# A Genetic Algorithm for Scheduling

Jack Ryder
jryder@kean.edu
Mathematics and Computer Science
Kean University
1000 Morris Ave.
Union, New Jersey, 07083

**Abstract**: Genetic algorithms use concepts from evolutionary biology as a technique for solving problems. They have successfully been used on a large number of scientific and engineering problems such as: optimization, machine learning, adaptive systems behavior and complexity. The genetic algorithm evolves over time a population of potential solutions by randomly selecting, based on fitness, solutions to reproduce a population of next generation solutions using crossover and mutation of solution parameters. This paper presents the genetic algorithm, necessary terminology, operations for selection, crossover, mutation, fitness, and gives an example in the area of scheduling.

Keywords: Genetic Algorithms, Scheduling, Engineering Computer Education

## 1. Introduction:

The genetic algorithm is a problem solving technique that uses the metaphor of evolutionary biology in its approach. A population of potential solutions are generated and tested for their ability to solve the problem. If one of the solutions solves the problem the process is complete. If not, a new population of solutions is generated using parts of the better performing solutions from the previous population, and the process of evolving a solution continues.

John Holland [1] developed genetic algorithms as an abstraction of biological evolution and provided the mathematical framework for adaptation of genetic algorithms. Many problems involve searching through a large number of possibilities for a solution. Other computational problems require programs to be adaptive. Still others require new or novel ideas in their solutions. Genetic algorithms are well suited to these types of problems. They have successfully been used for problem solving in such areas as machine learning, robotics, adaptive systems and optimization [2] [3].

## 2. Genetic Algorithm and Terminology:

Species (solutions) evolve over time through variation of genetic material by mutation and recombination. Individuals with better genetic material, based on environmental considerations, are more likely to survive into the next generation. The notions of genetic material, exchange of genetic material and survival of the fittest are fundamental to evolution and the development of genetic algorithms.

From biology each cell in an individual organism contains identical chromosomes. The chromosomes consist of genes (strings of DNA) where each gene can be thought of as encoding one trait of an organism. Each species genetic material is held by the collection of individuals in that species. Genetic algorithms use a chromosome as a potential solution to a problem. Each gene in the chromosome can be thought of as a trait or parameter that will help in solving the problem. The problem space is usually multidimensional, so each gene represents one dimension of the solution. The chromosome can be thought of as an array of traits. Collectively, all solutions (chromosomes) are called the population. The most common way to encode traits as genes in chromosomes is to use a binary string where each bit represents some trait of the solution.

In sexual reproduction each parent contributes some genes from their chromosomes to spawn a new offspring. Occasionally, one or more of the genes of the offspring may mutate and be different from their parents. Genetic algorithms use the term genetic operators for this exchange and change of genetic material. The two most commonly used genetic operators are recombination (sometimes called crossover) and mutation. In single point crossover, two parents are selected, a point (location) in the chromosome is randomly selected and two offspring are produced as follows: one offspring consists of chromosomes from the first parent up to the crossover point and the balance of the genes are taken from the second parent; and the second offspring is made up from the unused genes of the first and second parent. Multipoint crossover is also possible. The mutation operator involves toggling one or more of the bits in the chromosome when using bit strings. The user defines the probability of mutation (P(m)) and crossover (P(c)). P(m) is usually set quite low. It provides a means of introducing new genetic information into the gene pool and prevents the algorithm from getting stuck in local minimums. Note, P(m)=1 degrades the genetic algorithm into a random search. P(c) is usually set much higher since we do want to simulate evolution from generation to generation by producing new offspring in search of a solution. P(c) regulates what percentage of the new population will be created by crossover and the balance will come from the existing population. Note, it is a good idea to carry over some of the most fit (better) solutions from generation to generation.

Survival of the fittest is accomplished in the genetic algorithm by selecting two parents according to a fitness function for crossover (reproduction). The idea is that better solutions can be found in offspring by combining highly fit parental chromosomes from the current population. Fitness for each chromosome is a function of the underlying genes (dimensions of the problem). It measures how fit the chromosome is with regard to being a solution to the problem. The fitness function is domain dependent and defined by the designer of the genetic algorithm. One common method of selecting chromosomes for crossover is proportional fitness selection where the probability of being selected is proportional to the chromosomes fitness with regard to the whole populations fitness (like a roulette wheel). Another common method is selection based on ranked fitness. In either method, the higher the fitness the more likely the chromosome will be selected for reproduction.

The genetic algorithm creates a population of N solutions and then iteratively – evaluates the fitness of each solution, generates a new population of size N using crossover and mutation with P(c) and P(m) – until a solution is found or termination condition is met (possibly the number of iterations) and is given below.

```
Initialize a population of N chromosomes     (usually done randomly)
Repeat
        Evaluate all chromosomes using fitness function
        Clear new population
        While (new population is not full)
                Select 2 chromosomes from population
                Using P(c) generate two new offspring or
                        Use two selected from population
                Using P(m) mutate offspring
                Insert offspring into new population
        End while
        Update population with new population
Until (termination condition is satisfied)
```

**3. Course Scheduling Problem:**

Each semester universities and colleges need to schedule courses for their next semester. There are many constraints in scheduling courses (or scheduling tasks in general), such as faculty availability and preferences, room allocation, time considerations and leveling courses over all days and periods. The use of a genetic algorithm to solve course-scheduling problems was motivated by Junginger [4]. A simplistic view of scheduling is to match each task in a task list (courses) with a specific time (period) such that there are no schedule conflicts (teaching, room assignments, …). The genetic algorithm that I propose only takes into consideration teaching assignment conflicts – no teacher can teach two courses during the same time period.

The course scheduling problem is then stated as follows – given:

| | |
|---|---|
| C | a set of courses |
| P | a set of periods (times) |
| T | a set of teachers |
| A: T-> POW(C) | a course assignment function for the set of teachers to the power set of C, |

find an assignment from C->P (courses to periods) under the constraint that A(t) cannot teach two courses during the same period over all teachers.

**4. Scheduling Genetic Algorithm:**

The genetic algorithm as described in section 2 was used. Problem encoding, fitness evaluation, selection, crossover and mutation details are given below. User defined parameters to the scheduling genetic algorithm are:

| | |
|---|---|
| P(c) | probability of crossover |
| P(m) | probability of mutation |
| N | number of chromosomes (solutions) for the GA to process |
| C | number of courses |

| | |
|---|---|
| P | number of periods (schedule times) |
| T | number of teachers |
| ITS | maximum number of iterations to evolve the population without finding a solution |

Each teacher is required to teach four courses. For simplicity it is assumed that the first teacher teaches courses 1-4, the second teacher the next 4 and so on.

**4.1 Problem Encoding:**
A schedule solution (chromosome) can be represented with a Boolean matrix. Rows represent periods and the columns represent the courses being offered for the semester being scheduled. An entry of 1 in the solution matrix represents a course being taught in the given period and a 0 not being taught. Note, each column in the matrix representation will have exactly one 1, i.e.) the course must be scheduled for exactly one period.

As an example let C={1,2,3,4,5,6,7,8}, P={1,2,3,4,5}, T={1,2}, A(1)={1,2,3,4} and A(2)={5,6,7,8}. Here we have 8 courses, 5 periods, and 2 teachers with the first teacher teaching courses 1 through 4 and the second teaching 5 through 8. One possible solution is given below in Table 1. The solution is not unique, but it is optimal in the sense that there are no conflicts (a teacher teaching more that one course in one time period).

**4.2 Fitness Evaluation and Selection:**
Proportional fitness is calculated and used for selection in the scheduling genetic algorithm. The fitness of each possible solution (chromosome) is inversely proportional to the amount of conflict within the solution. A better solution will have less conflict. The fitness is calculated as follows:

1. Calculate the teaching assignment (course) conflict of each possible solution in the population by counting the number of periods with more than one course scheduled for each teachers
2. Total the conflicts for all chromosomes (solutions) in the population
3. The independent fitness of each solution is the reciprocal of each solutions conflict
4. Rescale each solution fitness (3) proportionately with respect to the total population fitness (2)

An example is given below.

| Solution | Conflicts | Fitness | Proportional Fitness |
|---|---|---|---|
| 1 | 8 | 1/8 | (1/8)/.801 = .156 |
| 2 | 5 | 1/5 | (1/5)/.801 = .250 |
| 3 | 3 | 1/3 | (1/3)/.801 = .416 |
| 4 | 7 | 1/7 | (1/7)/.801 = .178 |
| totals | 23 | .801 | 1.000 |

Selection of chromosomes for crossover was accomplished by using the proportional fitness of each solution as the probability for being selected for mating. Note, chromosome 3 has the highest probability of being selected and the fewest number of teaching assignment conflicts.

**4.3 Crossover and Mutation:**

Single point crossover was used in the scheduling genetic algorithm. This operation produces two offspring, which are added to the new population. The crossover point is a randomly selected column. Course schedules after the crossover column are exchanged by the two selected chromosomes to form two new offspring. The example below illustrates crossover at (after) course 5.

| | One solution | Another solution |
|---|---|---|
| Period/course | 1 2 3 4 5* 6 7 8 | 1 2 3 4 5* 6 7 8 |
| 1 | 0 0 0 0 0* 1 0 0 | 0 0 0 0 0* 0 0 0 |
| 2 | 0 0 0 0 1* 0 0 1 | 0 0 0 1 0* 0 0 0 |
| 3 | 1 0 0 1 0* 0 0 0 | 1 0 1 0 1* 0 1 0 |
| 4 | 0 1 1 0 0* 0 1 0 | 0 0 0 0 0* 1 0 1 |
| 5 | 0 0 0 0 0* 0 0 0 | 0 1 0 0 0* 0 0 0 |

| | One new offspring solution | Another new offspring solution |
|---|---|---|
| Period/course | 1 2 3 4 5* 6 7 8 | 1 2 3 4 5* 6 7 8 |
| 1 | 0 0 0 0 0* 0 0 0 | 0 0 0 0 0* 1 0 0 |
| 2 | 0 0 0 0 1* 0 0 0 | 0 0 0 1 0* 0 0 1 |
| 3 | 1 0 0 1 0* 0 1 0 | 1 0 1 0 1* 0 0 0 |
| 4 | 0 1 1 0 0* 1 0 1 | 0 0 0 0 0* 0 1 0 |
| 5 | 0 0 0 0 0* 0 0 0 | 0 1 0 0 0* 0 0 0 |

Both original solutions have a conflict of 3 before crossover and conflicts of 3 and 2, respectively, after crossover. The conflicts are underlined. For example the first new offspring has 3 conflicts: teacher 1 teaching courses 1 through 4 has conflicts period 3 and 4, and teacher 2 teaching courses 5 through 8 also has a conflict period 4. This single crossover has improved the population pool of chromosomes. One of the solutions now only has two conflicts to resolve.

Mutation for the scheduling algorithm was implemented as follows:

1. Randomly select a solution in the population
2. Randomly schedule a course during a random period; note, this also involves canceling the originally scheduled course for the solution
3. Insert the mutated solution into the new population.

The user supplies the rate of mutation. Mutation forces diversity into the population. By randomly changing bit values in a chromosome, new solutions are introduced into the population that may not be possible using combinations of existing chromosomes and the crossover operator exclusively.

**5. Results:**

A C program was written to test the scheduling genetic algorithm. Convergence to a solution was dependent on the input parameters and random numbers generated during the simulations. The algorithm was able to find an acceptable solution in most cases.

Results of a typical simulation are displayed in Figure 1. In this example there are 5 teachers, 5 periods, 20 classes and a population size of 10. Notice the total conflict within the population is decreasing over time, hence, a solution is evolving. The average fitness of all solutions in the population is increasing. The algorithm was able to find one acceptable solution within the population of 10 solutions with no conflicts after only 50 iterations.

Note that the decrease in conflict is not monotonic from generation to generation. Although additional conflict may be introduced using the genetic operators of both crossover and mutation, the overall trend is a reduction of total conflict.

The relation of population size and convergence time was tested using 10 teachers, 5 class periods, 40 courses and with probabilities of mutation (.2) and crossover (.8). Figure 2 displays the results of the population size vs. average convergence time (iterations) for 10 trials with each population size. Note, population sizes of 5, 6 and 7 were also tested but only 2, 8 and 8 trials, respectively, converged to a solution within the 1000 iteration imposed time limit.

An increased population size tends to reduce the number of iterations required for convergence to a solution. Increasing the population size does require extra processing time to maintain and manipulate the additional chromosomes. It is not feasible to increase the population size to include all possible solutions since this would be combinatorialy exhaustive. The larger the population size the more raw genetic material the algorithm has to use in its directed search. For small population sizes, with reduced genetic material available, it may be necessary to increase the mutation rate to offer additional variety into the population. This can be seen below.

The relation between mutation rate and convergence was investigated in the scheduling algorithm. Figure 3 displays the average number of iterations required, in ten simulations, for convergence to a solution using 5 teachers 5 periods, 20 classes and 10 chromosomes in the population. When the probability of mutation was set to 0, implying the use of crossover exclusively, only two of the ten trials converged. Although mutation is not an absolute requirement for convergence it significantly improved the convergence rate and reduced the iterations required to converge.

When the probability of mutation was set to 0 and 8 of 10 simulations did not converge to a solution, convergence was to a good solution – there were few total conflicts. Good solutions can be viewed as local minima in the search space of all possible solutions. Mutation provides for a means of escaping from such localized minimum as seen in the results.

**6. Scheduling Genetic Algorithm Extensions:**

There are a number of extensions, in the form of additional constraints, which would make this genetic scheduling algorithm more practical, such as including room assignments (one class per room), prioritizing faculty course requests, including faculty availability, leveling course loads over all times (periods), etc… Each of these extensions will cause additional conflict in the

population of solutions. Some of the conflicts may have precedence over others, for example it may be more important to schedule all classes in their own rooms with no teacher conflicts than to level the load and allow for teacher priorities. Including these and other extensions would require redefinition of the fitness function and changes to the chromosome structure for encoding constraints and conflict.

## 7. Conclusions:

The genetic scheduling algorithm provides an attractive alternate method of scheduling. It employs a structured yet randomized approach in the search process for a solution. The algorithm does not guarantee convergence to an optimal solution, rather, only, that it will converge to some minimum. Increasing the population size appears to reduce the number of iterations required for convergence. The additional expense is in the increased processing time of the larger populations. Mutation is a necessary genetic operator in practical applications. It provides a means of introducing novel, alternate solutions into the population pool that would not otherwise be examined when using crossover and selection with proportional fitness.

## References:


[1] J.Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975, 1992 (second edition)

[2] D.Goldberg, *Genetic Algorithms in Search, Optimization and Machin Learning*, Addison Wesley, 1989

[3] D.Lawrence, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991

[4] W.Junginger, *Evolutionary Algorithms in Management Applications* (Biethahn and Nissen eds.), Springer-Verlag, Germany, 1995


## Tables and Figures:

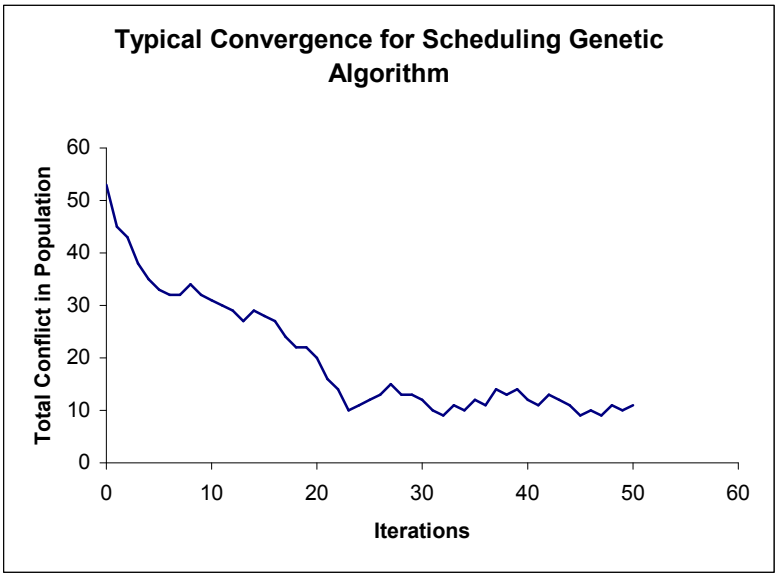


**Figure 1 - Convergence of Scheduling Genetic Algorithm**

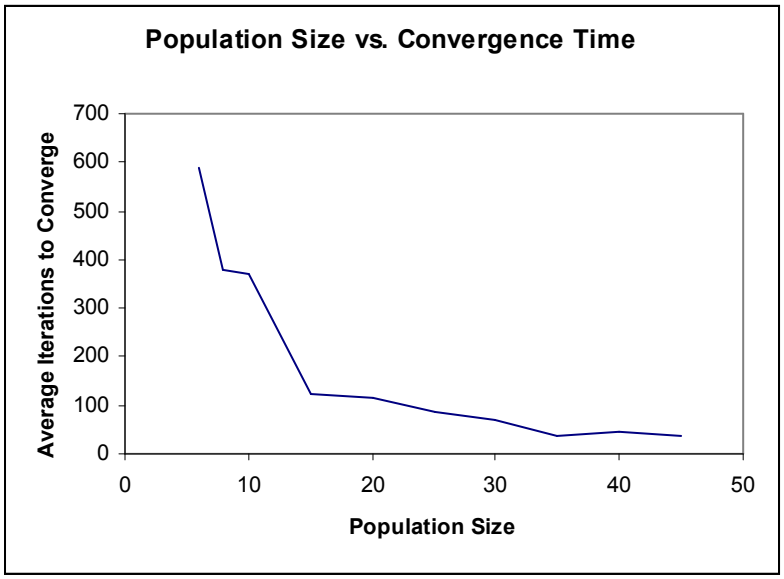


**Figure 2 - Population Size vs. Convergence Time**


**Figure 3 - Mutation Rate Effect**

| Table 1 | | Courses | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Periods | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| | 4 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| | 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

**Table 1 – Boolean Solution Matrix – One Chromosome**