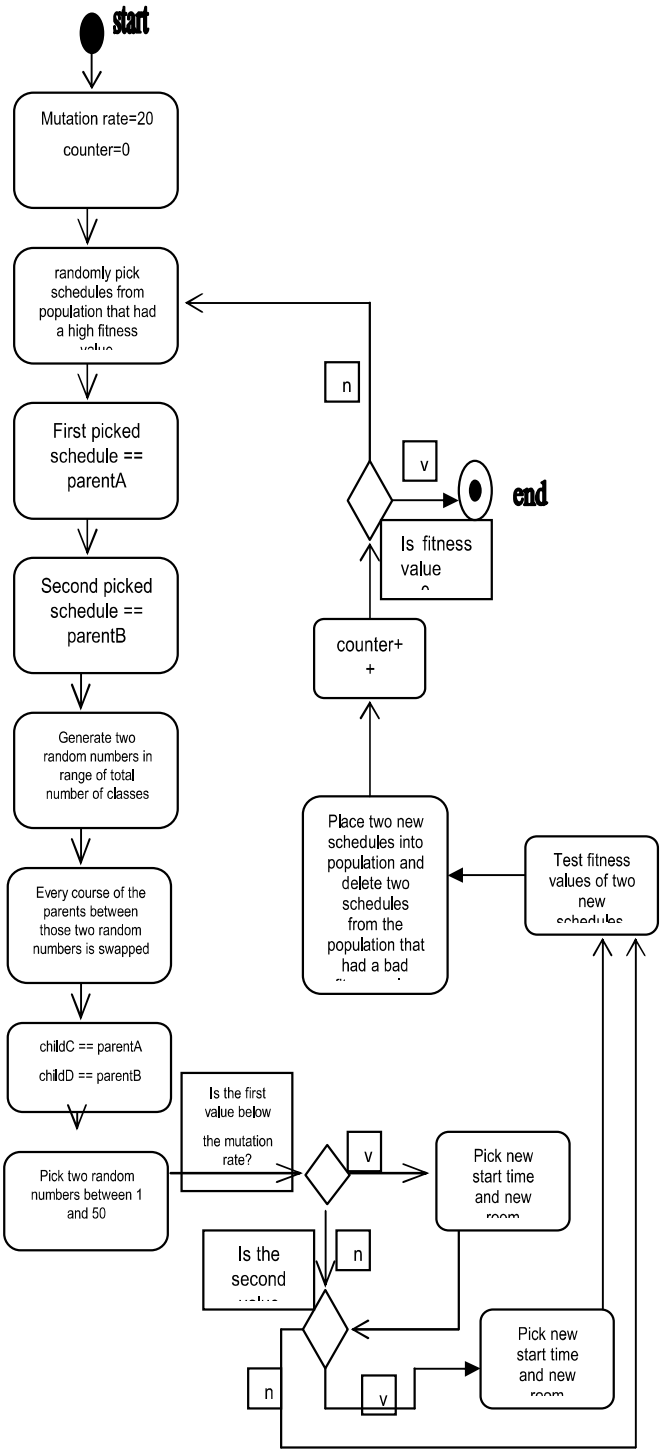


**Figure 4. useGenOperators() Operation Activity Diagram**

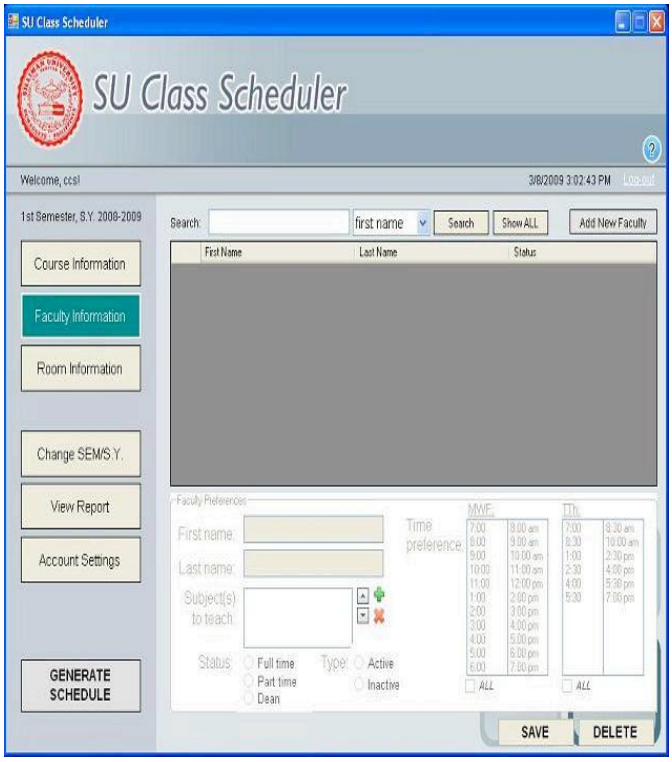


**Figure 5. SU Class Scheduler User Interface**

**Table 1. Testing Results**

| | TIME (in minutes) | # OF CONFLICTS | |
|---|---|---|---|
| | | Faculty-time | Room-Time |
| 1st trial | 6 | 1 | 0 |
| 2nd trial | 5 | 0 | 2 |
| 3rd trial | 5 | 0 | 1 |
| 4th trial | 6 | 1 | 0 |
| 5th trial | 6 | 0 | 1 |
| 6th trial | 5 | 0 | 2 |
| 7th trial | 5 | 1 | 0 |
| 8th trial | 5 | 2 | 1 |
| 9th trial | 6 | 0 | 2 |
| 10th trial | 5 | 1 | 0 |
| AVERAGE | 5.4 | 0.6 | 0.9 |

The testing contains the following constraints in the generation of class schedules.

- Total number of courses: 65
- Total Number of Rooms: 12
- Total Number of Faculty 14
- A faculty can only teach one class at a time.
- A room can only hold one class at a time.
- A faculty can only teach classes that he/she is capable of.
- A class may only be held in a room that has the correct equipment.
- A faculty may prefer not to teach MWF, TTH, in the morning, afternoon or evening classes.
- A class may only be held in a room that has enough number of seats.

There were three more batches of tests that were done – one with 500, 1000 and 1500 iterations. These iterations were tested for 50 trials each. The data included 61 courses, 14 faculty and 12 rooms. Three types of iterations were done to investigate how much looping the algorithm would need to get the lowest amount of conflicts possible. For the 500 iterations, there was an average of 10.43 conflicts. These conflicts were the total of conflicts from the faculty and the room. For the 1000 iterations, an average of 9.28 conflicts, and the 1500 iterations resulted in an average of 12.78 conflicts. These findings suggest that the higher the iteration rate, the more conflict is acquired. The same occurrence has happened on an iteration that is too low. That is why the researchers have decided to settle for an iteration of 1000, since it has the lowest conflict rate among the three tests.

## 7. SUMMARY OF FINDINGS, CONCLUSION AND RECOMMENDATION

### 7.1 Summary of Findings

The purpose of this system is to aid the secretary in creating class schedules, and it does not only create schedules but provides an environment that will ease the task of creating a schedule. With the use of this system, the span of time when making schedules will be reduced. This system makes sure that the final schedule will have a minimal number of conflicts as possible. All of the information that would be used in making the schedule will be based on what the secretary inputted.

More importantly, the system design intends to create graphical interface that users can easily understand. Security is of utmost concern, and the developers have taken steps to ensure that critical data is only accessible by authorized personnel. Authorized users are given freedom to manipulate data. This allows for information exchange that has the user at the center of consideration. In many cases an ideal solution may not exist, but in genetic algorithm, settling for suboptimal solution may be the best option.

## 7.2 Conclusion

The study concluded that that the higher the iteration rate, the more conflict is acquired. The same occurrence has happened on an iteration that is too low. Thus, embedded in the application program particularly in the algorithm is an iteration of 1000, since it has the lowest conflict rate among the three tests.

Further, the study revealed that genetic algorithm does not give a 100% ideal schedule, but it gives an outcome which is closer to what an ideal schedule would be. Specifically, it was also discovered that the proposed system can generate a class schedule in about 6 minutes with one conflict scenario. Further, although perfect schedules were almost impossible to come by, many schedules were found that would at least satisfy all the major constraints of the problem.

## 7.3 Recommendation

The Class Scheduler is developed for the purpose of automatically creating class schedules for the existing Colleges in Silliman University. The developers recommend that the system be implemented in the University because it will lessen the hassle of the college secretaries to manually encode and compare class schedules. However, due to lack of time to learn further knowledge about genetic algorithms, the Class Scheduler System, though a 100 percent running program, has still limitations and shortcomings.

The system does not guarantee a perfect, non-conflicting schedule. The development team may have not come up with the greatest solution to resolve the problem but this shouldn't curb the aspiring developers of this system to unravel better ways and come up with a much improved genetic algorithm and superior program ideas.

It is also recommended for further study the consideration of using other methodologies and specific algorithm in genetic algorithm. Similarly, to conduct a study on another heuristic approach in generating class schedules that might include other constraints not mentioned in the study.

Moreover, the researchers further recommend to future researchers to conduct an evaluative study on the accuracy of the final schedules generated by the proposed system for improvement and enhancement.

## 8. ACKNOWLEDGEMENT

The output of the developed system is made possible through the ultimate effort of the following developers who graduated their degree in BSIT namely: Justin Fred M. Opeña, Honeylet B. Laput and Elia - Grace V. Limbaga. Special thanks also to the faculty and staff in the College of Computer Studies, Silliman University who helped in the quality assurance component during the development of the class scheduler system.

## 9. REFERENCES

[1] Grefenstette, J. (1982). "Optimization of Control Parameters for Genetic Algorithms." *IEEE Transactions on Systems Management and Cybernetics.*

[2] Mitchell, 1997[ Mitchell, T. M., *Machine Learning.* WCB/McGraw Hill, 1997

[3] Patterson, 1990 *Introduction to Artificial Intelligence and Expert Systems*, Dan W. Patterson, Prentice-Hall International Editions, 1990.

[4] Turban, Efraim and Aronson, Jay E. (1998). *Decision Support Systems and Intelligent Systems. Fifth Edition.* Prentice-Hall, Inc.