# Genetic Algorithms in Activity Scheduling

Ryan Austin Fernandez
CS-ST Student
2401 Taft Avenue,
Manila, Philippines 1004
(+63)917-621-6469
ryan_fernandez@dlsu.edu.ph

Clarisse Felicia Poblete
CS-ST Student
2401 Taft Avenue,
Manila, Philippines 1004
(+63)915-624-5417
clarisse_poblete@dlsu.edu.ph

## ABSTRACT

Activity scheduling is a difficult problem, especially with the numerous complexities such as target audience for events, venue, multiple possible dates and times, date and time restrictions, and plenty other restrictions. Genetic algorithms, a class of approximation algorithm that is very useful for optimization, is proposed as a solution to this problem in this paper. This paper aims to details the development of a system to schedule activities for student organizations, which can help greatly in avoiding conflicts in the future. This was done by implementing a conflict based fitness function and defining assigned times and dates as units in the chromosome. The algorithm performed well, using actual data from the La Salle Computer Society's GOSM from the previous term, only producing minimal acceptable conflicts. It was then deemed feasible to use GA for this type of problem. It is recommended for future study on this topic to add more capabilities and constraints to the problem to better represent real life scheduling problems in this context.

## 1. INTRODUCTION

Scheduling is an important problem in the real world. This is evident in professional organizations in universities that hold numerous activities for their members. Unfortunately, this is not a simple task due to the numerous possibilities for schedules and the complex nature of the problem, since activities have variable times and may have differing target groups and venues.

Mitchell, T. (1997) and Russell, S & Norvig, P. (2010) detailed an evolutionary algorithm known as genetic algorithms or GA which is a unique way of finding solutions to optimization and constraint satisfaction problems, among many other problems. This would imply that GA could be used in scheduling (Mitchell, T., 1997; Russel, S. & Norvig, P.,2010).

Several studies have already been made on this problem. Corne, D., Fang, H.L. & Mellish, C. (1993) used genetic algorithms to schedule a single set of exams for a single set of students. Their study produced GA's that performed almost as well as human schedulers (Corne, D., Fang, H.L. & Mellish, C., 1993).

Wall, M.B. (1996) implemented GA for resource-constrained scheduling by encoding the results as an array of delay times and execution modes, since his system allowed for multiple execution modes. His algorithm did not perform well on situations where the input was tightly constrained but was otherwise successful (Wall, M.B., 1996).

Sedegheih, A. (2006) used simulated annealing and genetic algorithms. For the GA, he used rank based selection in selecting parents for crossover. For mutation, two jobs would swap schedules in the lineup. He was successful in generating good schedules (Segegheih, A., 2006).

Jankovic, M. (2008) used GA to create class schedules. The chromosome was designed as a hashmap, mapping each timeslot to a class, with which crossovers were performed by swapping the contents of this hashmap at a certain crossover point. Mutation was done by moving one class to another randomly chosen slot (Jankovic, M., 2008).

Omara, F.A. & Arafa, M.M. (2010) implemented a GA that used a two ply fitness function, the first aiming to minimize the execution time, the second concerned load balancing. It was found that their algorithm outperformed traditional GA due to the two ply fitness function (Omara, F.A. & Arafa, M.M., 2010).

After reviewing all these studies, however, none have dealt with a scheduling problem spanning across multiple dates and allowing jobs to coincide if they are not aimed at the same target group, which is a concern for the activity scheduling in the student organizations. This paper aims to see if developing a GA for this situation is feasible.

### 1.1. Objectives

The general objective of this paper is to develop and implement a system that find the optimal schedule using genetic algorithms.

The specific objective are:

- To build a model for the process and data requirements of the system
- To implement a GA model for the problem
- To implement a system that would allow users to use the implementation of the algorithm in a real world situation.

### 1.2. Scope and Limitations

For the data requirements, this project will handle multiple scheduling sessions, which comprise of sets of activities to be scheduled. For each session, the system should be able to handle a single date range and restrictions on specific days of the week, specific days, and any number of time ranges. Activities can have multiple target groups, but only one venue, one time range, but possible multiple allowed days of the week and allowed dates.

The GA model will only cover the algorithms described by Mitchell, T. (1997) and Russell, S & Norvig, P. (2010). The GA modelling for the fitness, chromosomes, and algorithm parameters will be included in the scope.

The system will be a web application developed on the J2EE platform to be run on an Apache Tomcat 7.0 Server. This application will handle users, adding sessions, adding activities, and generating schedules.

## 1.3. Significance of the Study

The constraints presented in this paper have been rarely used in most studies regarding GA's in scheduling. This paper may present a new kind of scheduling problem and insights on how to solve the problem using GA.

The application developed through this project will be of good use to the various student organizations, not just in De La Salle University but also in other universities. This application will be applicable to almost any scheduling context in terms of activity or event scheduling.

# 2. FORMAL DEFINITION OF THE ALGORITHM

A genetic algorithm (GA) is an algorithm that uses the concept of evolutionary genetics to arrive at its answer. Each potential answer comes from the "sexual reproduction" of previous answers (Russell, S. & Norvig, P., 2010). Genetic algorithms have risen in popularity due to the fact that evolution is known to be a robust and successful method for adaptation within biological systems; GA's can search spaces of hypotheses containing a lot of complex parts that interact with each other, where the impact of each individual part may be difficult to model; and GA's can easily be programmed in parallel and can take advantage of the decreasing costs of powerful computer hardware (Mitchell, T., 1997).

## 2.1. Fitness

GA's attempt to find the best solution using a metric called *fitness*. The higher the fitness of a solution, the more it can help contribute to finding the best solution. For example, in using GA's to solve the n-queens problem, the fitness could be the reciprocal of the number of queens attacking each other. That means a set-up where more queens are attacking each other would have a lower fitness since the denominator is higher, resulting in a lower quotient.

## 2.2. Chromosomes

Another aspect of GA's is how solutions are encoded. Solutions have to be encoded in a series of bits or separable units of information known as a *chromosome*. The n-queens problem would have each queen's position in each file string, so for a 4x4 n-queens problem, the solution's chromosome would look like "1 4 3 2".

The way chromosomes are defined is important, since now we come to the important operations of the GA process: crossover and mutation.

### *2.2.1. Crossover*

Crossover takes two chromosomes and chooses a random crossover point or multiple crossover points, depending on the implementation. Say we have two solutions to the n-queens problem "1 4 3 2" and "4 1 2 3". If the third position is chosen as the starting crossover point, this would result in two children: "1 4 2 3" and "4 1 3 2". The first and second positions for each chromosome were retained while the third and fourth were swapped. In a two point crossover, two crossover points are chosen. For example, the second and fourth positions are chosen, then the children would instead be "1 1 2 2" and "4 4 3 3". In a point crossover, only one bit is swapped, so if the first bit is selected, the offspring would instead be "4 4 3 2" and "1 1 2 3" (Mitchell, M. 1997).

### *2.2.2. Mutation*

Mutation is to decrease the chances of entering a local minima by perturbing the current status of the solution by introducing randomness. An implementation must just define an operation that changes the composition of a solution. For example, in our n-queens problem, a mutation could be swapping two random queens, going from "1 4 3 2" to "1 2 3 4" for example.

## 2.3. The Algorithm Proper

The algorithm proper has a few versions. It always starts with an initial population *n*. Each member of this population is a randomized chromosome.

According to Mitchell, T. (1997), there should be an elitism rate, which transfers a segment of the population to the next generation unconditionally. The selection of which members to bring to the next generation is through roulette wheel selection.

Roulette wheel selection is a weighted probability selection. Instead of a uniform probability distribution for each member of the population, the probability of chromosome *c* being selected is equal to

$$\frac{c_{fitness}}{\sum_{x \in population} x_{fitness}}$$

so for a population with chromosomes having fitness values 1, 10, 33, and 6, the probabilities for each chromosome being selected are 2%, 20%,66%,and 12% respectively. The selection is done with the entire population each time, which means there is a chance that the same chromosome be moves to the next population multiple times. There are other means of selection other than roulette wheel such as the tournament selection, where for each chromosome selected, two are selected with uniform probability and with a certain probability *p*, the one with the higher fitness is selected, but with a probability *1-p*, the lower fitness chromosome is selected. Rank selection is also an option, where the population is sorted according to fitness and the selection is weighted based on the rank instead of the fitness.

After this, the remaining part of the next population is populated with offspring. Pairs of chromosomes from the current population are chosen using the roulette wheel and the children resulting in their crossover are added to the result.

Afterwards, mutation happens. According to Russell, S. & Norvig, P. (2010), each chromosome in the new population can mutate with a probability *m*. However, Mitchell, M. (1997) says that *m* * *n* chromosomes in the new population are unconditionally mutated.

After mutation, the new population becomes the current population and the best chromosome is updated based on the new population; the process iterates until either a certain fixed number of iterations have passed or the best chromosome's fitness reaches a certain threshold.

# 3. ANALYSIS AND COMPARISON OF THE ALGORITHM TO OTHER ALGORITHMS

This section dives deeper into the properties of genetic algorithms and is divided into two parts: analysis of the characteristics and issues of genetic algorithms and comparisons with other algorithms used for approximation and optimization.

## 3.1. Properties and Issues of Genetic Algorithms

GA's are commonly used for optimization problems. Solutions for problems such as the traveling salesman problem, which are NP-hard can be approximated using GA's.

GA's tend to prefer chromosomes that follow a schemata, which is a pattern exhibited by a subset of the population. Take the n-queens for example. Solutions that follow the schemata "1 3 * *" may have a higher fitness than those what have "1 2 * *". Crossover aims to combine chromosomes with desirable schemata in hopes that a chromosome can follow two or more desirable schematas.

Generally, GA's run in O(*mn*) time, *m* being the number of iterations set and *n* being the population size. This may improve with parallelization.

An issue with GA is crowding, where a chromosome with extremely high fitness reproduces too much, quickly taking over the entire population. This reduces the diversity of the population, bringing the current hypothesis to a possible local minima.

### 3.2. Comparisons With Other Algorithms and Techniques

When compared to neural networks following the backpropagation algorithm, an ANN would go from one hypothesis to another slowly, whereas GA would move more abruptly to a new hypothesis. This reduces the chance of falling into a local minima.

## 4. APPLICATIONS

This section details the implementation details of the activity scheduler and is divided into two parts: the system overview and the genetic algorithm design specifications.

### 4.1. System Overview

This subsection details the system flow and data requirements for the system.

#### *4.1.1. System Flow*

To use the activity scheduler, a user would first have to have an account. This account associates the user to all their venues, target groups, and sessions. Users cannot access venues, target groups, and sessions of other users.

After registering and logging in, a user can now add a session. Each session associates a list of activities to be scheduled. The user then adds all the activities to be scheduled. At this point, the user can adjust the session settings, to be discussed in further detail in subsection 4.1.2. After finalizing the parameters, the user can then generate the schedule, after which the system will assign the date and time to each activity in the session according to the optimal schedule found by the algorithm.

#### *4.1.2. Data Requirements*

Each user has an id, username, password, first name, last name, and email. Passwords should have at least one of the following: lowercase character, uppercase character, number, special character.

Each user has target groups, venues, and sessions. Target groups have an id and a name. Venues have an id and a name as well.

Sessions denote a period where activities are to be scheduled. They have an id, an optional name, blacked out days which are days in the week where activities are prohibited, start date, end date, list of dates that are restricted, and list of times that are restricted.

Sessions also have a list of activities. Activities have an id, venue, name, length in minutes, days of the week it could be scheduled, start time range, end time range, list of target groups, list of other possible dates, and after running the algorithm, an assigned date and time.

Figure 1 shows the relational model that represents all these data requirements.

### 4.2. Genetic Algorithm Design Specifications

This section details the design considerations in implementing the genetic algorithm for this problem. This section is divided into three subsections: the fitness function, the chromosome definition, and the algorithm parameters.

#### *4.2.1. Fitness Function*

The main evaluation metric for a schedule of multiple activities is how many conflicts there are. Corne, D., et al. (1993) proposed a fitness function of 1 / (punishment + 1). Punishment is the amount of conflicts for that particular schedule. Since this study differs from theirs due to the target group parameter, their punishment computation was slightly modified to suit this paper's needs as follows:

- For each date and time conflict regardless of target group, add 2 punishment points.
- For each date and time conflict with intersecting target groups, add 7 punishment points.
- For each date and time conflict with same venue, add 10 punishment points.
- For each pair of activities where one activity takes place after the other on the same day, add 3 punishment points.

Punishment points can stack. If there is a date time conflict between two activities with the same target groups and venue, the punishment for that pair is 2 + 7 + 10 = 19.

The total punishment points of the entire schedule therefore is the total punishment of all unique pairs of activities in the schedule.

#### *4.2.2. Chromosome Design*

The chromosome is a schedule. To be more specific, a chromosome is a list of activities, each with an assigned date and time. The single unit in this chromosome is the activities.

##### *4.2.2.1. Crossover*

Crossover is done by selecting an activity *a* as the crossover point. For activities 1 to *a*, the assigned times are retained. For activities *a + 1* to *n*, *n* being the number of activities in the current session, the assigned dates and times are taken from the other parent.

For example, the problem is to schedule four Activities. There are two chromosomes as shown in Table 1.

**Table 1. – Parent Chromosomes**

| | **Chromosome 1** | **Chromosome 2** |
|---|---|---|
| **Act 1** | 08/12/16 14:00:00 | 08/19/16 13:00:00 |
| **Act 2** | 09/01/16 12:30:00 | 09/08/16 12:45:00 |
| **Act 3** | 08/15/16 08:00:00 | 09/01/16 10:45:00 |
| **Act 4** | 09/03/16 11:30:00 | 09/05/16 12:00:00 |

If these two chromosomes are to be crossed over, through uniform probability distribution, a crossover point is to be selected. For example, if the crossover point is at Act 4, the resultant children will be as shown in Table 2.

**Table 2. – Children Chromosomes**

| | Chromosome 1 | Chromosome 2 |
|---|---|---|
| **Act 1** | 08/12/16 14:00:00 | 08/19/16 13:00:00 |
| **Act 2** | 09/01/16 12:30:00 | 09/08/16 12:45:00 |
| **Act 3** | 08/15/16 08:00:00 | 09/01/16 10:45:00 |
| **Act 4** | *09/05/16 12:00:00* | *09/03/16 11:30:00* |

The italicized times are the times that came from the other parent.

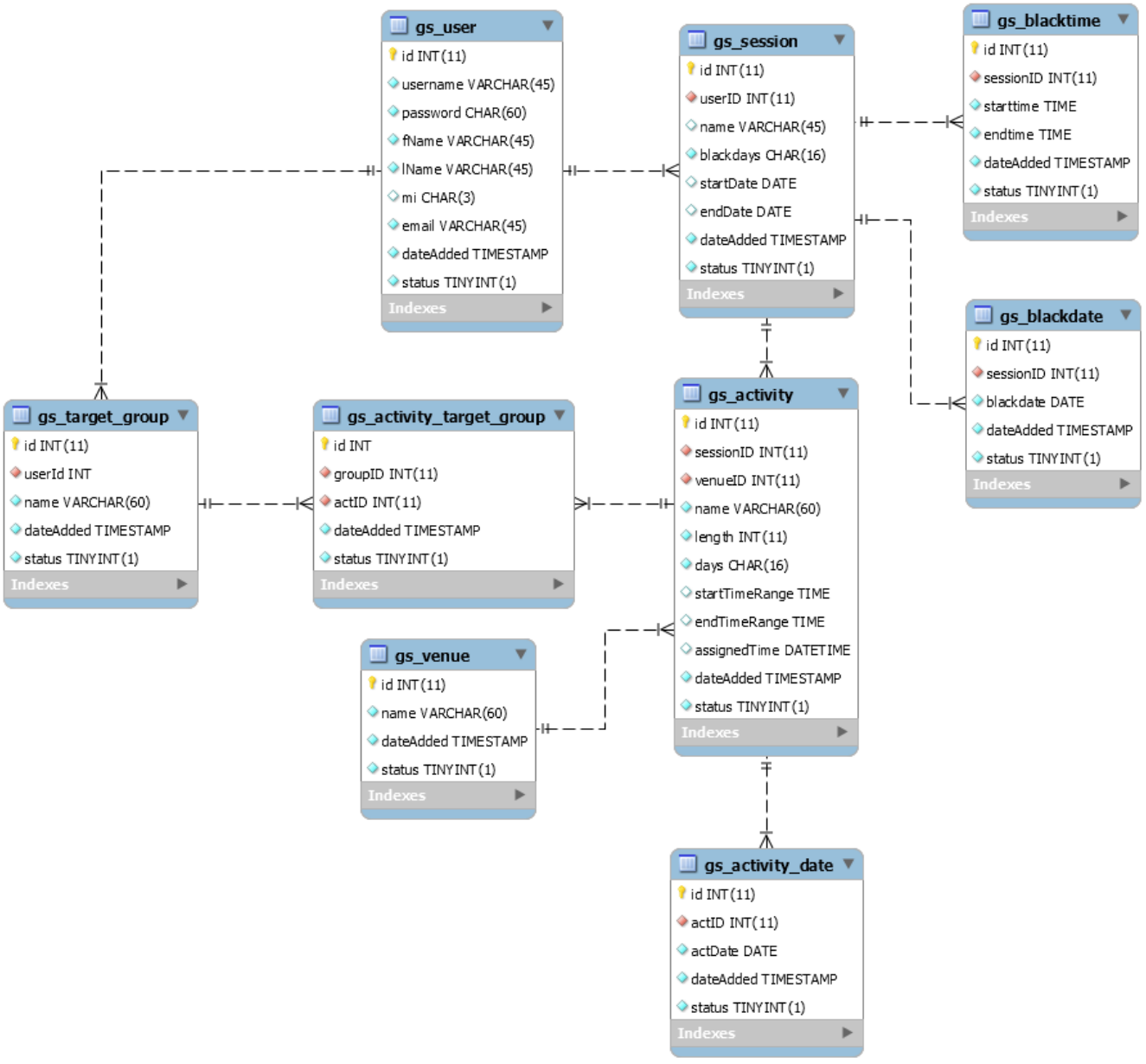


**Figure 1. – Relational Model of the Activity Scheduler System.**

#### *4.2.2.2. Mutation*

Mutation is a more complicated matter and it is related to the randomization of the initial population.

To mutate a chromosome, each activity, with a probability 0.7, has their assigned time randomized. This is done by first getting all possible dates. To do this, the entire date range is extracted from the session. An empty list of dates is initialized. Next, all additional dates are included if they are not in the restricted dates or restricted days of the session. Next, for each day in the date range of the session, if it falls on the allowed days of the week of the activity and is not yet in the list, it is added.

After the list of dates is collected, a list of times are also collected. This is done by getting the total time range of the activity, all the blacked out times from the session are then removed from this time range, possibly resulting in multiple time ranges.

For example, if an activity's time range is 0800H to 2000H and the restricted times are 1000H to 1100H and 1300H to 1400H, the resulting time ranges are 0800H to 1000H, 1100H to 1300H, and 1400H to 2000H.

After the possible date and time ranges have been determined, a date time pair is randomly selected. With uniform probability distribution, a date is selected. Afterwards, based on the length of the time period, one of the possible time ranges is selected. With the previous example, the time ranges have lengths of 120, 120, and 360 minutes respectively. They then would have a probability of 20%, 20%, and 60% respectively. After a time range is selected, the time range is split into fifteen (15) minute chunks. One of these chunks is randomly selected.

Finally, the selected date and selected time are combined and assigned to the activity. This is done for each activity in the chromosome that hits the 0.7 mutation probability.

### *4.2.3. Algorithm Parameters*

The algorithm was based on Mitchell, M. (1997). A population size of 50 was selected. Randomizing the population just involved randomizing the activity schedules as described in section 4.2.2.2 in each new chromosome for each activity. The fitness threshold was set at 0.14, which would allow two instances of two activities taking place one after another, which was deemed acceptable. An elitism of 20% was chosen, which allowed 10 members of the previous population to carry over. A mutation rate of 40% was selected, which means 20 chromosomes were mutated per iteration. This was chosen because it was deemed necessary that in a scheduling context, a large amount of perturbations in the search space was necessary to escape plateaus. Finally, 1,000 iterations were run at most, to allow ample time to find the ideal solution.

### *4.3. Results and Analysis*

The performance of the algorithm was satisfactory. Most of the time, it was able to produce a schedule with a minimal amount of conflict.

However, it did not produce the optimal solution all the time. This is due to the fitness threshold mentioned in section 4.2.3. Since it allowed one specific type of conflict, if a situation would lead to two possible schedules, one with no conflict, one with two instances of that specific type of conflict, the algorithm would sometimes settle for the worse answer. This is acceptable, however, in the long run since in real world application, that kind of conflict may not affect attendance to activities too much.

To stress test the algorithm, the GOSM spreadsheet (Goals, Objectives, Strategies, Measures) for the La Salle Computer Society during the third term of the academic year 2015-2016 was input into the system. This dataset contained 22 activities.

The results were satisfactory with the only activities taking place on the same day not having a time conflict, while another pair of activities were sat on the same day but had differing target groups.

## 5. CONCLUSION

The genetic algorithm performed well after multiple tests. Only acceptable amounts of conflicts resulted from the algorithm.

Over the course of this project, the proponents were successful in implementing a data model to meet the process requirements of the problem. A genetic algorithm was successfully implemented to schedule the data represented by this data model. Finally, a system was implemented that allows users to use the algorithm to schedule activities in the real world. Therefore, it is feasible to use GA for this type of activity scheduling algorithm.

As for recommendations for future research, perhaps adding more capabilities and constraints to the data model is desirable to represent the real world requirements of this kind of problem e.g. multiple time slots per activity and multiple venues per activity.

In conclusion, this paper can very well contribute to the organizations that may need help in effectively scheduling their activities. This could improve student attendance in these activities which could overall improve the quality of student and organization life in universities.

## REFERENCES


[1] Corne, D., Fang, H.L. & Mellish, C. (1993). Solving the Modular Exam Scheduling Problem with Genetic Algorithms. Technical Report 622, Department of Artificial Intelligence, University of Edinburgh.

[2] Czarn, A, et al. (2004). Statistical Exploratory Analysis of Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 8(4). 405-421.

[3] Omara, F.A. & Arafa, M.M. (2010). Genetic algorithms for task scheduling problem, *Journal of Parallel and Distributed Computing*, 70(1), 13-22, ISSN 0743-7315.

[4] Jankovic, M. (2008). Making a Class Schedule Using a Genetic Algorithm. Retrieved July 4, 2016, from Code Project: http://www.codeproject.com/Articles/23111/Making -a-Class- Schedule-Using-a-Genetic-Algorithm

[5] Mitchell, T. (1997). *Machine learning*. McGraw-Hill.

[6] Russell, S. & Norvig, P. (2010). Artificial Intelligence: A Modern Approach, 3rd Ed. New Jersey: Prentice-Hall.

[7] Sadegheih, A. (2006). Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance, *Applied Mathematical Modelling*, 30(2), 147-154, ISSN 0307-904X.

[8] Wall, M.B. (1996). A Genetic Algorithm for Resource-Constrained Scheduling. Massachusetts Institute of Technology.