Genetic algorithms in reactive control of robot navigation

Siddharth Shankar Jha
Roll No: 14EE30022
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur 721302, India
Email: siddharthjha@outlook.com

Starting Note (By Siddharth): I really admire Dr. Arkin and his work and have read many of his other papers related to roboethics. I was fortunate to find this one and had a great time writing this term paper.

Abstract

In this term paper, we explore the application of genetic algorithms to learning of navigation behaviours by mobile robots. It is mostly based on a research at GeorgiaTech. It evolves reactive control systems in certain environments, thus creating a set of "ecological niches" that can be used in similar environments. Using genetic algorithms as an unsupervised learning technique reduces the inputs required to design and optimize a navigation system. An introduction to behavioural, reactive control and genetic algorithms respectively is given in the next two sections. An in-depth analysis of the methodology is done in the following section. The next few sections deal with the simulation and the parameters of the genetic algorithm developed. Finally, the graphs and figures related to the simulation results are presented along with suitable discussions and conclusions.

Introduction to Behavioural and Reactive Control

Reactive control, especially reactive control in robotics has been an research interest for a few decades now. It was developed due to an increasing dissatisfaction and discontent with existing traditional robot control architectures. It draws from the behaviourist school of psychology, based on the idea that there should not be an existing symbolic representations of the outside world at the beginning. For a robot, put in simpler words, for example it'd require absolutely no programming of representations of what an obstacle would look like, or what kind of surface it's moving on. Instead all the information is obtained and learned gradually from the input of the it's own sensors. The robot uses that information to gradually correct its actions according to the changes in immediate environment. The robot behaviours are closely tied to the effectors that carry out the behaviour of the robot. The motor schema approach to reactive control has been proven to be quite effective in the field of robotics. It allows researchers to create robots that can function robustly in a dynamic world.

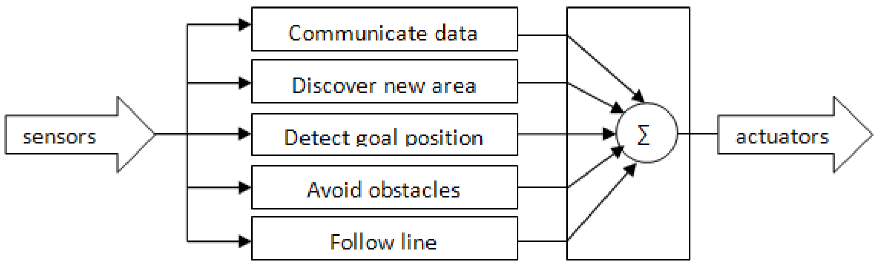


Figure 1: Behaviour Based Robotics

It also enables addition of a high level planner that can configure and call these behaviour functions. But, while this method does help in overcoming the shortcomings of a purely reactive based approach that might not be incorporate such a planner, it still is a tough job to identify the parameters for the controller. Much effort is required to identify the parameters for a given environment setting.

The design of a reactive control problem has 2 parts: a structure and a set of control values. The structure is determined by the tasks that the robot must perform, since this constrains the collection of behaviours that the robot can exhibit. A simple robot that just has to avoid a stimulus may have 1 or 2 behaviours while complex autonomous robots like self driving cars may require several such behaviours, like goal seeking and exploration. Once the structure of the system has been clearly defined, the system is tuned by manually adjusting the parameters that control the behaviours. Because a single parameter setting can affect a particular behaviour, its relation to other behaviours, and other emergent behaviours, it is really difficult to fine tune schema parameters manually.

For an autonomous robot, previous research at GeorgiaTech (by Prof. Arkin himself) has led to development of several such schemas. Some of them are:

- **Avoid-static-obstacle:** move away from a non-threatening impediment to motion.
- **Move-to-goal:** move towards an attractor.
- **Move-ahead:** move in a pre-specified compass direction.
- **Stay-on-path:** find a path in the environment and stay near its center.
- **Noise:** move in a random direction, useful for both exploration and handling problems with local maxima.
- **Docking:** move in a ballistic then controlled motion towards a docking workstation.
- **Probe:** move toward the most open space.
- **Avoid-past:** avoid areas that have been visited recently.
- **Escape and dodge:** avoid moving or potentially aggressive obstacles.

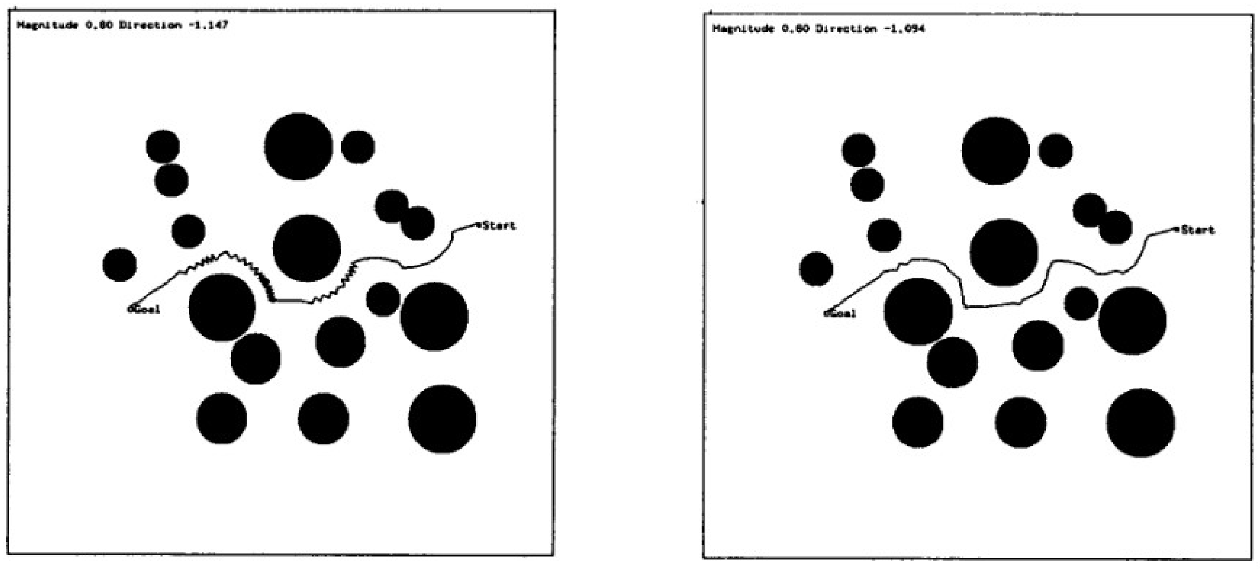

Magnitude 0.80 Direction -1.147
Start
Goal
Magnitude 0.80 Direction -1.094
Start
Goal

Figure 2: Comparison of paths planned without and with avoid-past schema active

For this term paper, we use three schemas namely **avoid-static-obstacle, noise** and **move-to-goal.** Genetic algorithm is used to determine the schema parameters of all the three.

We follow a scheme of unsupervised learning in this case. Thus, the robot is trained in multiple environments to account for the lack of training data. The navigation system must evaluate its own plans and learn via a reward/punishment system. The model GA-Robot presented in the paper optimizes the reactive control system, in a way that it focuses on optimizing the individual reactive behaviours themselves.

The genetic algorithm technique differs from other learning techniques in such a way that populations of robots learn, not an individual. Each robot is given a fixed set of parameters that control its behaviour. Then the robots are run in a simulation of the environment and the performance is evaluated. New parameters are generated and given to another generation. After many generations have been simulated, good parameter sets emerge. Thus evolution is said to occur, and the learning is said to occur on an evolutionary time scale.

Introduction to Genetic Algorithms

The GA is a hill climbing search method that finds optimal solutions by subjecting a population of points in a search space to a set of biologically inspired operations. The “fitness” of each member of GA population is computed by an evaluation function that measures how well the individual performs in the task domain. Best members are propagated according to fitness, while numbers of poor individuals are reduced. It may find optimal solution but is not guaranteed to do so.

Each iteration of the GA begins by decoding the bit-string into search-space points and using the search function to evaluate the fitness of the individual. If a minima is sought, the individuals which return lower search-function values will be assigned higher fitnesses. Once the population has been evaluated, a set of genetic operators is applied. Several genetic operators have been proposed, but the three most frequently used are reproduction, crossover, and mutation as shown in Figure 3.

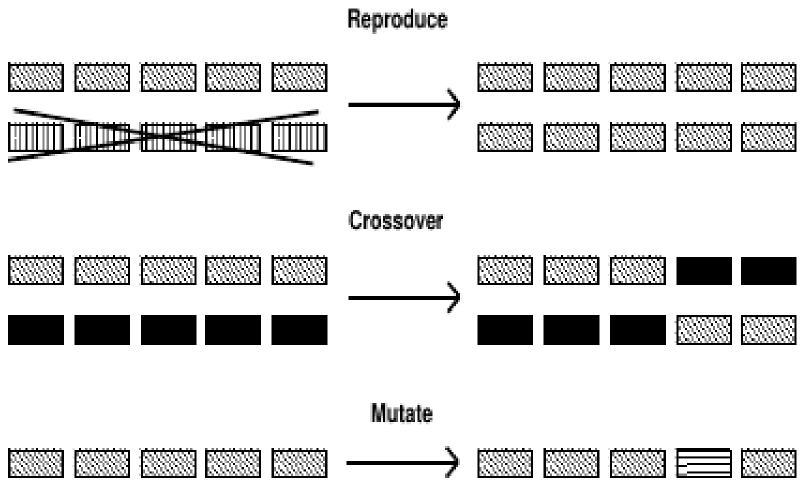


Figure 3: Three Basic Genetic Algorithm Operators

The reproduction operator selects the good strings in population and forms a mating pool. It is also known as Selection Operator. The Essential idea is that above –average strings must be picked from current population and their multiple copies must inserted in the mating pool in a probabilistic manner. A concept of roulette wheel is used to select the strings, with higher fitness strings having higher probability of being picked, akin to how a wheel with unequal divisions would work in a roulette.

During crossover, new strings are created by exchanging information among strings of the mating pool. The most common approach is that two strings are picked from the mating pool at random and some portions of the strings are exchanged between the strings. This creates a pair of new individuals that may or may not perform better than the parents. For example, if the string [00000000] was crossed with string [11111111] the result might be [00011111] and [11100000] etc. The choice of which individuals to cross and where to cut the chromosome is quite random random. This random-search and probabilistic component gives GAs much of their power of convergence.

The mutation operator is used to prevent loss of info that occurs as the population tends to converge on the fittest individuals. It is used to avoid local maxima and minima during optimization problems. Premature convergence is said to occur when the population cannot improve because all of the individuals in the population have the same value for a given gene in a chromosome. Since no amount of selection or exchanging of the same value will change it, only mutation would allow lost information to be recovered, and further, also maintains variety during convergence.

The GA-Robot methodology

The navigational performance of the robot is used as a performance metric. The simulation world is composed of a 2 dimensional space with multiple obstacles and a single goal. The task of robot in each simulation is to avoid obstacles and reach the goal. Robots are awarded inversely to travel time, distance covered and number of collisions.

The clutter of the environment in this paper is given by the percentage of area covered by obstacles in a graph. It can be 1%, 10% or 25% for the simulations mentioned. There are also three types of simulations, fixed, varying or general as demonstrated in Table 1.

World Type	*Obstacle Positions*	*Percentage Clutter*
Fixed	fixed	fixed
Varying	random	fixed
General	random	random

Table 1: Simulation characteristics

The motion of the robots with 3 schemas (noise, move-to-goal, avoid-static-obstacle) are described the following five parameters that need to be identified and optimized using genetic algorithm:

- **Goal gain:** strength with which robot approaches the goal.
- **Obstacle gain:** strength with which robot moves away from the obstacles.
- **Obstacle sphere-of-influence:** distance from obstacle at which robot is repelled.

- **Noise gain:** amplitude of random wandering.
- **Noise persistence:** number of time steps the noise vector is held constant.

An array containing values for these five parameters forms the genome used by the genetic algorithm. The parameters control the direction and speed at which the robots move.

Genetic Algorithm Methodology

Although traditional GA strategies require that the genome should be encoded as bits because traditional crossover and mutation operations requires bitwise exchange of data, but this is not a hard and fast requirement as there's only exchange of data of some sort. In this paper though, we use floating point list representation of chromosome for the genetic algorithm. The list is composed of robot control parameters.

The algorithm starts by generating a population of robots, and also generates a random environment containing obstacles and one goal. Each of the robots is initialized with a set of random values for the schema parameters. The algorithm on each generation is run thrice, until the robot reaches the goal or exceeds a maximum number of steps. A running total of the raw values of fitnesses for each robot is kept maintained during all the simulations, and used during the application of the genetic operators. A pseudo-code of the algorithm is presented in Figure 4.

```
begin
  /* Make a new environment */
  Obstacles.Create;
  /* Make a new population */
  Population.Build;

  for 1 to NUMBER_GENERATIONS do
  begin

    for 1 to RUNS_PER_GENERATION do
    begin
      /* Let Robots try to reach goal */
      for 1 to MAX_NUMBER_STEPS do
      begin
        Robots.Move;
      end

      /* Update environment */
      Obstacles.Recreate;
    end

    /* Prepare next generation */
    Robots.Reproduce;
    Robots.Crossover;
    Robots.Mutate;
  end
end
```

Figure 4: Pseudo Code of the genetic algorithm

Three parameters were measured: no of steps taken, distance covered and time taken. We scale the fitness function and the genetic operators maximize the scaled fitness function. Our aim however is to minimize penalties. If distance is heavily penalized, robot will take small steps and we don't exactly want that. Thus an additional penalty on not reaching goal is added.

As the chromosomes are floating point, a modified crossover operator is designed for the research. We calculate average and difference of the two genes at crossover point. The new gene value is given by

*new_gene = average + difference * randomnum*

where the randomnum is between -1 to +1.

Similarly mutation operator is modified for floating point representation.

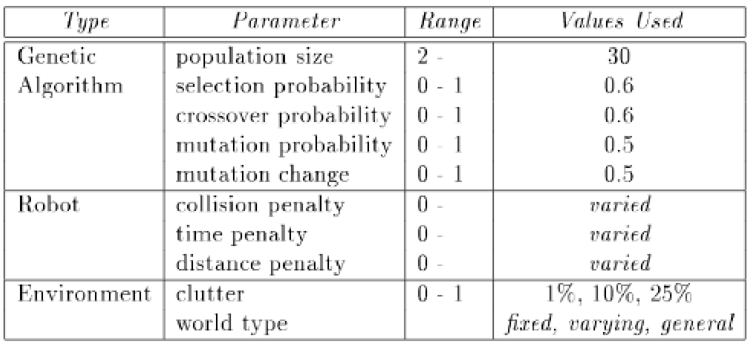
*new_gene = old_gene + old_gene*randomnum*mutation_delta*

which unlike standard mutation is somewhat dependent on the old value.

Factors Affecting Learning

There are several factors affecting the learning rate. Few of them are:

- **Genetic Algorithm Control Parameters:** The mutation probability is kept quite high in the work as the modified floating point mutation does not completely alter the old value. The probability was found optimal between 0.5 and 1.0. It helps the algorithm find deeper minima and also doesn't take random walks.
- **Robot Type:** Three types of robots are defined: safe, direct and fast. The penalties are different for each of the types. For example, safe robots have high collision penalty.
- **Environment Type:** The density of obstacles significantly changes the way robot learns, how frequently do path changes have to occur and the deviation from a straight line. Many parameters are selected for separate niches and a couple of planners select which pre-trained parameters are applied during a dynamic run with changing obstacle density and robot type.



Type	*Parameter*	*Range*	*Values Used*
Genetic Algorithm	population size	2 -	30
	selection probability	0 - 1	0.6
	crossover probability	0 - 1	0.6
	mutation probability	0 - 1	0.5
	mutation change	0 - 1	0.5
Robot	collision penalty	0 -	*varied*
	time penalty	0 -	*varied*
	distance penalty	0 -	*varied*
Environment	clutter	0 - 1	1%, 10%, 25%
	world type		*fixed, varying, general*

Simulation Results

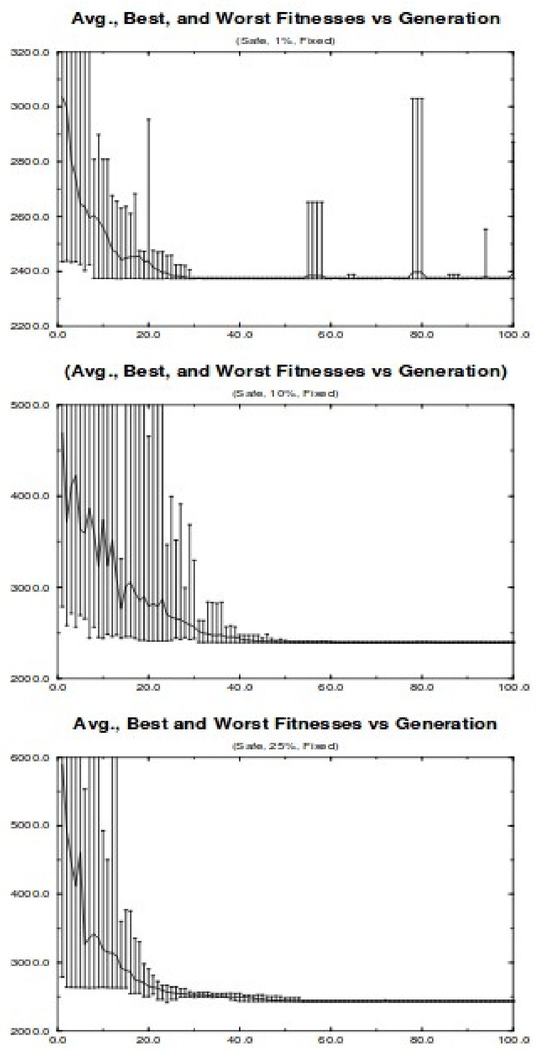


Figure 5: Convergence against different clutter

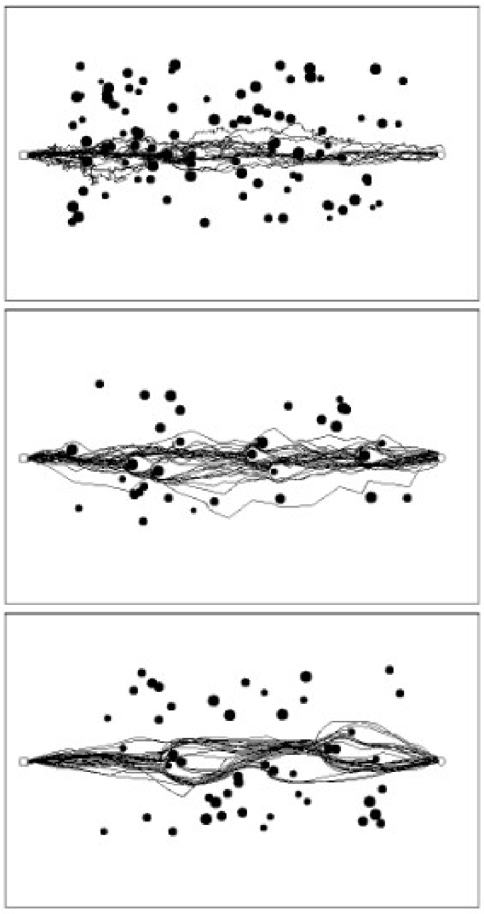


Figure 6: Beginner, Intermediate and Final generations path planning

Summary

A novel genetic algorithm based method to optimize robotic control parameters in a schema based navigation system was proposed. The system was fully implemented in the research, and has been evaluated extensively as well. Simulation results are shown for 3 different robot types in 3 different environments and parameters are evaluated finally.
The floating point representation of chromosomes is novel and allows for high mutation probability which maximizes chance of better convergence.

References

- **Using Genetic Algorithms to Learn Reactive Control parameters for autonomous robotic navigation** by A. Ram, R. Arkin, G. Boone and M. Pearce
- **Avoiding the past: A simple but effective strategy for reactive navigation** by T. Balch and R. Arkin
- **AuRA: Principles and practice in review** by R. Arkin and T. Balch