An Evaluation of Genetic Algorithm Solutions in Optimization and Machine Learning

Christina Carrick and Kevin MacLeod
Department of Mathematics and Computing Science
Saint Mary's University, Halifax, Nova Scotia, Canada, B3H 3C3
E-mail: GEN3001@husky1.stmarys.ca

Abstract

A genetic algorithm (GA) is a search algorithm in which a solution is evolved through natural selection. A new set (population) of solutions is created from a random partial exchange of information between the best solutions of the previous generation. This results in increasingly better solutions in successive generations. Traditional search methods have either been calculus-based, enumerative, or random. Each of these methods has limitations which GAs are able to overcome. GAs require neither a continuous search space (as do calculus-based methods) nor a lengthy amount of time and processing (as do enumerative and random searches). This contributes to the robustness of GAs and makes them more applicable to a wider range of problems than the traditional methods. The Travelling Salesman Problem and "maze-learning" have been widely studied. A number of heuristic paradigms have had varying degrees of success in solving these problems. In this paper we implement GA solutions to each and compare their performance to that reported for other techniques.

1. Introduction

Genetic algorithms are a search method first introduced by Holland at the University of Michigan in the 1960s [1]. Based on the principles of biological evolution, reproduction and survival of the fittest, GAs were used mainly for biological simulation at that time. By the 1970s Holland and his colleagues were exploring GAs as a problem-solving tool. In the past decade, GAs have been used to solve a variety of problems involving large solution spaces.

2. Traditional Search Methods

Traditional search methods fall into one of three categories: random, enumerative and calculus-based. Each of these methods has its own benefits and drawbacks [1]. A random search simply involves picking random points in the solution space. The primary benefit to this type of search is that it can be used on any type of problem. Unfortunately, it could take a very long time to get even a good solution, and there is no guarantee of getting one.

An enumerative search consists of listing all of the possible solutions and taking the best one. This method is guaranteed to find the best possible solution to a problem. As with the random search, however, finding that solution could take a length of time which makes its use impractical for any non-trivial problem. Heuristics can sometimes be used to limit the number of solutions examined, but this eliminates the guarantee of the best possible solution.

The calculus-based search method includes paradigms such as the Greedy algorithm and hill-climbing [2]. The common feature in these searches is the process of continually choosing the next step which gives the greatest immediate benefit, each time getting an increasingly better solution. There is, therefore, the guarantee of an optimal solution to the problem at hand using this method. The

drawback is that there is no back-tracking mechanism and so it is possible to land on a local maximum instead of the best solution, depending on the starting point of the algorithm. This makes the Greedy algorithm an excellent choice for problems which contain only one maximum in the solution space, but the more maxima there are, the more efficiency degrades. It should also be noted that if the calculus method is being used to maximize a function, that function must be continuous and differentiable. Since many problems have a scattered, non-continuous solution space, the calculus method can not always be used.

3. Search by Genetic Algorithms

GAs are a robust search method that gives good performance over a wide variety of problem types [1]. Using structured randomness, probabilistic transition rules, and simultaneous search at a number of points in the solution space, GAs do not experience the restrictions of the calculus-based search methods. While GAs do not guarantee the optimum solution, the probability of finding at least one of the better solutions is greater than with the Greedy algorithm since more than once search point is used.

GAs use a coding of the parameter set rather than the parameters themselves. The coded parameters form a string (chromosome), where each string represents a solution to the problem. An objective payoff function is used to measure how well a particular solution solves the problem (fitness). Better solutions are subsequently evolved within a population of chromosomes over a number of generations.

3.1. How GAs Work

Evolution takes place through three main operators: reproduction, crossover and mutation. During reproduction, strings are chosen from the population to mate in the next generation. The number of children (copies) awarded to each string is proportional to that string's fitness relative to the fitness of the entire population. In this manner, strings which are better solutions have higher fitness and receive more children in the next generation.

The crossover operator is an information exchange between two randomly chosen strings resulting from reproduction. Simple crossover consists of randomly choosing a position P within the length of the strings and exchanging information as shown in Figure 1. All elements to the left of P in string A get copied to string A' of the new population, and all elements to the left of P in string B get copied to string B' of the new population. Then all elements to the right of P in string A get copied into the corresponding elements of string B' of the new population. All elements to the right of P in string B get copied to the remaining elements of string A' in the population.

After the new population has been built through crossover, the mutation operator randomly alters string element values (alleles) according to a chosen probability. This probability is generally set fairly low; perhaps 1/1000 to 1/100 of the allele values will be changed. Mutation is a minor operator in that it is not meant to directly find good solutions. Rather, mutation serves to expand the search space slightly and prevents a solution from being permanently lost or out of reach.

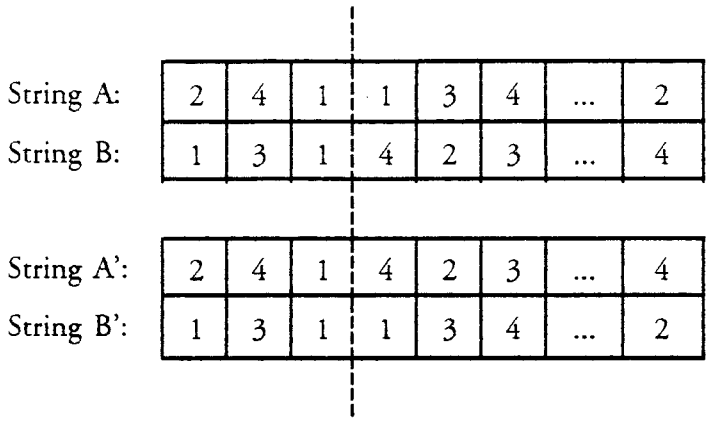

Figure 1

4. Training GAs to Run a Maze

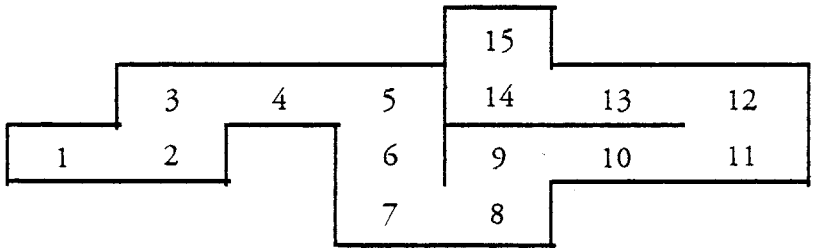

Figure 2

The maze-running problem was easily solved using the simple genetic algorithm as described above. Given a maze such as the one in Figure 2, the problem was to have the GA learn the correct sequence of moves necessary to get from point A to point B without running into any walls or backtracking along the path.

4.1. Coding the Problem

The parameters to the problem were the four directions up, down, left, and right. These were coded into the genetic algorithm as the numbers 1, 2, 3, and 4, respectively. The chromosomes were then n-element strings of numbers 1 through 4, n being the number of moves necessary to get from point A to point B. Each chromosome was then a sequence of moves representing a possible solution to the problem.

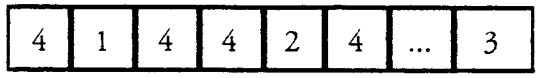

Figure 3

Each chromosome was rewarded for how far it travelled along the path without hitting a wall or backtracking. Fitness was calculated according to the following function:

$$\text{Fitness} = (\text{distance})^2 * \text{DISTANCE\_VALUE}$$

where DISTANCE_VALUE was a parameter usually set to a value of about 5. The distance was squared in order to give substantially more value to those chromosomes which travelled the farthest.

4.2. Results

Table 1

Maze Size	Population Size	Maximum Generations Allowed	Maximum Position In Maze (on average)	Generation During Which Maximum Positions Achieved
5	8	10	5	6
9	10	30	7	15
13	16	40	9	24
15	16	50	9	21

GAs were used to attempt to solve mazes of size 5, 9, 13, and 15. The population sizes used were 8, 10, 16, and 16, respectively. The chromosomes were allowed to evolve for 10, 30, 40, and 50 generations respectively. A number of trials were run for each parameter set since GAs use randomness and probability to find solutions. Table 1 shows the maximum position in the maze (on average) the GAs were able to achieve given these parameters, along with how many generations were used to obtain those maximum positions.

4.3. Comparison of GA search to traditional methods

Traditional methods have varying degrees of success in solving the maze problem. Using a random search, each try at a solution would have a $1/(4^n)$ chance of being the optimum solution, where n is the number of moves necessary to get from point A to point B. Enumerative search requires that 4^n solutions be listed. Even for a small maze requiring only 10 moves to get through, these two methods are searching, one by one, 1,048,576 points in the solution space. The random search might find the solution before the 1,048,576th try, but it might also take longer since each try is independent and may be duplicated. The Greedy algorithm would solve the problem by looking in all four directions (up, down, left, and right) and choosing the direction which would advance it farthest along the maze. This method would find the optimum solution in 4n steps.

The GAs were able to find the correct sequence of moves necessary to get to position 9 in the maze in 21 generations. This is an improvement over the random search (a 1/262144 chance of picking the correct sequence), enumerative search (262144 solutions to examine), and calculus-based search (36 solutions to examine). The GA did not solve for 10 positions in a maze in 50 generations, however. The calculus-based method is guaranteed to find the correct 10-move sequence in 40 steps. The maze is a problem for which calculus-based search is well-suited, out-performing the simple GA (ie without advanced operators) for mazes larger than 9 steps.

5. The Travelling Salesman Problem

The Travelling Salesman Problem has been widely studied and a variety of methods have been used to attempt to solve it, though none guarantee the optimum solution [3]. Given a number of cities n, the goal is to find a route (ordering) which visits each city and then returns to the city of origin such that the distance covered is minimized. Genetic algorithms have been the latest attempt at solving this problem.

5.1. Coding the Problem

Each chromosome was a string of n elements, where n was the number of cities to be visited. Each element could take on values 1 through n and each number represented a city with its own map coordinates. Each number between 1 and n occurred once and only once in each string, so that each string represented a complete trip including all n cities. The order in which the city numbers appeared in each string was the order in which the cities were visited for that solution, returning back to the city in the first string element to complete the circuit.

The fitness of each string was calculated according to the following function:

$$\text{Fitness} = \text{Cmax} - (0.1 * (\text{distance travelled})^2)$$

As with the maze problem, the distance was squared in order to spread out the fitness values as the chromosomes became more and more fit. The value Cmax was used to turn the minimum distance problem into a maximization problem to be solved by the GA. Cmax varied according to the number of cities in the problem, since the fitness should be positive and the distance travelled grows with the number of cities.

5.2. PMX Operator

The Travelling Salesman Problem is GA-Hard. This means that the problem can cause two highly fit strings to cross over and produce two strings with lower fitness [1]. Consider the following example:

string A: 1 2 3 4
string B: 4 3 2 1

Map: 2* 3* 1* 4*

Figure 4

Figure 4 shows four cities and two solution strings. Both strings are optimum paths for the given cities. If these cities cross over at position 2, however, we get the two strings depicted in Figure 5, both of which are very poor solutions and are specifically inferior to A and B.

string A' : 1 2 2 1

string B' : 3 4 4 3

Figure 5

The reason for this breakdown is that, unlike the maze problem where it could be said that a particular value at a fixed position was good or not good, the Travelling Salesman Problem is concerned with *relative* positioning within a string. For example, whether a 4 is good in position 1 is dependent upon the values of the other string elements. The Travelling Salesman Problem is purely an ordering problem, and so required an advanced crossover operator which would exchange positional information instead of *content* information. This was achieved through the use of the Partially Mapped Crossover (PMX) operator [4].

The PMX operator takes two strings and randomly chooses two crossover points. The *positional* information between these two points in the strings is exchanged. Using the previous example and choosing crossover points of 1 and 2 the information exchange takes place as follows (see Figure 6):

string A: 1 2 3 4 -> string A': 4 2 3 1 -> string A": 4 3 2 1 -> ->
string B: 4 3 2 1 -> string B': 1 3 2 4 -> string B": 1 2 3 4

Figure 6

1. In string B position 1 has value 4. So in string A move the 4 to position 1. The 1 from position 1 in string A takes the position vacated by the 4. In string A position 1 has value 1. So in string B move the 1 to position 1. The 4 from position 1 in string B takes the position vacated by the 1. This process has just swapped the 1 and 4 *within* string A and the 4 and 1 *within* string B to get strings A' and B'.

2. In string A position 2 has value 2, and in string B position 2 has value 3. So in string A' swap the 2 and the 3, and in string B' swap the 3 and the 2. The result is the two strings A" and B" and the crossover routine gives two highly fit strings.

It should be noted that if crossover points of 2 and 3 had been chosen in the example, the two resulting strings would have had lower fitness values than the parent strings. The PMX operator is still valuable, however, because:

a) the breakdown in crossover occurs less frequently as than in the simple crossover routine,
b) the breakdown is not as severe, with fitness falling only slightly,
c) it is more likely that the GA will recover from the minor breakdown in the PMX crossover than in the simple crossover.

5.3. Results

Table 2

# Cities	Population Size	Maximum Generations Allowed	Maximum Fitness Achieved (on average)	Generations Achieved (on average)	Fitness of Optimum Solution
4	4	10	10.9129	1	10.9129
4	6	10	10.9129	1	10.9129
4	8	10	10.9129	1	10.9129
8	8	30	66.6519	15	69.4666
8	12	30	67.7017	22	69.4666
8	16	30	66.8991	8	69.4666
14	14	50	177.9270	18	206.6335
14	20	50	174.9348	25	206.6335
14	28	50	170.3479	17	206.6335

The Travelling Salesman Problem was examined for problem sizes of 4, 8, and 14 cities whose map layout was chosen randomly. Various population sizes were chosen for each of these, starting with the population size equal to the number of cities in the problem. The program was allowed to run for 10, 30, and 50 generations for the 4, 8, and 14-city problems, respectively. As can be seen in Table 2, such large numbers of generations were not necessary to get near-optimal solutions. (As with the maze problem, these results are averages over a number of trials.)

5.4. Comparison of GA search to traditional methods

As with the maze problem, the time and space requirements for using a random or enumerative search with the Travelling Salesman Problem grow very rapidly with the size of the problem. If n is the number of cities to be visited and m is the number of visits we are willing to allow in the solution, each try in a random search has a $1/(n^m)$ chance of being the optimum solution and there are n^m solutions to be listed for the enumerative method. The best possible circumstances are when m = n, and then there are still n^n solutions to be examined. The Greedy Algorithm has been the most natural choice to attempt to solve the problem. If from each city its nearest neighbor is chosen as the next city to visit, a better-than-average path will result. This method, however, can result in a very expensive closure of the circuit and can ignore many short distances [3].

The random search is not guaranteed to ever find near-optimal solutions, let alone within 50 tries. The enumerative method will find the optimum solution, which is better than the GA solutions, but only after quite a lengthy search. For the 8-city problem, 40320 solutions must be examined to gain

an improvement of 2.54% over the GA solution which took only 22 generations to evolve. As for the calculus-based search method, it's 8-city solution may be less fit than that of the GA solution, depending on which city is chosen as the starting point of the algorithm. The GA consistently found near-optimal solutions.

6. Conclusions

The GA was not as good at solving the maze problem as the calculus-based method, but it out-performed all three methods in solving the Travelling Salesman Problem for 4, 8, and 14 cities. (This statement assumes that the increase in solution fitness over GA solutions is not worth the search time necessary for an enumerative search.) GAs were useful in getting *good* solutions in both types of problems, showing the robustness of the algorithm. It should also be noted here that the maze problem was examined using only the three basic operators common to all GAs. An advanced operator or improved objective function might well improve GA performance in the maze problem.

References

[1] Goldberg, David G, "Genetic Alogithms in Search, Optimization, and Machine Learning". Addison-Wesley, 1989.

[2] Noyes, James L, "Artificial Intelligence With Common Lisp". D. C. Health and Company, pp 157-199, 1992.

[3] Gould, Ronald, "Graph Theory". Benjamin/Cummings Publishing Company, Inc, pp 147-148, 1988.

[4] Goldberg, David E, Lingle, Robert, "Alleles, Loci and the Travelling Salesman Problem". Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp 154-159, 1985.