**Genetic Algorithms and their Applications**

**Andrew Miller**

Department of Computer Sciences, Mathematics, and Engineering

Shepherd University

P.O. Box 3210

Shepherdstown, WV 25443

Amille24@rams.shepherd.edu

**ABSTRACT**

*Genetic algorithms are part of evolutionary algorithms. These algorithms are based off of evolution in nature. The evolution of solutions is how it learns to solve a problem. Genetic algorithms belong to a bigger picture which is machine learning. Machine learning is the study of algorithms that learn from data to solve a problem. A genetic algorithm learns how to solve the problem each time it attempts to find a solution. The algorithm evolves after each attempt. It uses the data from each attempt to learn from it and create a better data set. Genetic algorithms can be used to solve many problems. Genetic algorithms are used in many areas like robotics, software, games, and other areas. They can be used in areas outside of Computer Science like the medical field as well.*

**KEYWORDS:** Algorithm, Genetic Algorithm, Machine Learning, Evolutionary Algorithm, Classifier System.

**1. INTRODUCTION**

Many real world problems use computer programs to find a solution to the problem. Having computers learn how to solve a problem in a way to mimic humans learning to solve problems makes it easier to solve them with the speed of computers. Machine learning is used to allow computers to solve problems by learning through a process to find an optimal solution. One can use a searching algorithm to search through every possible solution path but that can be nearly impossible for most real world problems because of the complexity and size of the problem. Some problems increase exponentially when increasing the size of the problem. The traveling salesman problem is one of these problems and we will look at solving this problem with a genetic algorithm. Genetic algorithms evolve as the algorithm attempts to solve a problem. The algorithm learns from each attempt to create a better possible solution to find the solution faster than searching every possible solution. By finding better possible solutions, genetic algorithms can skip over possible solutions that are not better than the better solutions generated during the current attempt.

**2. History of Evolutionary Algorithms and Genetic Algorithms**

During the 1950s and 1960s, computer scientists studied using the idea of evolution as a tool to optimize engineering problems. In all the systems they created, they revolved around the idea of evolving a population of possible solutions using operators based on natural genetic variation and natural selection. In Germany in the 1960s, Ingo Rechenberg and Hans-Paul Schwefel introduced evolution strategies which is a method they used to optimize real-valued parameters for devices like airfoils. They developed the concept of evolution strategies further in the 1970s and it is still being researched to this day. The research of evolution strategies has been conducted independently from the field of genetic algorithms although the two communities have begun to interact with each other [5].

A technique called evolutionary programming were developed by three computer scientists, Fogel, Owens, and Walsh in 1966. This technique took candidate solutions and represented them as finite state machines. These finite state machines were then evolved by randomly mutating their state-transition diagrams and selecting the fittest one. A broader formulation of evolutionary programming is still an active area of research.

In the 1960s, John Holland invented the concept of genetic algorithms. The concept of genetic algorithms was developed further by Holland, his colleagues, and students at the University of Michigan in the 1960s and 1970s. Holland took a different approach than other evolutionary computer scientists. Instead of creating an algorithm to solve a specific problem, Holland looked at how adaptation occurs in nature and tried to find ways to mimic it in computer systems. Holland’s genetic algorithm is a method for moving from one population of chromosomes to a new population by using a kind of natural selection together with the operators of crossover, mutation, and inversion. The selection operator selects the chromosomes in the population that will be allowed to be used to create a new population. On average, the fitter chromosomes are selected over the less fit chromosomes to produce offspring. Crossover exchanges parts of two chromosomes which is based off of recombination between two single chromosome organisms in biology. The mutation operator randomly selects some alleles in a chromosome and changes the allele to a different value. The inversion operator reverses the order of a part of a chromosome. Evolution strategies, evolutionary programming, and genetic algorithms are still the core areas in evolutionary computing today.

**3. Machine Learning**

Machine learning is the study of algorithms that learn from a data set. These algorithms build a model from the input data and then use that information to make decisions or predictions. It is called machine learning because the algorithms learn from the data and are educated when it makes a decision.

Machine learning tasks are broken down into three categories. These categories are supervised learning, unsupervised learning, reinforcement learning. In supervised learning, the machine is given inputs and what is wanted as outputs. The machine then finds how to map the inputs to the outputs. This is considered supervised because the machine is given outputs or the solution. In unsupervised learning, the machine is only given inputs and has to find the correct output or solution. Genetic algorithms fall into this category. In reinforcement learning, the machine interacts with a dynamic environment to perform the goal it is given.

**3. Genetic Algorithm**

Genetic algorithms are just one of the algorithms you will find in the subset of algorithms under machine learning. Genetic algorithms are also a subset of evolutionary algorithms. This is because the concept of genetic algorithms and is based off of natural evolution. In natural evolution, there is inheritance, mutation, selection, and crossover which genetic algorithms have the same processes.

Each attempt made by the algorithm allows the algorithm to learn from it and evolve. The algorithm begins with a data set and then creates a randomly generated population. Then over an iterative process a new generation is created. During the iterations, each individual in the population is tested to see how fit the individual is. This is done with a fitness function. The fitness function is the most important part of a genetic algorithm. It tests each individual in the population and the individuals that have the top fitness are selected. The philosophy of “survival of the fittest” applies for genetic algorithms. These top individuals are then taken to go through the process of mutation, crossover, and selection. The individuals are then used to create a new generation. The iterative process continues through more iterations of the genetic algorithm until a solution is found or too many generations have been created.

**4. Genetic Algorithm in Classifier Systems**

Classifier systems are a machine learning system. There are three levels of activity from the point of view of classifier systems. These three levels are performance system, credit assignment system, and rule discovery system. The performance system interacts directly with the environment. This level is rule based and are message-passing, highly standardized, and highly parallel. For this reason, the performance level is a classifier system. The credit assignment system determines which rules are effective by evaluating the rules using an algorithm called bucket brigade algorithm. The rule discovery system is where new rules are generated to replace less useful rules. This is done by using a genetic algorithm.

**Discovery**

*Genetic Algorithm*

**Credit Assignment**

*Bucket Brigade*

Message Message

From to

**Performance**

*Classifier System*

Input Output

Interface Interface

**Figure 1. Organization of a Classifier System**

Classifier systems are parallel, message-passing, rule-based systems where all rules have the same simple form. The simplest version of all messages must be a fixed length over a specified alphabet. The typical alphabet is a k-bit binary string. The rules have a condition and a corresponding action that is accessed when the condition is met. The condition is tested against messages and finds messages that satisfy the condition to activate the rule. The action is for what message is to be sent when the rule is activated. The classifier system consists of four basic parts. These are input interface, classifiers, message list, and output interface. The input interface translates the current state of the environment into messages. The classifiers define the system’s procedures for processing messages. The classifiers are the rules used by the system. The message list holds all the current messages. The output interface translates some messages into actions that modify the state of the environment. The classifier system basic execution cycle has these steps:

1. Add all messages from the input interface to the message list.
2. Compare all messages on the message list to all conditions of all the classifiers and record all matches.
3. For each set of matches satisfying the condition part of a classifier, post the message specified by its action part to a list of new messages.
4. Replace all messages on the message list by the list of new messages.
5. Translate messages on the message list to requirements on the output interface which produces the system’s current output.
6. Return to step 1.

Each condition must specify exactly the set of messages that satisfy it therefore it is difficult to implement the definition of the condition part of the rule.

An example classifier system is one of an organism or robot. The rules are fragments of the organism or robot. The system has a vision field that provides it with information about the environment and can move around the environment. The goal of the system is to acquire specific kinds of objects in the environment and avoid the other objects. The system creates a message for each object in the vison field. These messages contain the properties of the objects. The messages are fixed length. This system has three kinds of effectors that determine its actions in the environment. Effectors come from the output interface when the translated messages are translated into actions. One effector controls the vison vector which controls moving the field of vision. The second effector controls the rotation of the motion vector which controls the direction the organism or robot moves. This second effector can also control whether to align the vison vector with the motion vector or not. The third effector controls the rate of motion the organism or robot moves.

The following example provided by Booker, Goldbert, and Holland helps with understanding of a classifier system. All credit for this example goes to Booker, Goldbert, and Holland. Let the following be possible properties of objects and their corresponding bit(s) for a message:

The messages will be in the form of a binary string with the rightmost position being d1, the next bit to the left being d2, and the next bit to the left being d3, etc. This system is to represent an organism that eats insects but avoids eating wasps. The system’s goal is to find small, moving objects that are not striped. Let’s say the system has an object in its field of vision. The input interface would create a message with the properties of that object. The system needs to know the origin of the message so two bits are added to the message to act as a tag to identify the origin. In this case, the message is coming from the input interface so the tag that identifies the input interface is 00. In this example, messages have a fixed length of 16 bits. The message would be tested against all the classifier’s conditions. One of the classifiers would identify the goal. In this example, a classifier that represents the goal would have a condition of binary string of 00########000001. The # represents a position with no meaning to the condition so it acts as a wild card. If the object in the system’s field of vision is in the center of the vision field, small, moving and not striped, then a possible message of 0000000000000001 would be created. The message would meet the classifier that represents the goal so the action message from the classifier would then be sent to the message list. An action message would take the same form of a 16 bit binary string but the tag (first two bits) would be different. In this case, let’s say the action message from the goal classifier is 0100000000000000. The meaning of each bit position in the action message will be different from the meaning of each bit position of the input message. The message list would then be translated by the output interface where the output interface holds the meaning of each bit position in an action message. The example action message would mean to the output interface that the output is to align the motion vector with the vison vector and to move fast forward. The output interface would send these outputs to the system and the system would then execute the outputs. This is a basic example and Booker, Goldbert, and Holland [4] go into more detail on how more complex situations can be handled in their paper. One of the concepts Booker, Goldbert, and Holland [4] went over for a complex situation is the concept of building blocks. The system can use rules as building blocks and combine them to handle complex situations. Booker, Goldbert, and Holland [4] introduced these three pairs of rules:

* IF there is an alert and the moving object is near,  
  THEN move at a fast rate in the direction of the motion vector.  
  IF there is an alert and the moving object is far,  
  THEN move at a cruise rate in the direction of the motion vector.
* IF there is an alert, and a small, not striped object is in the vision field,  
  THEN align the motion vector with the vision vector.  
  IF there is an alert, and a large T-shaped object is in the vision field,  
  THEN oppose the motion vector to the vision vector.
* IF there is an alert, and a moving object in the vision field,  
  THEN send a message that causes the vision effectors to center the object.  
  IF there is an alert, and no moving object in the vision field,  
  THEN send a message that causes the vision effectors to scan

Some of the concepts in the rules were previous explained in Booker, Goldbert, and Holland’s [4] paper in more detail so for here I will just give a brief explanation. The system would have rules that keep it on an alert status. The duration of an alert would be determined by a timer. The system would send an alert if there is a moving object in the vision field or if the alert timer wasn’t zero. If the alert timer wasn’t zero and there were no moving objects then the system would decrease the alert timer. The T-shaped object is from an example of detecting a compound object by the relations between its parts. The T-shaped object can be thought of as a hawk in the example of an insect eater. A combination of rules can be obtained by choosing one rule from each pair of rules. This combination can provide a potentially beneficial behavior to the system. In this case of three pairs of rules there are eight possible combinations. The combinations of rules from pairs of rules would produce distinct combinations. Taking an approach using build blocks provides a combinatorial advantage. The authors added more examples but I wanted to touch on this one as build blocks will come up later on.

Booker, Goldbert, and Holland [4] compared the relation of classifier systems to other artificial intelligence problem solving systems. The problem solving and learning mechanisms in classifier systems were based off of adaptive processes in both natural and artificial systems. Classifier systems have two weaknesses. The first is the rules are written in a language that lacks descriptive power in comparison to what is available in other rule-based systems. The rules cannot be used to express arbitrary, general relationships among attributes. The second weakness is since rules can fire off simultaneously, control issues are raised that conventional rule-based systems do not have. For these two weaknesses, classifier systems seem to be an unconventional approach when used to build a conventional expert system. Classifier systems have advantages and these advantages show when classifier systems are used to solve problems they were designed to solve. Classifier systems are complex systems and genetic algorithms are a key part of these systems.

**5. Problems Solved with Genetic Algorithm**

Genetic Algorithms are used in many areas to solve real problems that have faced this world. One problem that was solved with a genetic algorithm was the design of an antenna that would create the best radiation pattern. NASA found the design in figure 2 by using a computer program that used a genetic algorithm.



**Figure 2. Shape of NASA ST5 Spacecraft Antenna**

Another problem is the verification and validation of software. Praveen Srivastava and Tai-hoon Kim believe this area of software engineering where progress towards automation has been slow [2]. They found a genetic algorithm that performs better than other methods on a small scale.

Another problem in the medical field is attribute weighting for a medical data set. Attribute weighting is used in classification methods that classifies a new case into a class [3]. Kirsi Varpa, Kati Iltanen, and Martti Juhola found that using a genetic algorithm approach, the accuracies of the classifications were improved.

A classic problem is the traveling salesman problem. Two of my fellow classmates and I looked at solving this problem with a genetic algorithm. The traveling salesman problem is a salesman starts at a city and must travel through a set number of cities only once and must get back to the starting city. The problem is finding the shortest distance that meets all the requirements. We looked at creating a Java program that would use a genetic algorithm to find the path between all the cities with the shortest distance. We approached the problem by placing cities on a xy-plane. We could then use the distance formula to find the distance between any two given cities.

Our program contained a city class, tour class, tour manager class, population class, genetic algorithm class, draw starting cities class, draw cities class, number out of range exception class, and a main traveling salesman problem class. The city class would define what a city is. A city would have an x-coordinate and a y-coordinate. We then would need methods to get the x-coordinate and to get the y-coordinate. We, also, created a method that would return the distance between the current city object and a city object passed into the method. The tour class defined the order the cities would be traveled through to go through all the cities. The tour would start and end with the same city to meet the requirements of the traveling salesman problem. The tour class contains a method that generates the first tour of the cities and the order is random. The tour class, also, contains a get method to get a city from the tour and a set method to set a position in the tour to a city. We then needed a method that would get the total distance of the tour. The fitness function is contained in this class and we calculate a tours fitness by creating a fraction with the tours distance. We take one and divide it by the tours distance to get our fraction. The reason for this is because the shorter the distance is the higher the fitness is of the tour. If we tried to compare the raw total distance of the tour, the higher number would seem to be a higher fitness. With fractions, the fraction with a lower number denominator is higher than a fraction with a higher number denominator. This would give us a tour with a shorter distance would be considered higher than a tour with a larger distance.

The tour manager class is there to help create the very first tour. We add each city to the tour manager. Once all the cities are added, we can then use the tour manager to create the first tour. The tour manager class has an add city, get city, and number of cities methods to give us the functionality we needed. The population class is used to create a population of tours which would be one generation in the genetic algorithm. The population class creates the first population when set to initialize by using the tour class’s generate individual method. After the first population is created, the genetic algorithm will use operators to create new tours for the rest of the populations. The population class has a get tour and set tour methods. Also, the class contains a get fittest individual method which returns the tour with the shortest total distance. This will be used by the genetic algorithm to evolve the tour and create new tours.

The genetic algorithm class is where the program evolves solutions to create new generations. The class has an evolve population method that takes fittest individual from the previous generation and saves it to the new population. The method then applies selection, crossover, and mutation to the rest of the individuals from the previous generation to create a new generation. The class also defines methods for selection, crossover, and mutation.

The main traveling salesman problem class is where we run the genetic algorithm and define the graphical user interface for the program. The class defines all the GUI elements and has an inner button handler class for the two buttons on the GUI. The start button is where we run the genetic algorithm to find the fittest solution at the end of the max generations that were set. The add city button allows a user to add a city to the xy-plane to add a city to the problem. The add city button uses the draw starting cities class to add the city to the panel that draws all the cities on the GUI.

The draw starting cities class and the draw cities class are similar classes. The draw starting cities draws the cities on the panel without connecting the cities. The draw cities class draws the cities and connects them together with the tour the genetic algorithm had found. The number out of range exception class is a custom exception class that is used during validation of the coordinates when a user adds a city to the program. The program we created to solve the traveling salesman problem helped us understand genetic algorithms more and helped us learn how to use Java’s GUI libraries to create a GUI for our program.

**6. Real World Uses**

Genetic algorithms have real world uses and some of them are not a use that you would expect. One use of genetic algorithms is for automotive design. A genetic algorithm can be used to come up with designs that are aerodynamic for cars and planes by using computer modeling with the genetic algorithm. Also, a genetic algorithm can provide combinations of materials that would make cars faster, lighter, and more fuel efficient. Using a genetic algorithm to come up with the combinations of materials would save the cost of researching each combination in a laboratory.

Another use is for engineering design. Genetic algorithms can be used to optimize the structural and operation design on buildings, factories, etc. Not only can a genetic algorithm be used to create a design but it can be used to find weaknesses in a design. Another use is a joke and pun generator. I would have never thought a genetic algorithm could be used in this way but it can be used to return jokes and puns based off of a word or subject.

Another use is for encryption and code breaking. A genetic algorithm can be used to create and break encryptions. Every time an encryption’s complexity is increased, a genetic algorithm can then be created to break the encryption. Another use is for computer-aided molecular design. A genetic algorithm can be used to aid in the understanding of protein folding and analyzing the effects of substitutions on those protein functions.

**7. Conclusion**

Using genetic algorithms for real world problems helps find solutions faster. Computers can learn without the need of humans to give the computer inputs except for the first time. Real world complex problems are being solved with genetic algorithms where before these problems might not have been able to be solved in a reasonable amount of time. My classmates and I solved a classic problem in Computer Science with a genetic algorithm and learned how useful they can be. There are many real world uses for genetic algorithms and as machine learning evolves, genetic algorithms may become building blocks to more complex algorithms and to complex systems. Classifier systems are already using genetic algorithms in their systems. Genetic algorithms or a form of genetic algorithms will be around for a long time to solve the complex problems of the world.

**REFERENCES**

[1] Bhasin, H., and Bhatia, S., “Application of Genetic Algorithms in Machine Learning,” International Journal of Computer Science and Information Technologies, Vol. 2, No. 5, 2011, pp. 2412-2415.

[2] Srivastava, P.R., and Kim, T., “Application of Genetic Algorithm in Software Testing,” International Journal of Software Engineering and Its Applications, Vol. 3, No. 4, October 2008, pp. 87-95.

[3] Varpa, K., Iltanen, K., and Juhola, M., “Genetic Algorithm Based Approach in Attribute Weighting for a Medical Data Set,” Journal of Computational Medicine, Vol. 2014, 2014.

[4] Booker, L.B., Goldberg, D.E., and Holland, J.H., “Classifier Systems and Genetic Algorithms,” Machine Learning: Paradigms and Methods, pp.235 -282 1989 :MIT Press/Elsevier

[5] Mitchell, M., An introduction to genetic algorithms, MIT press, 1998.