Autonomous Robot Path Planning using a Genetic Algorithm

Salvatore Candido
candido@uiuc.edu
Everitt Lab
1406 West Green Street
Urbana IL 61801

Department of Electrical and Computer Engineering
University of Illinois at Champaign-Urbana

Abstract

This paper discusses using a genetic algorithm, a search strategy based on models of evolution, to solve the problem of robotic path planning. This method provides a solid alternative to conventional methods of path planning. Aside from being efficient and robust, the optimization parameters for the desired path can be changed without changing the overall algorithm. Although this paper discusses the two dimensional mobile robot case the algorithm can easily be extended to three dimensions or a higher dimensional configuration space.

Introduction

One of the most difficult problems in building truly autonomous robotic systems is developing robust automatic motion planning. In order for an autonomous robot to be useful it must be able to efficiently and reliably plan a route between some starting point and a destination that does not cause the robot to collide with obstacles. Path planning algorithms attempt to connect these initial and final configurations by specifying a series of intermediate configurations through which the robot can safely traverse.

Many algorithms exist which attempt to solve this problem but all have shortcomings. The path planning problem is known to be PSPACE hard (Reif, 1979). This means that the complexity of the path planning problem increases exponentially with the dimension of the configuration space. The configuration space is the space of all complete specifications of the position of every point of a robot system (Choset et al., 2005). Many global path planning methods presuppose a complete representation of the configuration space. This is at best computationally expensive and often intractable. Potential field and bug approaches are local methods that do not make this assumption but are not complete methods. Local minima or loops will often cause this class of path planners to fail (Choset et al., 2005).

An alternate strategy which has not been greatly explored is the use of a genetic algorithm to plan a path. Genetic algorithms are search strategies based on models of evolution (Holland, 1975). They have been shown to be able to solve hard problems in tractable time. Also, an advantage of genetic algorithms is that one can change the optimization criteria for the path while not changing the overall algorithm.

Davidor (1990) has done work in this area and his paper, "Robot Programming with a Genetic Algorithm", specifically discusses path planning for a robot arm. This paper uses a similar path representation to that used by Hocaoğlu and Sanderson (1998) who also discuss path planning with a genetic algorithm. While these papers lay a foundation for the work done in this paper, neither hybridize the genetic algorithm with a local search scheme.

This paper discusses path planning in two dimensions for mobile point robot. This translates directly into a variety of systems such as a car or a novelty vacuum robot. Although this two dimensional case is in many ways the simplest case, it is also the most instructive. Working in two dimensions allows the researcher to simply view the path and see how using various operators, path representations, and parameters affect the efficiency and robustness of the algorithm's output. Also, since the configuration space is never explicitly computed, this method scales directly to three dimensional path spaces or higher dimension configuration spaces for use in multi-actuator robot arms and other general robotics systems.

Population and Path Representation

Several path representations were used over the course of this project. The first was a variable length, real coding. Each member of the population, a path, was represented by a series of two dimensional displacements. Beginning at an anchor point, the desired starting point for the robot,

a list of two dimensional displacements was specified. Summing the displacements produced a series of coordinates that acted as waypoints along the path.

Like most path planning algorithms, this project assumed the availability of a local planner. A local planner can take two points which are a short distance from one another and produce commands to move the robot between the two points. The most common assumption (and the one made in the paper) is the straight line planner which operates by moving in a straight line between the two points.

This initial path representation failed. In addition to avoiding obstacles and minimizing distance, the fitness function of the genetic algorithm had to have a term to draw path endpoints towards the goal. This worked impressively for environments which were sparsely populated with obstacles. However, in environments where a robot had to navigate through corridors this scheme did not always work. (This is explained further in the results section.) Making clever use of the parameters of the fitness function and lengths of the paths in the initial population it was possible to overcome these shortcomings. However, without previous knowledge of the robot's environment and significant tweaking of parameters for each environment this path representation scheme does not work robustly. This problem is a deceptive one when using this path representation.

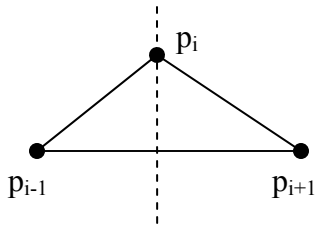


Figure 1

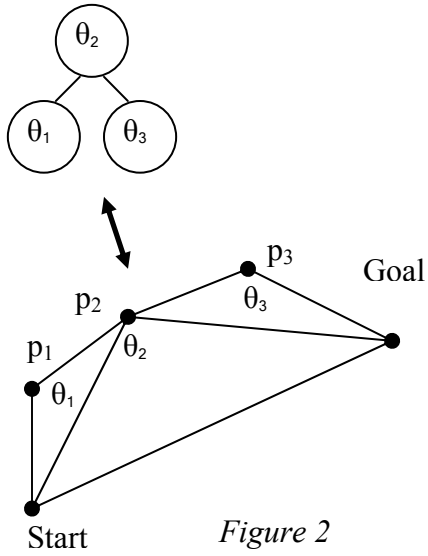


Figure 2

To fix these shortcomings a second path representation scheme was adopted. In this representation, each path is represented as an ordered set of angles. The angles are stored in a binary tree which can be decoded into a path. For the two dimensional case one needs only one angle per waypoint on the path for a unique representation of a path. Each node in the tree represents the angle that two path segments make at a specific point in the path, say p_i. Varying the value of the angle at the node corresponding to p_i we can move p_i to anywhere equidistant from the previous waypoint, p_{i-1}, and the next waypoint, p_{i+1} (Figure 1). The left child of that node then can modify the path segment between p_{i-1} and p_i in the same manner while the right child modifies the path segment between p_i and p_{i+1} (Figure 2). In this fashion, the path is uniquely defined in a recursive manner. By varying the height of the tree at specific nodes we can vary the resolution of the path at different places as necessary (Figure 3).

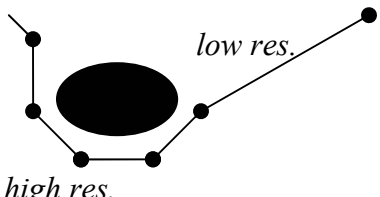


Figure 3

In the actual code, this binary tree was represented as an array of angles with the indices of the array corresponding to positions in the binary tree. The path waypoints were generated recursively and then sorted to build the actual path to test fitness and display. The tree was limited to six levels which is equivalent to sixty-four

degrees of freedom in each path. This number was chosen because it shortened run times yet allowed more resolution than necessary to find a path in the experimental environments.

By experimentation it was found that the best population sizes were between twenty and fifty paths. The results published later in the paper were generated with a population size of forty paths except for the ones hybridized with a local search. Those were generated with a population of only twenty paths.

Fitness Function

A good path, first and foremost, must avoid collisions with obstacles. Then, when considering two paths the better one is defined as the path where less distance traversed on route to the goal. Keeping these two rules in mind, a suitable fitness function can be expressed as (1).

$$f(P) = a_1 \cdot \delta(P) + a_2 \sum_{o_i \in O} C(P, o_i, \Delta) \tag{1}$$

The first term is simply a scaled version of the length of the entire path. In two dimensions, this can be calculated by summing the Euclidean norms of all the path segments comprising a path as shown in (2).

$$\delta(P) = \sum_{p \in P} \sqrt{(\Delta y)^2 + (\Delta x)^2} \tag{2}$$

The second term of (1) is a penalty function to assess against paths intersecting obstacles. It is computed by counting the number of collisions between the path and all obstacles, in the robot's workspace checked at some resolution, Δ, along the path. By experimentation it was found that a ratio, a_2/a_1 of about 10 was best with $\Delta = 0.1$.

This fitness can easily be extended to penalize paths for traversing certain terrain or modified to push paths away from obstacles to maximize clearance. As mentioned previously, a genetic algorithm method can easily change the search criteria without modifying the entire planning algorithm.

Operators

Several operators were used to search for and select candidate paths. Simple cross, random mutation, and the selection operators are components of the simple genetic algorithm. The other operators are specialized operators which have a specific purpose for this application.

Simple Cross – Simple cross is a simple one point crossover between two of the arrays of angles representing each path.

Subtree Cross – Subtree cross is also a one point crossover. A node is chosen in each binary tree and the nodes as well as their children are swapped between the two trees. Another variant of this operator swaps the two subtrees in the same location in both trees. This crossover is

better than the simple cross. Each subtree, not consecutive values in the array holding the tree, represents a segment of the path. Using the subtree cross with the same cross site in both trees is equivalent to exchanging a portion of the two paths.

Random Mutation – This mutation randomly selects one node in the binary tree and replaces its value with a random angle.

Small Mutation – This mutation randomly selects one node and perturbs the angle contained there a small amount. This operator is performed more and more frequently as the number of generations increase and serves to fine tune an otherwise good path.

Flip Mutation – This mutation randomly selects one node and makes a convex segment of the path concave or vice versa. This mutation can be used to target path segments that intersect with obstacles and perturb them into free space (Figure 4).

Proportionate Selection – This operator propagates paths from one generation to the next giving each path a number of copies proportionate to its fitness compared to other paths.

Tournament Selection – This operator propagates paths from one generation to the next by selecting the best path of from a number of randomly selected paths.

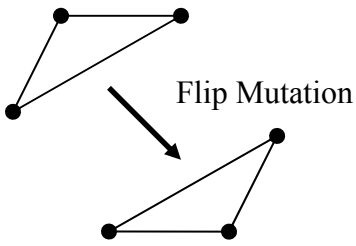


Figure 4

Although all operators were implemented during the course of this project, only a subset was used to obtain the results published later in the paper. As previously stated, subtree cross is superior to simple cross and thus simple cross was not used. Fifteen crosses were done on the population per generation and selection was done with replacement. All three mutation operators were used. Random mutation had a probability of .05 which decreased as the number of generations increased. Small mutation and flip mutation started with a probability of .025 but were applied with increasing probability as the number of generations increased. This changing mutation rate was adopted to facilitate diversity initially but then fine tuning later on. Lastly tournament selection with a tournament size of two was chosen over proportionate selection.

Hybridization

Using the system already described the algorithm already performs well. However, it seems fairly obvious from the results shown below that better paths are available. The genetic algorithm using mutation will eventually find optimal paths but once the population converges to a solution it will take a long time to move to that optimal path through mutation alone.

Using local search techniques a genetically generated path can quickly be turned into a close to optimal path. To shorten path length, after generating a path from the genetic algorithm a path straightening mechanism was employed. When applied to a path the straightening mechanism removed waypoints in a deterministic manner testing to see if the perturbation produced a more

fit path. The enhanced path was used to evaluate fitness but did not replace the original path in the population, a Baldwinian scheme.

After experimentation it was found that the genetic algorithm did not perform as well enhancing all paths. This is because many similar paths are identical after applying the local search. Since there is no longer a fitness distinction between two similar paths, the genetic algorithm cannot choose which path is better to propagate to the next generation and continue the search with.

To fix this problem a scheme was chosen where local search was only applied to paths not intersecting obstacles. Essentially, the genetic algorithm was used to find a feasible path. Once a good path is available to work with, local search is used to quickly build a close to optimal solution. This arrangement drastically improved run times and the quality of paths generated.

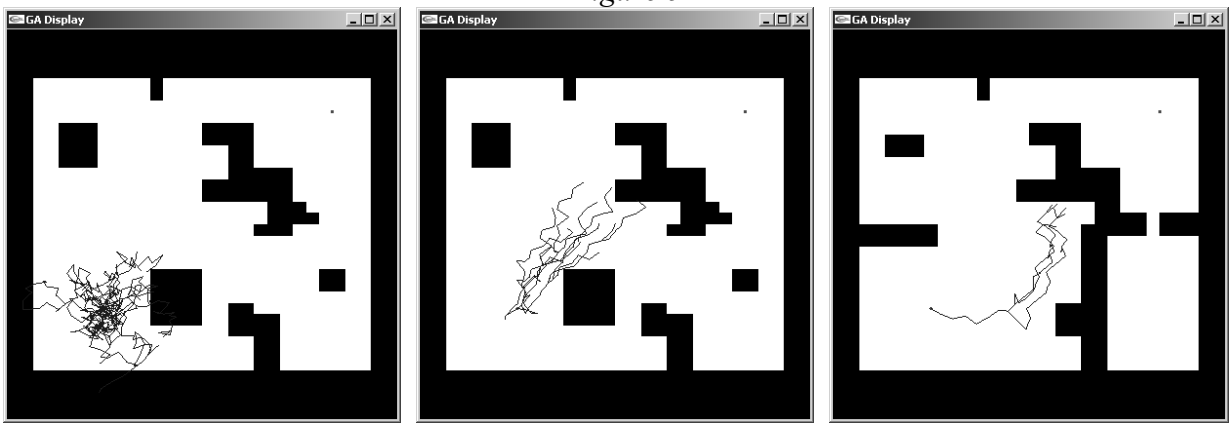
Implementation

In order to gain maximum flexibility, this algorithm was coded from scratch using Microsoft Visual Studio.NET. C++ and the standard template library (STL) provided a good framework to build objects that contained paths and operators modifying them. OpenGL was used to display the algorithm at work and its results in a separate processor thread from the genetic algorithm code.

Results

The first path representation produced excellent results in easy path planning problems. However, because the fitness function relied on the distance to the goal as a search criterion, paths could become stuck in local minima in the search space. Paths in the population would grow until they encountered an obstacle. Unable to go through it, if paths could not shorten their distance at every step of growing around the obstacle the population would stall and converge. Figure 5(c) shows the initial population for one run on the genetic algorithm. Figures 5(b) and 5(c) show two instances where the population was "trapped" in a local minimum.

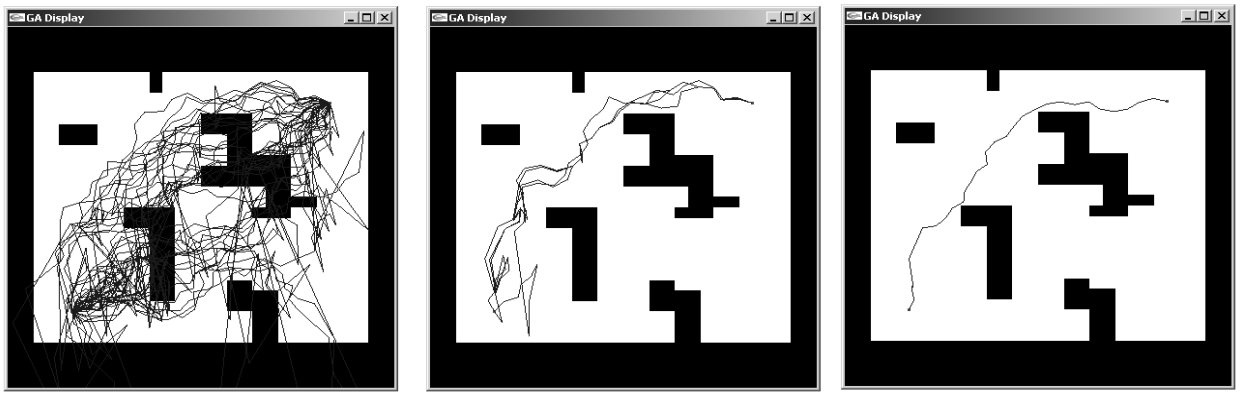
Figure 5



(a) Generation 0 *(b) Trapped* *(c) Trapped*

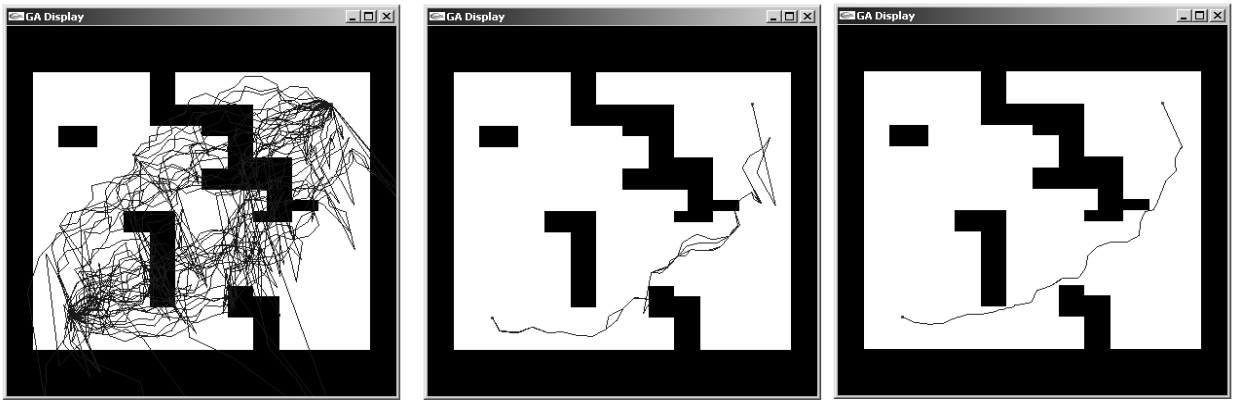
With the new path representation scheme the local minimum problem was no longer an issue. Figures 6, 7, and 8 display three runs of the algorithm in three different environments. Notice that the genetic algorithm was quick to find a feasible solution but it took many generations for the population to converge to a path one would be likely to accept as a good solution.

Figure 6



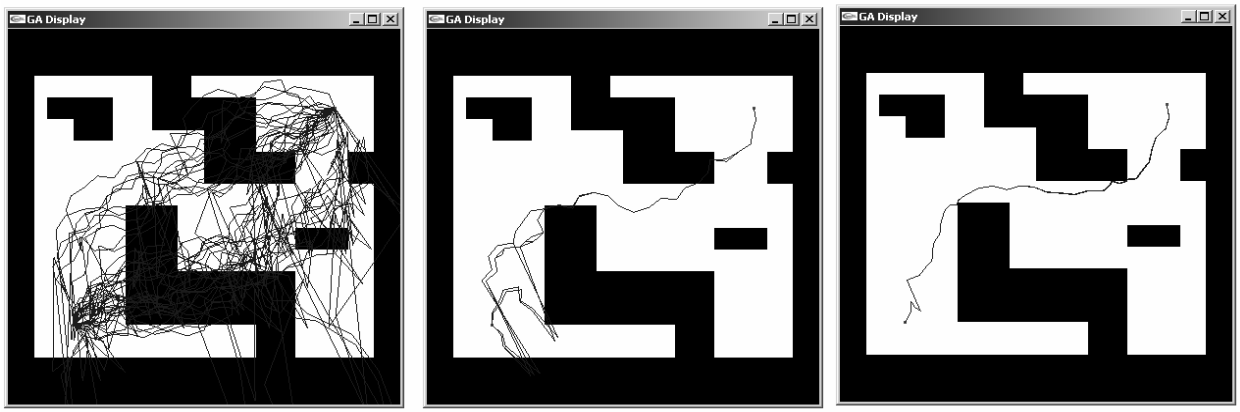
(a) Generation 0 *(b) Generation 23* *(c) Generation 120*

Figure 7



(a) Generation 0 *(b) Generation 25* *(c) Generation 204*

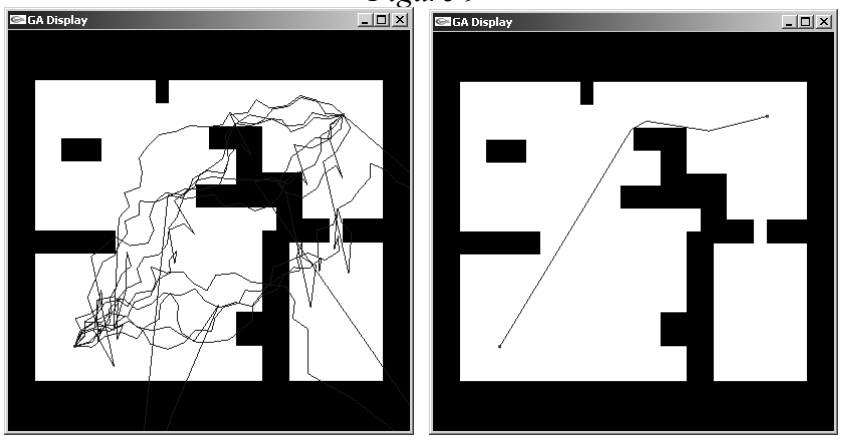
Figure 8



(a) Generation 0 *(b) Generation 42* *(c) Generation 185*

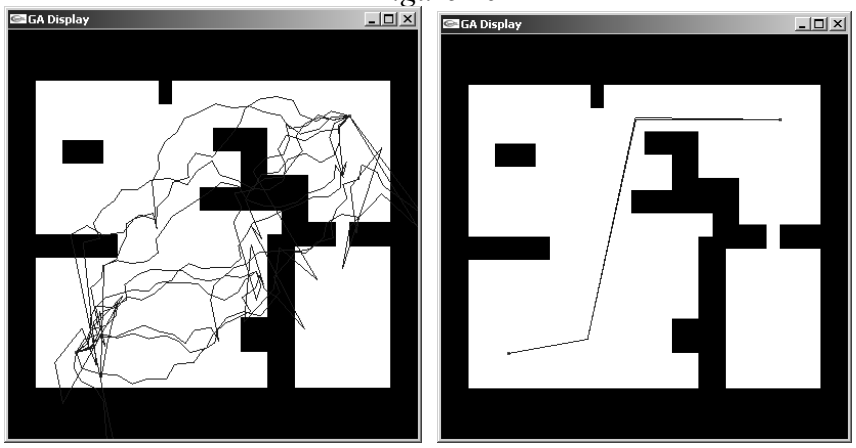
Once local search was introduced, the genetic algorithm simply had to find a feasible path and within a few generations the population converged to a close to optimal path. Figures 9 and 10 display the results of two runs of the algorithm hybridized with the application of local search on the paths.

Figure 9



(a) Generation 0 *(b) Generation 20*

Figure 10



(a) Generation 0 *(b) Generation 43*

All the results shown above were typical of most runs with the same parameters and operators. Occasionally the population converged to an unfeasible solution but this was only a small fraction of runs. The results of these simulations indicate that this is a reasonable method for path planning.

Although this work was done in two dimensions this algorithm could easily be extended to higher dimension spaces. Also, the hybridization scheme could be improved to return optimal paths rather than close to optimal paths.

Conclusions

This paper has discussed using a genetic algorithm to solve the problem of robotic path planning. With the use of the correct path representation, fitness function, genetic operators, and parameters for the genetic algorithm it is possible to generate feasible paths in a small number of generations. Using a hybridization scheme which combines the genetic algorithm with a local search scheme on feasible paths we can dramatically reduce the time to find good paths while increasing the quality of generated paths. The simulation results demonstrate that the algorithm works well in a variety of environments. More needs to be done before this system can be applied to a real robotics system. However, this work shows encouraging results for future efforts.

Acknowledgments

This work was done as a semester project for GE 531 at the University of Illinois at Champaign-Urbana during the fall semester of 2005. Thanks to the University of Illinois, the Department of General Engineering, and Professor David E. Goldberg for providing the means and forum to complete this work.

Sources

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., et al. (2005). *Principles of robot motion: theory, algorithms, and implementations*. Boston: MIT Press.

Davidor, Yuval. (1990). Robot programming with a genetic algorithm. *Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*, 186-191.

Hocaoğlu, Cem & Sanderson, Arthur C. (1998). Multi-dimensional path planning using evolutionary computation. *Proceedings of the IEEE Conference on Evolutionary Computation*, 165-170.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

Reif, J.H. (1979). Complexity of the mover's problem and generalizations. *Proceedings of the IEEE Transactions on Robotics and Automation*, 421-427.