Autonomous Robots and Genetic Algorithms: Part II - Robot Path Planning with a Genetic Algorithm

EYLER ROBERT COATES, JR [1], WILLIAM A. RUSSELL, JR.[1], KIMBERLY S. JOHNSON[2], KENNY COKER[3], EUGENE SNIPES[1]

[1]University of Southern Mississippi, School of Engineering Technology, Hattiesburg MS
[2]Lockheed Martin, GMPO Team, Stennis Space Center, MS
[3]Seagate Technologies, Moorpark, CA

Abstract

An autonomous vehicle was constructed containing a Motorola 68HC11 microprocessor, a Vector V2X electronic compass module, infrared collision avoidance sensors, and a dc motor powered tracked platform. The robot completed in the 1997 IEEE Southeastern Conference (SECON) hardware contest hosted by Virginia Tech in Blacksburg, Virginia. The objective of the competition was to retrieve as many metal balls as possible in a timed, head-to-head competition with another robot. The metal balls were randomly placed at polar grid coordinate crossings on a twelve-foot by twelve-foot plywood playing field. The polar grid coordinate system was painted on the playing field. The random ball locations on the polar grid were provided to each team fifteen minutes before the start of each round.

The initial inclination was to follow the polar grid as painted on the playing surface. However, this would probably not be the shortest path to collect all the balls. Since points are awarded to the team with the most balls collected in the shortest time, a genetic algorithm and electronic compass were used to optimize the distance traveled. There are twenty balls on the table and using enumeration methods of finding the shortest pathway, the solution would require all 19! (or 1.21645×10^{17}) possible paths to be evaluated. Computing the total distance of any path to collect all 20 balls requires about 0.008 seconds of computational time. At this rate, it would require about 30,858,726 years for one to be assured of the optimal path. The genetic algorithm approach was chosen to seek out a superior collection path. This approach while it does not guarantee optimality, has been shown to quickly find good solutions to difficult combinatorial problems. Using the genetic algorithm approach, after only 5 minutes and examination of only 37,500 paths, excellent results were obtained for an efficient collection path.

Introduction to the Traveling Salesman Problem

There is a class of optimization problems that have been traditionally difficult to solve. This includes combinational problems such as the traveling salesman problem (TSP). The traveling salesman problem can be described as follows: The traveling salesman must visit every city in his territory exactly once and then return to the starting point. Given the distance (or cost) of travel between all cities, how should he plan his itinerary for minimum total distance (or cost) of the entire tour?

This simple sounding problem becomes very difficult to solve as the number of locations increases. Consider a tour of 4 cities, from any given starting point, we have 3 cities to choose from. Having made that choice we now have 2 cities left to choose from. Finally there is only one choice for the last city. We have 3 x 2 x 1 possible paths. However, we should note half of these paths are merely inversions of another path. So some could say that we have only 3 unique paths. Generalizing, the number of paths is (N-1)! or the number of unique paths, not counting inversions, is (N-1)!/2. Either case quickly becomes a formidable number as the number of locations increase. For example, 5 locations yield 24 paths; 10 locations yield 362,800 paths; 15 locations yield 87,178,291,200 paths and 20 locations yield 1.21645×10^{17} possible paths! This hard-to-solve but simple-sounding problem becomes rather important because many engineering and scientific problems can be reduced to a TSP. Also, many other NP-complete optimization problems are transformable into each other which increases the interest in solving TSP efficiently [5]. In fact, the literature on TSP is enormous. A good start in the literature would include the comprehensive survey by Lawler *et al.* [4].

Introduction to Genetic Algorithms

Genetic algorithms (GA) have as their model the evolution process found in nature. Interest in GA began over 20 years ago with the work of Holland [3]. Since then the area has achieved wide attention. Some fundamental reading in the area would include Goldberg [2]. Today, there are many books and conferences on the subject.

A very fundamental description of an evolutionary process that improves the species might be: Suppose one starts out with an initial population. Some of these population members will be stronger and/or healthier than the other members. The healthier ones will tend to survive to the reproduction age compared to the weaker ones according to the Darwinian theory. Also, because of the basic principle of transference of hereditary factors from parents to offspring discovered by Mendel, the children of strong parents will have a tendency to also be strong. The children will not be exact copies of their parents but will contain characteristics of both of them to various degrees. And the children will have some of their own unique characteristics. We should mention that the strongest are not guaranteed to survive, nor their children guaranteed to have inherited the best characteristics of their parents, but the tendency is slightly in their favor.

The genetic algorithm attempts to emulate this evolutionary process to solve engineering problems. The process starts with a collection of feasible solutions to a problem. Each solution is considered to be an "individual." The parameters of the solution are considered the "gene." The collection of these initial solutions is the initial "gene pool." The GA will then evaluate each member of the initial population. The fitness of the "individual" is based on the worth of the solution to the problem. The best solutions are given a higher chance of survival. The GA then stochastically eliminates a portion of the initial population with a bias toward eliminating the poorer solutions. Finally, new solutions are generated from the remaining members of the population. There are numerous methods for generating the new members. The basic methods include crossover and mutation. Crossover means that we take portions of the genes (solution parameters) from two parents and combine them to form a new gene. This new gene defines an individual (solution), which has characteristics of both parents. Mutation in GA is done by altering a small portion of a gene in a random fashion to give the member an entirely new characteristic. After this new generation has been formed, the entire process is repeated again beginning with the evaluation of the fitness of the new members of the generation. The process can be repeated as long as it is practical. A flowchart of a typical GA is given in figure 1.

Based upon the flowchart, the GA can be a simple and elegant method to solve a variety of problems. The representation of the solution parameters as the gene and the determination of fitness value by evaluating the solution tend to be problem specific and represent the main challenge to applying GAs to a particular problem.

Data and Equipment Used in Research

Visual Basic was used as the programming language for the genetic algorithm since it is a good prototyping tool and had good graphic capabilities that could show the progress of the robot path as it evolved into a shorter path. The computer used was a Pentium-based laptop so that the path for the robot could be easily computed on site at the competition.

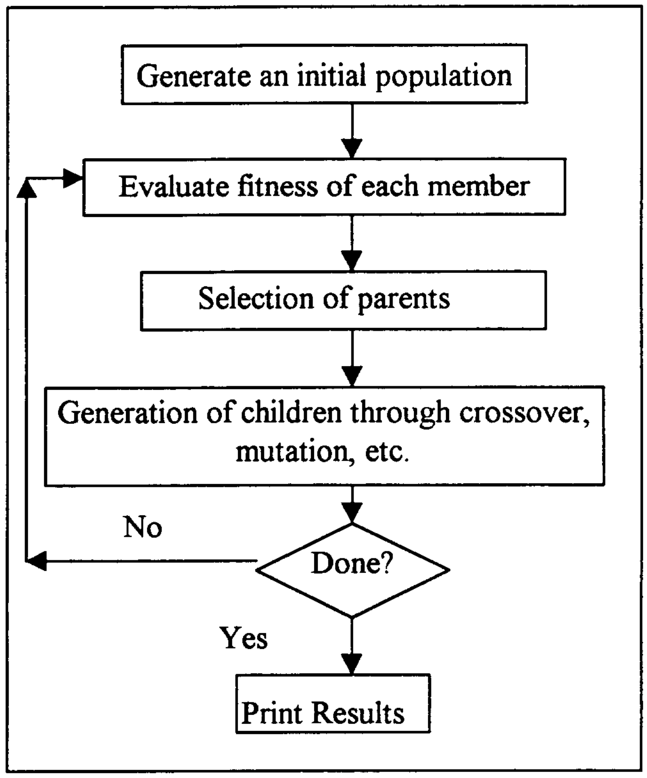
Generate an initial population
Evaluate fitness of each member
Selection of parents
Generation of children through crossover, mutation, etc.
No
Done?
Yes
Print Results

Figure 1 Flowchart of GA

Observations and Algorithm Methodology

As mentioned earlier, the representation of the solution parameters as a gene is application dependent. In the TSP, one way of representing a tour is to first assign integers to each location. A tour could be represented by a complete listing of the integers. For example, a 5-city tour that began in location 1 could be represented by (1,2,5,4,3). Thus a simple listing of the tour locations would become the representation of the gene of an individual solution.

The usual GA calls for two parents to generate a child. This two-parent (sexual) reproduction causes a complication for the TSP. That is because it is possible that the combinations of genes from the two parents would represent non-feasible paths. For example, suppose that one parent has the gene (1,2,5,4,3) and another parent has the gene (5,1,4,2,3). Now suppose that we combine the genes with a crossover operation so that the child has the first 3 "chromosomes" from the first parent and the last two "chromosomes" from the second parent. The child's gene would be (1,2,5,2,3) which would represent an infeasible solution since the location number 2 is visited twice and the location number 4 is never visited. There are two methods of getting around the problem of infeasible genes. One involves either repairing or ignoring infeasible genes. The other method is to avoid creating infeasible genes at the reproductive stage.

One method that avoids the creation of infeasible genes for TSP has been proposed by Chatterjee *et al.* [1]. Their method uses an alternative type of GA that relies on asexual or single-parent reproduction. The idea is that the gene of a single parent can be cut and spliced any number of times and places. When the slices are reassembled, the new gene (child) will still contain each location only once. For example, the single parent (1,2,5,4,3) can be cut randomly into (1,2), (5,4) and (3). A random re-assembly could possibly produce a gene that looks like (5,4,3,1,2) which is naturally feasible. Thus, a crossover operation from a single parent can be accomplished.

Chatterjee *et al.* [1] also proposed a method of mutation that would preserve feasibility in the child gene. A mutation-like operation can be achieved by simply reversing one or more of the slices. For example, the slice (5,4) could have been mutated into (4,5) before the re-assembly and the result of our example could have looked like (4,5,3,1,2).

Finally, the evaluation of the fitness function for each member of the population is simply a computation of the actual Euclidean distance of the path represented by the gene. During the selection of the survivors of a generation, the members were randomly paired. The member with the smallest distance was given a greater chance of survival and the other member had the greater chance of elimination.

Originally, GA theory proposed that the initial generation be randomly chosen. However, researchers have found that the performance characteristics of GA's can be enhanced when the initial generation has been carefully chosen. Chatterjee *et al.* [1] mentioned that when they applied the nearest-neighbor (greedy) algorithm to find a beginning solution to start the GA, the convergence time of the GA to optimal solutions was reduced by half. The nearest-neighbor algorithm involves creating a path to the nearest location and continuing in that manner until there are no locations left. This heuristic very often produces good if not optimal solutions for small-scale traveling salesman problems although there are patterns that will "trick" the heuristic into longer paths. The nearest-neighbor algorithm does have the advantages of providing a fairly good path in a short period of time (about 1 second) which makes it an excellent tool for providing a starting point for the GA. The starting location and ending location of the path were fixed by the rules of the competition. Normally, this would present a complication to most TSP algorithms. However, this does not present a problem to the GA because the internal workings of the GA are problem independent with the exception of the evaluation of the fitness function. In this case, the fitness function is the Euclidean distance of the entire path. Thus, all that is required to adapt the GA to this variation of the TSP is that the distance calculated would be required to consider the starting and ending locations.

Results

An algorithm was designed to exploit the characteristics noted in the previous section. Starting out with nearest neighbor heuristic to find a good initial solution, the GA maintained a 75-member population for 500 generations. An example of the initial path created by the nearest neighbor algorithm is given in Figure 2.

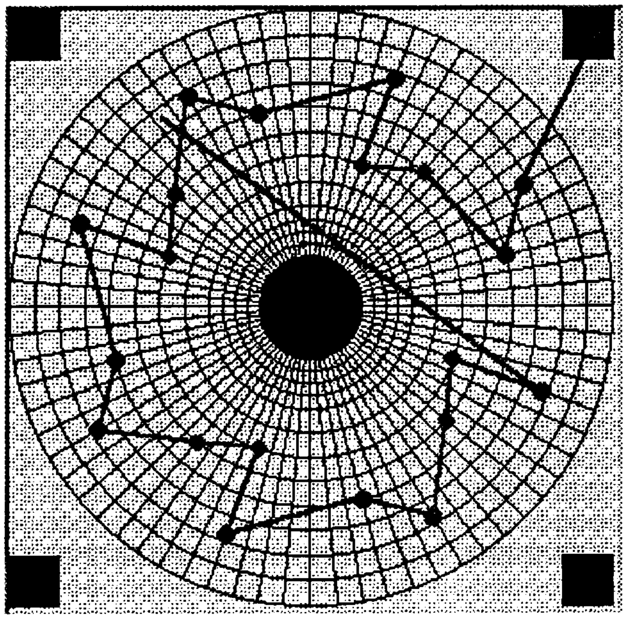

Figure 2 Initial Path From "Greedy" Heuristic

The program kept track of the best path produced over the 500-generation period. An example of the progress of the best path over the evolutionary process is shown in Figure 3. Thus, in all, for a particular problem, 37,500 (75 individuals x 500 generations) possible paths were evaluated. This required about 5 minutes of computer time on a Pentium-based laptop. The population size of 75 and the 500 generation run length were suggested by experimental results from Chatterjee *et al.* [1] who generated a table based upon different size problems.

The GA would often generate a path that would range up to 30% shorter than the path given by the nearest-neighbor heuristic. This is remarkable considering that the nearest-neighbor heuristic produces very good solutions on its own. An example of the final path that was produced from the initial path in Figure 2 is given in Figure 4.

There are twenty balls on the table and using enumeration methods of finding the shortest pathway, the solution would require all 19! (or 1.21645×10^{17}) possible paths to be evaluated. Computing the total distance of any path to collect all 20 balls requires about 0.008 seconds of computational time. At this rate, it would require about 30,858,726 years for one to be assured of the optimal path.

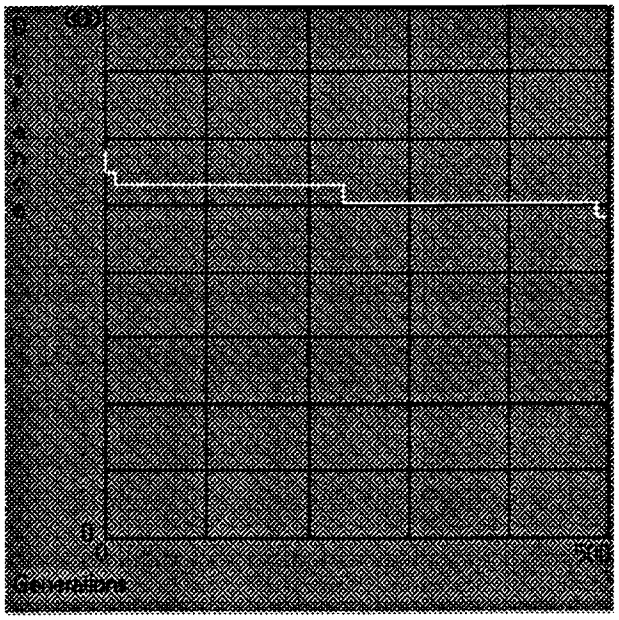

Figure 3 Progress of Best Path over Evolution

The genetic algorithm presented here only evaluated 37,500 paths which represents an incredibly small fraction of the total possibilities (1/ 3.24387 x 10^{12}). And the GA did so at an equally small fraction of the total time that would be required to enumerate all the possibilities. Yet, the best solutions it generated would appear by inspection to be very good and no improvements were apparent.

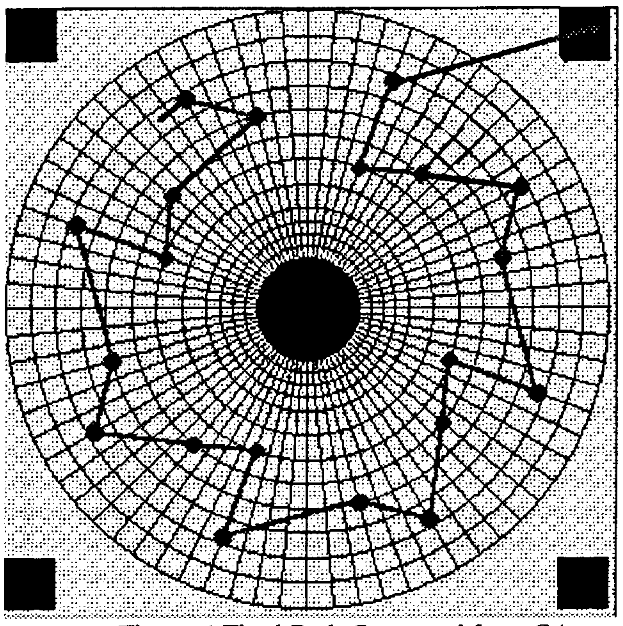

Figure 4 Final Path Generated from GA

The rules of the IEEE-SECON robot competition gave each team only 15 minutes to view the playing table to see where the balls were located before the competition began. The required computer time of 5 minutes was just about the right time since there was some time needed before the GA to enter the locations of the 20 balls into the program and there was time needed afterward to download the shortest path information to the robot.

Concluding Remarks

The genetic algorithm approach was chosen to seek out a superior collection path for the robot. This approach while it does not guarantee optimality, has been shown to quickly find good solutions to difficult combinatorial problems. Using the genetic algorithm approach, after only 5 minutes and examination of only 37,500 paths, excellent results were obtained for an efficient collection path.

GAs have already been shown to be useful for traveling salesman problems. The major contribution of this research is that a GA algorithm that has been shown to be practical for variations from the pure traveling salesman problems. In this case, the path problem had fixed starting and stopping locations, a non-complete round trip, and a pit in the playing board that must be avoided. The method was practical in the sense that a good solution was found in 5 minutes for a moderately sized problem.

References

1. Chatterjee, S., Carrera, C., and Lynch, L.A., 1996, Genetic algorithms and traveling salesman problems, *European Journal of Operational Research*, **93**, pp. 490-510.

2. Goldberg, D.E., 1990, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

3. Holland, J.M., 1975, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor.

4. Lawler, E.L., Lenstra, J.K., Rinnooy, K., and Shimoys, D.B., 1985, *The Traveling Salesman Problem*, John Wiley and Sons, New York.

5. Pál, K.F., 1993, Genetic algorithms for the traveling salesman problem based on a heuristic crossover operation, *Biological Cybernetics*, **69**, pp. 539-546.

WILLIAM A. RUSSELL

William Russell graduated from Southern Arkansas University in 1987 with a BS in Engineering Physics. In 1989, he earned his MS degree in Electronics and Instrumentation from the University of Arkansas at Little Rock and completed his doctorate in 1994. While working his way through school Dr. Russell has tested Sparrow Missiles for General Dynamics, and was the Engineering Manager for Scanning Technologies, Inc., which is a bar code industrial automation company. Before returning for his doctorate, William worked as an automation engineer for a startup company called ESI Automation. After his doctorate was completed, he continued his ultrasonic diagnostic equipment research in obstetrics for the University of Arkansas for Medical Sciences. Dr. Russell is now an assistant professor for the University of Southern Mississippi. He is continuing his obstetrical research which includes: precision low noise circuits, RF instrumentation, embedded microprocessors, digital signal processing, printed circuit board design and fabrication, and automation.

EYLER ROBERT COATES

Eyler Robert Coates graduated from Louisiana State University in 1979 with a Bachelor of Science degree in Industrial Engineering. He has worked at Cincinnati Milacron, Hatteras Yachts and Davis Yachts as an industrial engineer. He obtained a Master of Engineering Science degree from Louisiana State University in 1996. He has completed his doctoral coursework at LSU and plans to complete his dissertation in 1998. Eyler Coates is currently an assistant professor at the University of Southern Mississippi in Hattiesburg.