Using Genetic Algorithms to Learn Reactive Control Parameters for Autonomous Robotic Navigation*

Ashwin Ram, **Ronald Arkin**, **Gary Boone**, and **Michael Pearce**

College of Computing
Georgia Institute of Technology

Abstract

This paper explores the application of genetic algorithms to the learning of local robot navigation behaviors for reactive control systems. Our approach evolves reactive control systems in various environments, thus creating sets of "ecological niches" that can be used in similar environments. The use of genetic algorithms as an unsupervised learning method for a reactive control architecture greatly reduces the effort required to configure a navigation system. Unlike standard genetic algorithms, our method uses a floating point gene representation. The system is fully implemented and has been evaluated through extensive computer simulations of robot navigation through various types of environments.

* *Adaptive Behavior*, volume 2, issue 3, pages 277–304, 1994.

1 Introduction

Navigation through a cluttered environment to a specified destination without hitting obstacles is a common, but complex and underconstrained robotic task. Apart from the computational constraints on the navigation system, the system must be robust enough to navigate through a large number of possible environment configurations.

Traditional robotics research has focused on symbolic representations and world modeling to solve navigation problems (Albus, McCain, & Lumia, 1987; Elfes, 1987). A large part of the work that these systems perform concentrates on the mapping of sensor data to a high-level symbolic representation of the environment. Such systems combine data from different types of sensors, process this data to keep their internal world models up to date, and perform path planning from this world model. While these systems perform well in constrained environments, their performance is less robust in dynamic, realistic environments. Symbolic robotic navigation systems rarely meet real-time constraints in tasks that seem trivial to humans, and do so only under highly constrained and closely supervised conditions.

An alternative approach to robot navigation is reactive control, which ties sensor data directly to the actions of the robot. The behaviors of these robot systems emerge from the careful organization of these reactions. The behaviors thus operate without the use of a world model, and are computationally less demanding than those of a symbol processing robot navigator. These architectures can act in real time, and are more robust in dynamic environments than their symbolic architectural counterparts (Anderson & Donath, 1991; Arkin, 1989; Brooks, 1989b; Payton, Keirsey, Krozel, & Rosenblatt, 1992). Developing a reactive system thus requires the selection and structuring of the control parameters that underlie the behaviors of the robot. While simpler than modelling a complex and dynamic environment, selecting parameters to control robot behaviors can be difficult.

This paper describes the application of genetic algorithms to the problem of optimizing robot navigation control parameters in a reactive control system. Genetic algorithms provide an unsupervised learning method that greatly reduces the effort required by the designer to configure a navigation system. Furthermore, since the genetic algorithms are run as simulations on a computer and do not require that the learning occur on the actual robotic system, they greatly decrease the amount of time required to present the system with a sufficient number of learning trials.[1]. Our approach is to train a reactive control system in various types of environments, thus creating a set of "ecological niches" that can be used in similar environments that were not presented in the learning phase. Further, by variously weighting the different costs of the robot navigation, we can create robots optimized for safety, speed, or distance.

[1]From a robotics perspective, simulation results must be viewed with some skepticism as the transition to real robotic hardware is often a difficult one. The simulation used in this research is based upon the simulators we have used in the past in porting our research to our Denning robots. A good correlation between simulation and robotic performance has been observed in these cases (e.g., (Arkin, 1989; Arkin & Murphy, 1990).)

2 Robotics

2.1 Reactive Control for Robot Navigation

The reactive control approach to robotic navigation grew out of a dissatisfaction with traditional robotics architectures. Reactive control draws from the Behaviorist school of psychology, in that there are no explicit symbolic representations of the external world. The subsumption architecture (Brooks, 1989a), which typifies the purely reactive control model, is composed of simple behaviors (like wandering, obstacle avoidance, and goal following) that combine to produce emergent behaviors that were not explicitly designed to be exhibited by the system. The simple behaviors of a reactive control system acquire the information about the environment directly from the sensors, instead of through an intervening world model. These behaviors are closely tied to the effectors that carry out the behavior of the robot (Kaelbling, 1986; Payton, 1986).

These non-representational systems avoid many of the pitfalls experienced by the traditional symbolic and world-model driven systems, but at some cost. The subsumption architecture does not allow explicit representations of high-level goals, so it is difficult to reconfigure the systems for different tasks or for reasoning about unperceived objects. Further, as behaviors are added to the system and its complexity increases, the interaction of the various behaviors often becomes difficult to predict and debug. Robust individual behaviors must first be designed and implemented, then tuned to fit the response characteristics of the sensors and effectors.

The motor schema approach to reactive control has proven to be a powerful method in the field of robotics (Arkin, 1989). This model of robotic systems allows researchers to construct robots that can function robustly and in real time in a dynamic, open world. Specifically, this method enables the integration of a high-level planner to configure and instantiate these behaviors, thereby introducing more flexibility than provided by the purely reactive approach. A more detailed description of the motor schema method is given in the following section. However, while this approach overcomes the limits of the purely reactive paradigm, the problem of parameter tuning remains. Much effort is often required on the part of the user to determine the proper parameter settings for a given type of environment.

2.2 Schema-Based Reactive Control

The design of a reactive control architecture has two 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 behaviors that the robot can exhibit. Simple robots that are designed to avoid predators need few behaviors, while more complex robots may also have goal seeking and exploratory behaviors. Once the structure of the system has been defined, the system is tuned by adjusting the parameters that control the behaviors. Because a single parameter setting can affect a particular behavior, its relation to other behaviors, and emergent behaviors, it is difficult to fine tune schema parameters manually.

In the Autonomous Robot Architecture (AuRA), motor schemas provide the reactive component of navigation (Arkin, Riseman, & Hanson, 1987). Instead of planning by predetermining an exact route through the world and then trying to coerce the robot to follow it, motor schemas (behaviors) are selected and instantiated in a way that enables the robot to interact successfully with unexpected events while striving to satisfy its higher level goals. Motor schemas are manifested as analogs of potential fields (Arkin, 1989). Multiple active schemas are typically present, each producing a velocity vector driving the robot in response to its perceptual stimulus. The individual vectors are summed and normalized, yielding a single combined velocity for the robot. These vectors are continually updated as new perceptual information arrives, resulting in immediate response to new sensory data.

Some of the schemas we have already developed include:

- **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.
- Various **maintain-altitude**, **move-up**, and **move-down** schemas useful for navigation in rough terrain.
- **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.

These schemas have been developed for navigating dynamic environments, although the schema-based approach could be applied to other robot behaviors and higher level tasks, such as "survive" or "deliver-mail." Other work in our lab has used schemas for manipulator positioning.

For this paper, we used three schemas, **move-to-goal**, **avoid-static-obstacle**, and **noise**. The schema parameters controlling the behavior of these schemas were determined autonomously using a genetic algorithm, as discussed below. Our results show that the method can be used to tune schema-based reactive control systems by learning parameter settings that optimize performance metrics of interest in various kinds of environments.

2.3 Robot Learning

There are several factors to be considered in designing a robot navigation system that learns. To ensure adequate generalization of a given environment, many trial runs are required during training. Due to the time and wear costs for both robot and teacher, it is impractical to have a human instruct the robot during training. This problem is compounded by training the robot for multiple environments. Therefore, unsupervised learning is required. Further, because a goal is reached or an obstacle hit through the combination of many simple actions, it is impossible to choose a particular behavior that led to either good or poor performance. It is therefore difficult to assign credit and blame in navigation. The navigation system must evaluate its own plans and learn via a cost/benefit analysis based on easily measurable characteristics of the system. For example, the time of travel of a robot from start to goal can be easily and objectively measured and used by the training system.

Although learning is an important feature of intelligent and autonomous robot systems, work beyond the conceptual stage is limited. Fikes, Hart, and Nilsson extended the STRIPS robot navigation system to allow it to learn from its failures (Fikes, Hart, & Nilsson, 1972). Barto, Anderson, and Sutton attempted to solve nonlinear robot navigation tasks using a two-layer connectionist network (Barto, Anderson, & Sutton, 1982). This simulation allowed the robot to learn associations between landmark and the directions of travel that would lead it to the goal, which would provide positive reinforcement. Previous workers have also applied genetic algorithms to robot navigation. Dorigo and Schnepf used this method to train simulated robots to avoid obstacles and follow moving targets (Dorigo & Schnepf, 1991). The genetic algorithm was used to determine when the robot should switch from one behavior to another, as only one behavior is active at a time. Thus the grain size of the learning is at a fairly high level; the robots could not learn how to optimize their individual behaviors. Grefenstette, Ramsey, and Schultz's SAMUEL system takes a different approach; rather than optimize individual behaviors ("decision rules"), a genetic algorithm is used at the level of tactical plans comprising an entire set of decision rules for a given task (Grefenstette, Ramsey, & Schultz, 1990).

Our method, GA-Robot, also uses genetic algorithms to optimize a reactive control system but, unlike the above methods, it focuses on optimizing the individual reactive behaviors themselves. An interesting direction for future research would be to combine our technique for optimizing robot behaviors with other methods for selecting and composing behaviors (e.g., (Dorigo & Schnepf, 1991; Grefenstette, Ramsey, & Schultz, 1990; Ram, Arkin, Moorman, & Clark, 1992)), as well as with other kinds of learning algorithms such as operationalization, case-based reasoning, or reinforcement learning (e.g., (Gordon & Subramanian, 1993; Ram & Santamaria, 1993)).

The genetic algorithm approach differs from other learning techniques in that populations of robots learn, not individuals. Each robot is given a fixed set of parameters which control its behavior. The robots are then run through the simulated environment and their performances evaluated. New parameters are generated and given to another generation. Only after many generations have been simulated will good parameter sets emerge. Thus, the learning is said to occur on an evolutionary time scale.

Our method uses a floating point representation, which has been tried before with success (Davis, 1991; Janikow & Michalewicz, 1991). Janikow and Michalewicz used a standard crossover operator, except that it crossed *between* the genes, leaving them unmodified. Davis proposed a crossover operator that averaged the parent genes at *every* position to produce a new offspring genome. Our method defines crossover as point-exchanges rather than substring exchanges. At a crossover point, the new value is the average of the parent values plus or minus part of their difference. Thus, the offspring values can grow beyond the range of the parents while remaining closely related.

3 Genetic Algorithms

A genetic algorithm (GA) is a hill-climbing search method that finds near-optimal solutions by subjecting a population of points in a search space to a set of biologically-inspired operators (Goldberg, 1989). The "fitness" of each member of the GA population is computed by an evaluation function that measures how well the individual performs in the task domain. The best members of the population are propagated proportionately to their fitnesses, while the numbers of poorly-performing individuals are reduced or eliminated completely. By also exchanging information between individuals to create new search points, the population explores the search space and converges to the neighborhood of the optimal solution to the problem. The algorithm may find the optimal solution, but is not guaranteed to do so.

Genetic algorithms apply their operators to a representation of the search-space points chosen to facilitate the genetic operators. In a traditional GA, the representation is a position-dependent bit string, where each bit is a "gene" in the string "chromosome" (Goldberg, 1989). The choice of bit strings allows chromosomes to be conveniently cut into substrings, enabling the exchange of information between individuals.

Typically, 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*. These operators are expressed graphically in Figure 1. Note that each of the rectangles in the figure represents a single bit of the string (in practice, most representations use much longer strings).

The reproduction operator selects the fittest individuals and copies them exactly, replacing less-fit individuals so the population size remains constant. This increases the ratio of good individuals to the number of poorly-performing ones. The selection process uses a weighted roulette wheel; the best individuals are preferred, but not guaranteed, to be reproduced.

The crossover operator allows two individuals to exchange information by swapping some part of their representations. 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]. The choice of which individuals to cross and where to cut the chromosome is random. This random search component gives GAs much of their power (Goldberg, 1989).

The mutation operator is used to prevent the loss of information that occurs as the population converges on the fittest individuals. *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. Since no amount of selection or exchanging of the same value will change it, mutation allows lost information to be recovered, and further, maintains variety during convergence.

A GA can be thought of as a search method that tries to maintain a balance between exploiting points in the search space that have already been reached and exploring other points that are yet to be tried. The reproduction operator exploits the knowledge present in the population by increasing the numbers of fitter individuals. The crossover operator explores the search space by producing new points to evaluate. This simultaneous exploration and exploitation moves the algorithm toward populations containing the fittest substrings in the fittest combinations. The GA eventually settles on a cluster (or multiple clusters) of near-optimal individuals with similar bit strings. The convergence time and solution quality depend on the nature of the problem and the parameters that control the GA.

4 The GA-Robot Method

Since the performance of a navigational robot is determined by the values of its reactive control schema parameters, genetic algorithms can be used to optimize these parameters using the navigational performance of the robot as a fitness metric. We have developed a new method, GA-Robot, that uses a modified version of a standard genetic algorithm to learn optimum schema parameters under different conditions. The method is applied to a robot simulation in a two-dimensional world composed of static virtual obstacles and a single goal position, as shown in Figure 2. The task of each robot in a simulation is to move from the start to the goal, while avoiding obstacles along the way. All of the simulations use the same distance from start to goal but the clutter of environment, as measured by the area occupied by obstacles, varies. Robots are rewarded inversely to traversal time, distance travelled, and number of collisions. The size of a robot step can vary from zero to a maximum value, and depends on the current values of the schema parameters and the distance and position of the nearest obstacles and the goal. Collisions with obstacles, which are defined as intrusions into the safety margin surrounding obstacles, do not involve physical contact and are not lethal.

Let us consider the design of the simulations and the GA-Robot algorithm in more detail.

4.1 Simulation Design

A robot in our simulation uses three primitive behaviors: **move-to-goal**, **avoid-static-obstacle**, and **noise**, as described in Section 2.2. These behaviors define the structure of the navigation system. The three behaviors are controlled by five schema parameters:

- 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 through the environment. The velocity vectors contributed by each schema are multiplied by the schema gains. For example, if the robot is within the sphere of influence of an obstacle, the **avoid-static-obstacle** schema will determine a repulsive vector which will be multiplied by the obstacle gain. After the other schemas have calculated their vectors, all of the vectors are summed and normalized to give the robot's actual step. Because the structure of the system is fixed, a robot always exhibits the three behaviors with some strength, although setting a gain to zero essentially removes the behavior. Note that since the individual schema vectors are summed and normalized, the gain parameters are significant only in their relative magnitude.

The simulated world can vary in obstacle coverage and location. We used 1%, 10%, and 25% clutter worlds, where the clutter is defined by the percentage of the world area occupied by obstacles. Furthermore, we varied the robots' environments, not only with respect to degree of clutter, but also with respect to how the worlds changed between robot trials. There were three types: *fixed*, *varying*, or *general*, as shown in Figure 3. In fixed worlds, the obstacle sizes and locations remain constant throughout the evolution of the population. In varying and general worlds, the obstacles are replaced after each simulated trial of a robot. Thus, if a robot is given three trials per evaluation, it will be run through three different environments. In varying worlds, the percentage of clutter is constant, but the number, sizes and locations of the obstacles changes. For general worlds, each trial's world is created using a random percentage of clutter between zero and the specified percentage. The intent was to evolve robots in different types of worlds to study the formation of ecological niches. For example, we expected robots evolved in 25% fixed or varying worlds to be optimized for high-clutter worlds, whereas those evolved in 25% general worlds would be more suited to a broader range of worlds. Note that, in every case, the environment remains constant during each trial run.

The clutter values were chosen to simulate a variety of real environments from nearly empty to crowded. By optimizing for particular values of clutter, we created robots effective in these different environments. Furthermore, once trained, a robot might dynamically switch parameter sets to effectively navigate changing environments, such as in (Ram, Arkin, Moorman, & Clark, 1992; Ram & Santamaria, 1993). Recognizing and adapting to new environments can also be done via a high-level planner; this extension is an important issue for future research.

In addition to optimizing the robots for different types of worlds, we also evolved robots suited to particular niches by varying the weights on robot fitness penalties. For example, weighting the fitnesses to minimize collisions optimizes the robots for safety. In GA-Robot, the raw fitness function, shown in Figure 4 and discussed in Section 4.2, provides weights for collisions, time, and distance. We combined these into three robot types: *safe*, *fast*, and *direct*. Safe robots were optimized to avoid hitting obstacles. While both avoid collisions, fast robots prioritized speed, whereas direct robots preferred shorter trips. Example weightings are shown in Figure 5.

The basic problem is to choose reactive control schema parameters that create a specific kind of robot tailored to a specific kind of environment (or range of environments). This optimization is nontrivial, since the parameters are not orthogonal, and may interfere with each other. For example, the robot must approach the obstacles to reach the goal; the goal gain (which moves the robot towards the goal) and the obstacle gain (which moves the robot away from obstacles) must be balanced. Furthermore, the environment can present situations in which the obstacle-avoidance force cancels the goal-attraction force, such as in box canyons. As the world clutter increases, the likelihood of these regions increases. We used a genetic algorithm, as discussed in the next section, to evolve robots using different combinations of robot type, world clutter, and world type, as shown in Figure 6.

4.2 Genetic Algorithm Methodology

The standard genetic algorithm requires that the search-space points be encoded as binary strings because the genetic operators crossover and mutation are defined as bit operations. However, this is an unnecessary requirement because the operators conceptually require only an exchange or change of data; bit-strings are used as a conveniently splittable representation. Our method defines the chromosome as a list of floating point robot control parameters. By using a floating point representation for each of the parameter values, the simulation does not need to translate from the bit string to a value that can be used by the reactive control system, thus increasing the efficiency of the algorithm.

The algorithm starts by generating a population of robots, an environment containing obstacles, and a single goal. Each of the robots is initialized with a set of randomly generated values for the motor schema parameters. In each generation, the robots are run through the environment three times. The robots move through the environment until they either reach the goal or exceed the maximum allowed steps. Next, for varying or general worlds, the obstacles are destroyed and recreated with the same clutter (varying) or a random clutter (general). A running total of the raw fitnesses of each of the robots is maintained during the simulations, and used during the application of the genetic operators. The top-level code for the algorithm is shown in Figure 7.

Optimal robots maximize safety, speed, and efficiency. Our fitness function was designed to combine data measurable by the robot into a single measure maximized by the GA operators. *Safety* in our simulation was measured by the number of collisions with obstacles. *Speediness* was measured by the number of steps taken to find the goal. Finally, *directness* was measured by the distance travelled between the start and the goal. Note that since for each step, the robot may make a large or a small movement, these metrics distinguish between large, meandering movements, and small, direct ones. Each of these was multiplied by a weighting factor and summed to give the raw fitness, as shown in Figure 4.

As collisions, steps, and distance must be minimized, each raw fitness was subtracted from the largest raw fitness in the population to give the robot's final fitness, as illustrated in Figure 8. In addition to providing a value for the GA operators to maximize, this operation scales the fitnesses to maintain differentiation between individuals even when the population has converged. To see this, consider an early population whose best fitness is 750 and whose average is 500. By roulette wheel selection, we expect 1.5 copies of the best individual. Later in the simulation, the best is 1100 and the average is 1000. Now we only expect 1.1 copies of the best individual. Thus, the *selection pressure* has been reduced (Whitley, 1989). Now suppose the minimum fitness in the previous cases were 250 and 900. By scaling, we maintain a constant selection pressure: $(750 - 250)/(500 - 250) = (1100 - 900)/(1000 - 900) = 2$.

Although the genetic operators maximize the scaled fitness function, our goal is to minimize penalties which are represented by the raw fitness values. In the following discussion, we will refer to optimization in terms of minimizing the raw fitnesses.

Note that if the fitness function in Figure 4 is weighted heavily against distance, the algorithm will produce a robot that takes extremely small steps and never reaches the goal. If desired, the fitness function may be modified to add a penalty to the raw fitness of robots that fail to reach the goal. Furthermore, the weights must be scaled to account for the magnitudes of the metrics. This may be done automatically by normalizing the weights between the maximum and minimum values in the population. Finally, the fitness function may be extended to minimize other expenditures of an autonomous system's limited resources. For example, a more sophisticated robot model and fitness function may be used to reduce physical wear on the robot, the risks incurred in route choice, and difficulties due to terrain type. In general, the GA-Robot method presented here can be used with other fitness functions to optimize other parameters of interest, depending on the application.

After the raw fitness ratings has been accumulated over several simulations, the genetic operators are applied to the population. The reproduction operator selects the fittest individuals randomly in proportion to their fitnesses and copies them, replacing the least fit robots which are chosen randomly in proportion to their unfitness. The number of reproductions is also random, but controlled parametrically.

The crossover operator was modified to allow information exchange between floating-point chromosomes. The operator chooses a single gene location to exchange, unlike the bit-string crossover, which swaps all of the genes after the chosen crossover point. The exchange itself is made by first calculating the average and difference between the two genes at the exchange point of each mate. The two mates are copied, but the genes at the crossover point are given by `new_gene = average + difference * randn1p1()`, where `randn1p1()` returns a random number between negative one and positive one. The probability that the operator is applied to each gene location during a mating is controlled by the crossover probability parameter.

Finally, the mutation operator is applied to the population. It, too, was modified to work with the floating-point representation. If a gene mutates, the gene is increased or decreased by a random percentage, as given by `new_gene = old_gene + old_gene * randn1p1() * MUTATION_DELTA` . Note that, unlike standard bit-string mutation, the change depends on the value of the gene. The consequences of this type of mutation are discussed in Section 4.3.1.

4.3 Factors Affecting Learning

We ran extensive computer simulations to study the behavior of our system and to validate the floating-point extension of the genetic algorithm. Because the computer can run large numbers of simulations without human supervision, we were able to study a variety of factors that influence robot learning. These factors can be categorized broadly into three types, *algorithmic*, *robotic*, and *environmental*, as listed in Figure 9. Algorithmic factors provide insight into the behavior of the genetic algorithm itself, which is required to ensure that the algorithm is operating effectively. Robotic and environmental factors focus on different types of robots in different types of environments; these must be chosen to reflect the real robot and its likely environments. By using multiple sets of robotic and environmental factors, GA-Robot was able to optimize robot performance in response to both the kind of robot desired (safe, direct, or fast) and the kind of environment for which it was intended (cluttered/uncluttered and fixed/varying).

4.3.1 Genetic Algorithm Control Parameters

A primary difficulty in applying a genetic algorithm is determining good control parameters, such as the probabilities of reproduction, crossover and selection. Since extensive parameter studies defeat the purpose of using an optimization algorithm, a few variations suffice to provide confidence in the performance of the algorithm. While our empirically-determined reproduction and crossover probabilities were similar to established research (Davis, 1991), our mutation parameter was found to be equally optimal between 0.5 and 1.0. This was due to the relative mutation used by our algorithm. In a bit-string representation, the mutation operator randomly inverts bits. Depending on the significance of the bit in the decoded gene, the inversion may have a huge or a negligible effect. As a result, while too low mutation probability allows premature convergence, too high a probability reduces the algorithm to a random walk. Between those points, the noise is high enough to prevent premature convergence, but low enough not to hide the global minima among large, random steps. An optimal mutation rate can be found such that the global minima are findable, and more likely to be created by crossover than destroyed by mutation.

However, our algorithm cannot take noisy random walks because our floating-point mutation cannot have huge effects on the genes; the change is confined to a percentage of the present gene's value. While it may seem odd that mutation considers the magnitude of the gene, it is motivated in GA-Robot by the relative nature of schema gains. In schema-based reactive control, individual vectors contributed by each schema are multiplied by the gains, summed, then normalized; thus, a gain is only meaningful relative to the others. Because the mutations are within a percentage of the previous values, the mutation operator is effectively a random local search. This search prevents premature convergence and helps the algorithm find deeper minima.

4.3.2 Robot Type

The robot type is determined by the weightings used for the penalties in the raw fitness function, as described in Section 4.2. For example, a robot penalized more for collisions than time or distance will develop parameters which move it more slowly and cautiously. These penalties affect learning by altering the search function.

Robots optimized to avoid collisions do so at the expense of other abilities. For example, they may be unable to navigate between close objects. On the other hand, robots with reduced collision avoidance can navigate through most environments more quickly, but often collide with obstacles. The amount of punishment assigned to a collision should depend on the fragility of the robot being simulated. By optimizing for several combinations of penalties, a high-level planner could choose parameters based on the robot's condition and goals. For example, a robot may switch parameters to maximize directness when fuel is limited, speed when time is limited, or safety when durability is limited.

4.3.3 Environment Type

The environment type has a significant influence on the robot's learning. The density of the obstacles in the environment affects how much the best route deviates from a straight line from the start to the goal, and how frequently the robot has to make course changes. As the density of the environment increases, it becomes more efficient to go around a cluster of obstacles rather than to navigate through them and risk collisions. The obstacles may also vary in their organization; they may be totally random, be clustered into groups, or form highly-organized patterns such as fences or box canyons. Further, the obstacles or the goal may move as the robot travels.

Because robots learn to navigate in particular environments, a robot may be said to occupy a *niche* in the same way that animals evolve to fill a particular niche in nature. An effective strategy for dealing with unknown or changing environments would be to pretrain for several niches, then design a high-level controller to choose the niche parameters appropriate to the present world. The reactive control system would thus adapt to its current environment, making it more successful, efficient and robust. An alternative might be to train a robot in a wide range of environments so as to evolve a general-purpose robot.

In GA-Robot, we have two high-level controllers which monitor the robot and the environment and choose control parameters accordingly. One selects the robot type (safe, fast, or direct) based on the robot's condition and goals, while the other estimates the surrounding clutter and variability and chooses the nearest training percentage (1%, 10%, or 25%) and world type (fixed, varying, or general). The table in Figure 5.1, which lists the results of the genetic algorithm simulations, provides the optimal robot control parameters that GA-Robot learned; these are used by the high-level controllers to modify the behavior of the robot during navigation after the learning phase is over.

4.4 Evaluation Metrics and Simulation Results

The utility of applying genetic algorithms to robotic navigation learning can be evaluated in several ways. This section describes three methods used to determine algorithm effectiveness and solution quality in the GA-Robot system, and presents the results of our simulations.

4.4.1 Convergence Evaluations

One way to evaluate a GA-based system is to examine the convergence of the genes. The initial robot parameters are randomly generated and evenly distributed. If there is an optimal solution to the problem for a given simulation environment and fitness function, the population should converge toward this solution's set of values. The number of generations required for convergence depends on the GA's control parameters (i.e., population size, crossover probability, etc.) and the difficulty of the problem. Note that these methods evaluate the algorithm's operation, not its final effectiveness.

In each of the experiments, a trace of the five reactive control parameters was saved for later analysis. Given enough time to converge, each of the reactive control parameters of the populations converged to within about 2 percent of the starting value for that parameter.

To evaluate the performance of the genetic algorithm, we plotted the convergence of the population as the algorithm progresses. In general, better performance is defined by lower minima in fewer iterations. The convergence results for the safe robots in 1%, 10%, and 25% cluttered, fixed worlds are shown in Figure 10. These graphs plot the average, best, and worst *raw* fitnesses in the population for each generation. These curves show the weighted sum of collisions, steps, and distance decreases as the population evolves. Note that each generation combines three simulations of the same parameters to reduce random effects. Also, these simulations did not penalize robots that failed to reach the goal.

The graphs illustrate several typical features of the optimization process. First, note the discontinuous nature of the penalties in the fitness function, as shown by the spikes after convergence in the top graph. Because these simulations were set up to create a *safe* robot, collisions received a disproportional penalty. Although the population converged quickly to cautious robots, collisions could still occur and, when they did, caused large jumps in the worst fitness values.

The sequence shown in Figure 10 suggests faster convergence for the most cluttered environment. This is an effect of the safe robots, which are penalized for collisions. The dense environment provides more opportunities for collisions; the robots make more mistakes and quickly learn to avoid them. In general, however, the denser and changing environments required more iterations for convergence.

4.4.2 Objective Behavior Measures

As discussed above, a simple and common method of evaluation is to track robot fitnesses over several generations. Raw fitness values decrease as the simulation progresses, converging toward an optimal value. However, because the fitness function combines several measures of robot performance, it is not possible to determine which measure is most optimized for each iteration of the algorithm. For example, reducing the likelihood of collisions may decrease the raw fitness without increasing the directness of the path. While the minimization of the raw fitnesses is a good measure of algorithm performance, fitnesses alone are not a sufficient measure of robot optimization. However, the same metrics used by the robot, the number of collisions, the time required, and the distance travelled, provide independent measures of solution quality and are available for evaluation. For example, in our simulations of the safe robots, which weighted the collisions heavily, the number of the collisions quickly dropped to zero, causing an increase in the total number of steps as the robots slowed near obstacles.

The GA found improved parameter sets for almost all of the robot and environment types considered in our experiment design. The effectiveness of the optimization is shown in Figure 11. The values shown for collisions, steps, and distance are taken from the last of the three runs given each robot. In all but three instances, the robots show a clear improvement in the desired performance metric. In the cases in which the optimized robot shows *increased* numbers of collisions, the results may reflect a bad run in the series. However, in these cases, the robots were being optimized for speed or directness, not collisions.

4.4.3 Visualization of the Navigation

Although visualization methods are subjective, they can provide insights into robot learning. A visualization of the simulation gives a qualitative feel for the success of the learning algorithm and the emergent properties of the interacting schemas.

The GA simulations produce a trace of the robot moves for each run which is used by a separate program to display the paths of the robots as sequences of points in configuration space. An example is shown in Figure 2. Robots start at the square on the left and travel to the goal circle on the right. The black circles represent the obstacles scattered throughout the environment. Figure 12 shows three simulations from the optimization of direct robots in a 25% cluttered, general world. Recall that in the non-fixed environments, a new set of obstacles is generated for each iteration of the genetic algorithm. Further, in the general world, the area covered by the obstacles is a random value less than the nominal percentage. Thus, in Figure 12, the bottom two simulations are less cluttered than the top one.

Figure 12 shows the evolution of the direct robot population. The top simulation traces the paths of the initial population robots. The middle and lower simulations are intermediate and final populations. The initial population shows the effect of random parameter choices. Note that their paths are not totally random; given a random attraction to the goal, a random desire to avoid obstacles and a random noise parameter, they are still drawn toward the goal. While some of the robots do reach the goal, many do not. Of those that do, many of the paths are too direct, causing collisions. Thus, following the initial generations is a fanout in which robots become more cautious, but reach the goal. After several hundred further simulations, the robots tend to anticipate the obstacles, taking shortcuts rather than approaching, then having to maneuver around them. The final population shows the smooth paths taken by the direct robots.

The characteristic paths of the direct population can be better seen in relation to the other robot types, as shown in Figure 13. As expected, the safe robots (top) take the least direct routes, in some cases going outside the obstacle field entirely. The fast robots (middle) also take less direct routes than the direct robots, instead seeking the paths that enable them to travel at the highest speeds. They tend to follow wide trails, even if these trails occasionally lead away from the goal. Finally, the direct robots (bottom) show more fan-out than the fast robots as they seek the best short cuts. Because they can vary their speed to find direct lines to the goal, they are less constrained in their choice of path. The fast robots appear to choose the same paths because only a few paths allow maximum speed. The direct robots have some weighting for speed, so do not find the absolute shortest path.

The effect of increasing environmental clutter is shown in Figure 14. Although optimized to avoid obstacles, the safe robots find direct paths when the clutter is low (top). As the clutter increases, however, the paths begin to diverge (middle). Finally, in a cluttered environment, the robots find many indirect and slow routes through the obstacle field. Just as safe robots appear fast and direct in sparse environments, we observed that direct and fast robots become cautious as increasing clutter forces reductions in speed and reduces opportunities for short cuts.

Finally, consider the effect of the world type on the optimization. Figure 15 traces populations of fast robots optimized for fixed (top), varying (middle), and general (bottom) worlds. It appears that the varying-world robots are least able to find speedy paths, while the general-world robots do best. The fixed-world robots are expected to do well since they can find the existing fast paths through the unchanging world. The varying world removes this circumstance; the world is always cluttered, but never the same. These robots must try to balance speed and safety in situations that provide few and inconsistent opportunities for speed. The middle figure illustrates their inability to find consistent solutions. The general world, however, is often *less* dense because it is recreated each trial with a random clutter less than the nominal value (25% in this case). Thus the robots have many chances in sparse worlds to find parameters that give fast, safe behavior. Further, as shown in the bottom figure, their success in sparse worlds scales, enabling them to work effectively in denser ones. This suggests that training in sparse worlds allows the discovery of parameter sets that are useful, but harder to find, in cluttered worlds.

5 Conclusions

Genetic algorithms provide a powerful method for searching for near-optimal solutions in complex search spaces. Drawing from analogies in biology and evolution, they can be applied to the learning of robotic coordination in reactive control systems. In contrast to programming by humans, genetic algorithms allow not only rapid discovery of good behavioral parameters, but also tuning of parameters to varying robot and environmental conditions. An attractive feature of this approach is that the designer need only specify the robot and the environment; a deep understanding of the robot's behavior and environmental interactions is not required. The algorithm will abstract the salient features of the environment as it searches for parameters that effect good performance in it. For example, the fast robots in our simulations learned to find wide paths which enabled greater speeds.

In this paper, we presented a novel GA-based method to optimize robotic control parameters in a schema-based navigation system. The system, called GA-Robot, is fully implemented and has been evaluated extensively. In addition to demonstrating the power of the method, the experimental results discussed above provide insight into the working of the system, and provide a basis for appropriate design of the genetic algorithm (for example, selection of crossover probability). The results also provide a basis for the application of our method to other kinds of problems (for example, constraints on types of robots, desired ranges of environments, or alternative performance metrics).

Our research treats GA learning as a way to integrate global path planning and local navigation. This work provides the foundation for the integration of the learned parameters into a hierarchical navigation system. Future work will extend GA training to mediate the subsystems. The global path planner will pass the positions of subgoal landmarks, the condition of the robot, and the crowdedness of the local environment to the schemas, thus affecting where the robot goes and how carefully it navigates through the obstacles, without explicitly stating what path to take. This separation of local and global navigation tasks is one of the strengths of the schema-based approach to reactive control.

5.1 Acknowledgements

The Georgia Tech Mobile Robotics Lab is supported by the National Science Foundation under grants IRI–9100149 and IRI–9113747. We thank Paul Panaro for his contributions to this research.

References

J. Albus, H. McCain, & R. Lumia. (1987). NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). NBS Technical Note 1235, Robot Systems Division, National Bureau of Standards, Washington, D.C.

T. Anderson & M. Donath. (1991). Animal Behavior as a Paradigm for Developing Robot Autonomy. In *Designing Autonomous Agents.* P. Maes (Ed.), Cambridge, MA: MIT Press.

R. Arkin. (1989). Motor Schema-Based Mobile Robot Navigation, *International Journal of Robotics Research*, Vol. 8, No. 4, pp. 92–112.

R. Arkin & R. Murphy. (1990). Autonomous Navigation in a Manufacturing Environment, *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 4, pp. 445–454.

R. Arkin, E. Riseman, & A. Hanson. (1987). AuRA: An Architecture for Vision-based robot Navigation, *DARPA Image Understanding Workshop*, pp. 417-431, Los Angeles.

A. Barto, C. Anderson, & R. Sutton. (1982). Synthesis of Nonlinear Control Surfaces by a Layered Associative Search Network. *Biological Cybernetics* Vol. 43, pp. 175–85.

R. Brooks. (1989). The Whole Iguana. *Robotics Science*, pp. 433-56.

R. Brooks. A robot that walks: Emergent behaviors from a carefully evolved network. (1989). In *IEEE International Conference on Robotics and Automation, Vol. 2*, pp. 692–696.

L. Davis (Ed.). (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold.

M. Dorigo & U. Schnepf. (1991). Organization of Robot Behavior Through Genetic Learning Processes. *Fifth International Conference on Advanced Robotics*, Vol. 2, pp. 1456–60.

A. Elfes. (1987). Sonar-based Real-world Mapping and Navigation. *IEEE Journal of Robotics and Automation.* R-A-3(3), pp. 249–265.

R. Fikes, P. Hart, & Nils Nilsson. Learning (1972). and Executing Generalized Robot Plans. *Artificial Intelligence*, Vol. 3, pp. 251–88.

D. Goldberg. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company.

D.F. Gordon & D. Subramanian. (1993). A Multistrategy Learning Schema for Assimilating Advice in Embedded Agents. *Second International Workshop on Multistrategy Learning*, pp. 218–233.

J.J. Grefenstette, C.L. Ramsey, & A.C. Schultz. (1990). Learning Sequential Decision Rules using Simulation Models and Competition. *Machine Learning*, Vol. 5, No. 4, pp. 355–381.

C. Janikow & Z. Michalewicz. (1991). An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms. *Fourth International Conference on Genetic Algorithms*, pp. 31–36.

L. Kaelbling.(1986). An Architecture for Intelligent Reactive Systems. *SRI Technical Note*, V 400, SRI International. October.

D. Payton. (1986). An Architecture for Reflexive Autonomous Vehicle Control. *IEEE Conference on Robotics and Automation*, pp. 1838–1845.

D. Payton, D. Keirsey, J. Krozel, & K. Rosenblatt. (1992). Do whatever works: A robust approach to fault-tolerant autonomous control. *Applied Intelligence*, Vol. 2, No. 3, pp. 225–250.

A. Petland & R. Bolles. (1989). Learning and Recognition in Natural Environments. *Robotics Science*, pp. 164–207.

A. Ram, R.C. Arkin, K. Moorman, & R.J. Clark. (1992). Case-Based Reactive Navigation: A Case-Based Method for On-Line Selection and Adaptation of Reactive Control Parameters in Autonomous Robotic Systems. Technical Report GIT-CC-92/57 College of Computing, Georgia Institute of Technology, Atlanta, GA.

A. Ram, & J.C. Santamaria. (1993). Multistrategy Learning in Reactive Control Systems for Autonomous Robotic Navigation. *Informatica*, Vol. 17, No. 4, pp. 347–369, 1993.

D. Whitley. (1989). The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. *Third International Conference on Genetic Algorithms*, pp. 116–121.

Figures

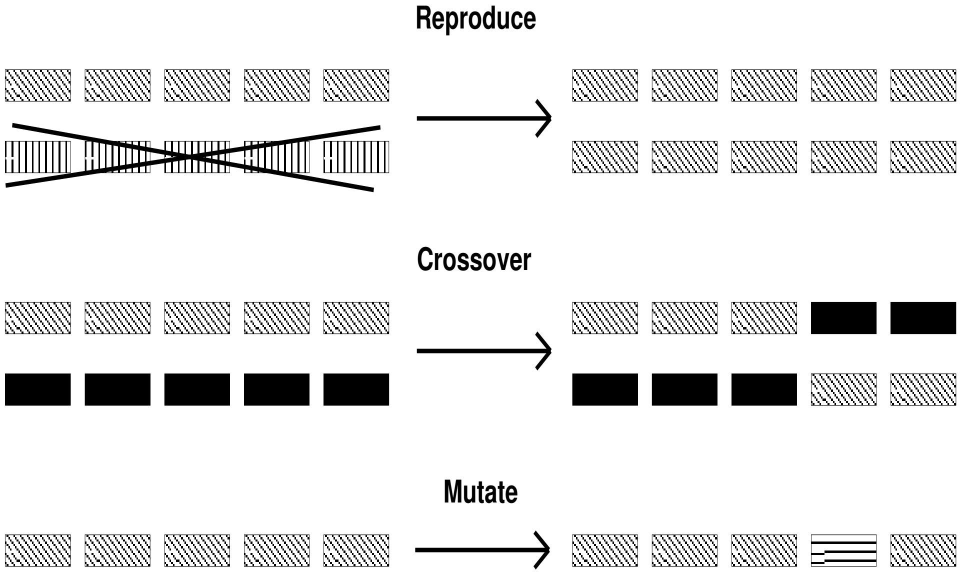

Figure 1: Genetic operators

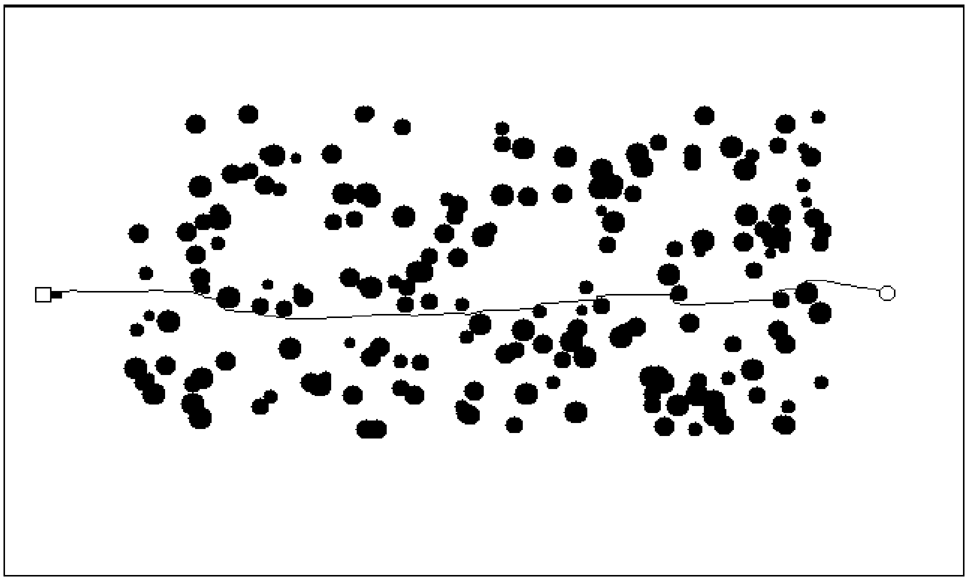

Figure 2: Example of a robot path in a 25% cluttered world

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

Figure 3: Types of worlds, specifying change in obstacles after each trial

```
raw_fitness = collision_weight * number_of_collisions
                  + time_weight * number_of_steps
                  + distance_weight * distance_travelled
```

Figure 4: Raw fitness, defined as a function of weighted penalties

Robot Type	*Weight*		
	Collision	*Time*	*Distance*
Safe	100	1	1
Fast	10	10	1
Direct	10	1	10

Figure 5: Example weightings for three types of robots

Expt. #	*Robot*	*Clutter*	*World*
1–9	Safe	1%, 10%, 25%	Fixed, Varying, General
10–18	Fast	1%, 10%, 25%	Fixed, Varying, General
19–27	Direct	1%, 10%, 25%	Fixed, Varying, General

Figure 6: Experiment Design

```
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 7: Top-level code for the genetic algorithm simulation.

```
begin
   highestRawFitness = calc_highest_raw_fitness();
   lowestRawFitness = calc_lowest_raw_fitness();
   totalRawFitness = calc_total_raw_fitness();
   /*  Scale/invert the raw values so that
       maximization by the GA will minimize
       the steps/collisions/dist. Note that the
       fitnesses are not allowed to equal zero  */
   for 1 to POPULATION_SIZE do
       citizens[indiv]->setFitness(
            highestRawFitness + 1 -
                 citizens[indiv]->getRawFitness());
end
```

Figure 8: Algorithm for evaluation of the population.

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*

Figure 9: Factors that affect the genetic algorithm and robot learning

Robot	Clutt.	World	Colls.	Steps	Dist.	Goal gain	Obst. dist.	Obst. gain	Noise gain	Noise pers.	Relative Gains		
											Goal	Obst.	Noise
Safe	1%	Fixed	0	199	398	4.460	9.095	3.213	-0.026	12.3	0.58	0.42	0.00
		General	0	198	395	2.166	1.946	11.043	0.168	113.1	0.16	0.83	0.01
		Varying	0	198	395	3.059	1.876	14.052	0.127	5.3	0.18	0.82	0.01
	10%	Fixed	0	250	412	1.750	3.866	2.421	0.287	5.9	0.39	0.54	0.06
		General	0	198	396	3.931	8.796	1.349	0.263	25.2	0.71	0.24	0.05
		Varying	0	210	405	2.229	4.926	2.650	0.307	80.6	0.43	0.51	0.06
	25%	Fixed	0	275	461	1.935	3.329	3.658	1.263	21.1	0.28	0.53	0.18
		General	0	248	401	1.750	4.814	1.662	0.418	54.8	0.46	0.43	0.11
		Varying	0	366	467	1.294	4.925	3.035	0.839	17.0	0.25	0.59	0.16
Fast	1%	Fixed	0	198	395	3.389	83.231	0.424	-0.017	0.7	0.89	0.11	0.00
		General	0	198	395	6.606	1.718	0.026	-0.054	4.6	1.00	0.00	0.00
		Varying	0	198	395	4.132	0.321	0.216	-0.365	1.2	0.95	0.05	0.0
	10%	Fixed	5	206	411	2.251	-0.900	0.638	0.258	3.2	0.72	0.20	0.08
		General	0	198	395	4.600	-1.066	1.858	0.256	5.9	0.69	0.28	0.04
		Varying	1	200	399	3.076	4.027	0.428	0.289	15.2	0.81	0.11	0.08
	25%	Fixed	12	234	468	5.508	21.744	2.002	-0.194	14.5	0.73	0.27	0.0
		General	0	199	398	6.729	3.315	2.670	0.483	1.5	0.68	0.27	0.05
		Varying	10	223	442	2.944	15.357	1.423	0.403	28.7	0.62	0.30	0.09
Direct	1%	Fixed	0	297	396	1.316	1.731	0.819	0.112	15.2	0.59	0.37	0.05
		General	0	198	395	4.381	8.855	0.537	0.179	4.1	0.86	0.11	0.04
		Varying	0	198	395	3.368	7.748	1.685	0.013	2.3	0.67	0.33	0.00
	10%	Fixed	0	422	401	0.974	1.744	1.040	0.088	1.7	0.46	0.50	0.04
		General	0	199	398	2.479	2.185	1.585	0.068	2.6	0.60	0.38	0.02
		Varying	0	262	400	1.550	9.524	0.581	0.164	25.8	0.68	0.25	0.07
	25%	Fixed	0	680	468	0.945	4.941	1.228	0.239	8.0	0.39	0.51	0.10
		General	1	354	407	1.205	3.713	0.653	0.029	37.0	0.64	0.35	0.02
		Varying	0	337	414	1.229	3.083	1.543	0.412	12.6	0.39	0.49	0.13

Figure 10: Final optimization results. Negative parameters are effectively zero. The relative gains may not sum to one due to rounding.

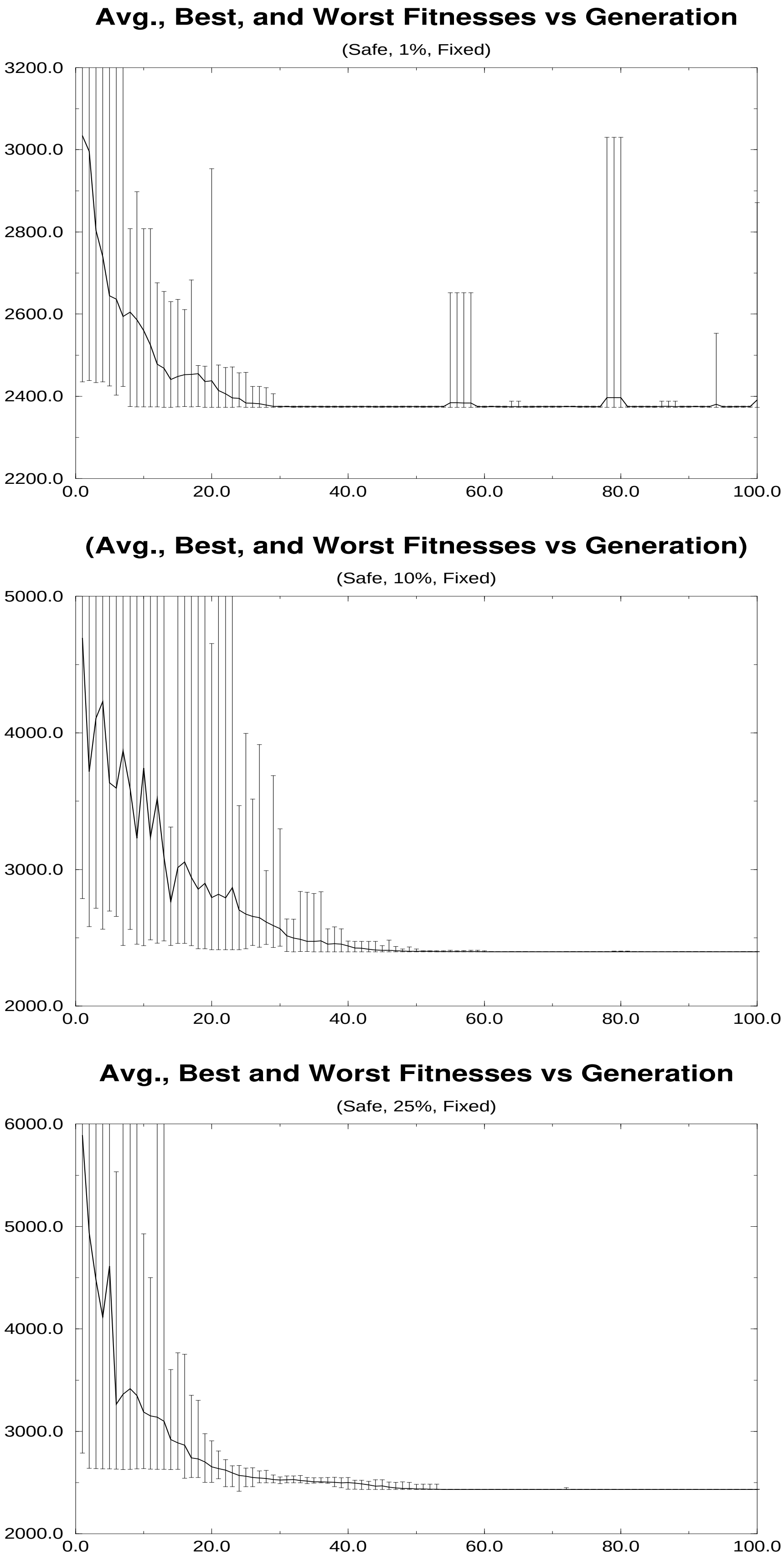

Figure 10: Convergences for a Safe Robot in 1%, 10%, and 25% cluttered, fixed worlds. Note that these are the raw fitnesses, which decrease as collisions, steps, and distance decrease.

			Collisions			Steps			Distance		
Robot	*Clutt.*	*World*	*Init.*	*Best*	*Change*	*Init.*	*Best*	*Change*	*Init.*	*Best*	*Change*
Safe	1%	Fixed	0	0	-	399	199	50.1%	400	398	0.5%
		General	0	0	-	409	198	51.6%	399	395	1.0%
		Varying	0	0	-	403	198	50.9%	411	395	3.9%
	10%	Fixed	0	0	-	1043	250	76.0%	435	412	5.3%
		General	0	0	-	395	198	49.9%	415	396	4.6%
		Varying	0	0	-	382	210	45.0%	412	405	1.7%
	25%	Fixed	0	0	-	1098	275	75.0%	520	461	11.3%
		General	0	0	-	477	248	48.0%	434	401	7.6%
		Varying	0	0	-	665	366	45.0%	470	467	0.6%
Fast	1%	Fixed	0	0	-	404	198	51.1%	413	395	4.4%
		General	0	0	-	427	198	53.6%	408	395	3.2%
		Varying	0	0	-	403	198	50.9%	411	395	3.9%
	10%	Fixed	6	5	16.7%	459	206	55.1%	432	411	4.9%
		General	0	0	-	395	198	49.9%	415	395	4.8%
		Varying	0	1	-	382	200	47.6%	412	399	3.2%
	25%	Fixed	0	12	-	1098	234	78.7%	520	468	10%
		General	0	0	-	477	199	58.3%	434	398	8.3%
		Varying	165	10	94.0%	775	223	71.2%	929	442	52.4%
Direct	1%	Fixed	0	0	-	399	297	25.6%	400	396	1.0%
		General	0	0	-	409	198	51.6%	399	395	1.0%
		Varying	0	0	-	420	198	52.9%	400	395	1.3%
	10%	Fixed	2	0	-	750	422	43.7%	411	401	2.4%
		General	0	0	-	673	199	70.4%	398	398	-
		Varying	0	0	-	818	262	68.0%	412	400	2.9%
	25%	Fixed	0	0	-	1098	680	38.1%	520	468	10%
		General	0	1	-	477	354	25.8%	434	407	6.2%
		Varying	0	0	-	665	337	49.3%	470	414	11.9%

Figure 11: Optimization effectiveness. The values are derived from the best individual of the initial population and the best individual in the subsequent generations.

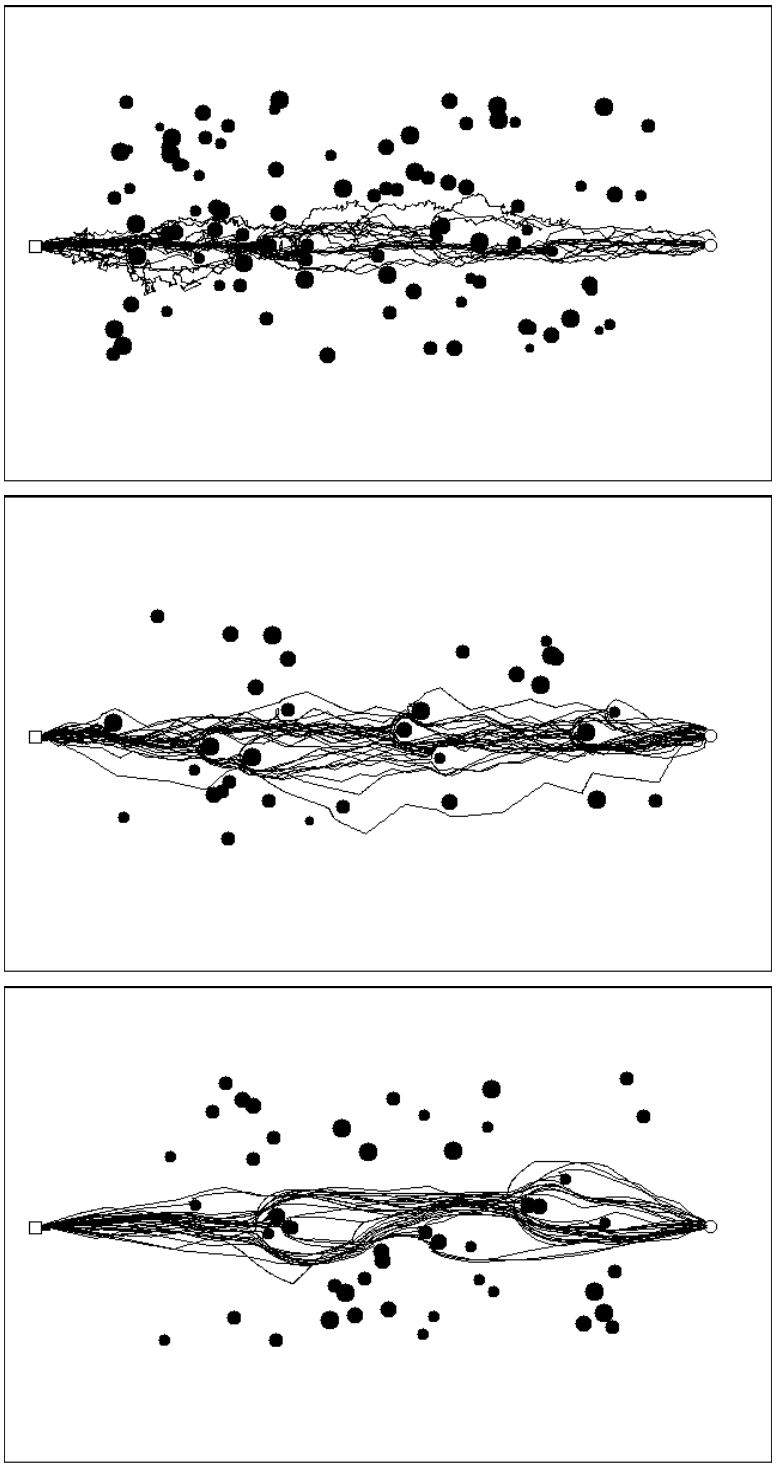

Figure 12: Paths (left to right) of direct robots through 25% cluttered general world (initial, intermediate, and final populations). Note that not all of the robots in the initial population reached the goal. Those that did incurred many collisions.

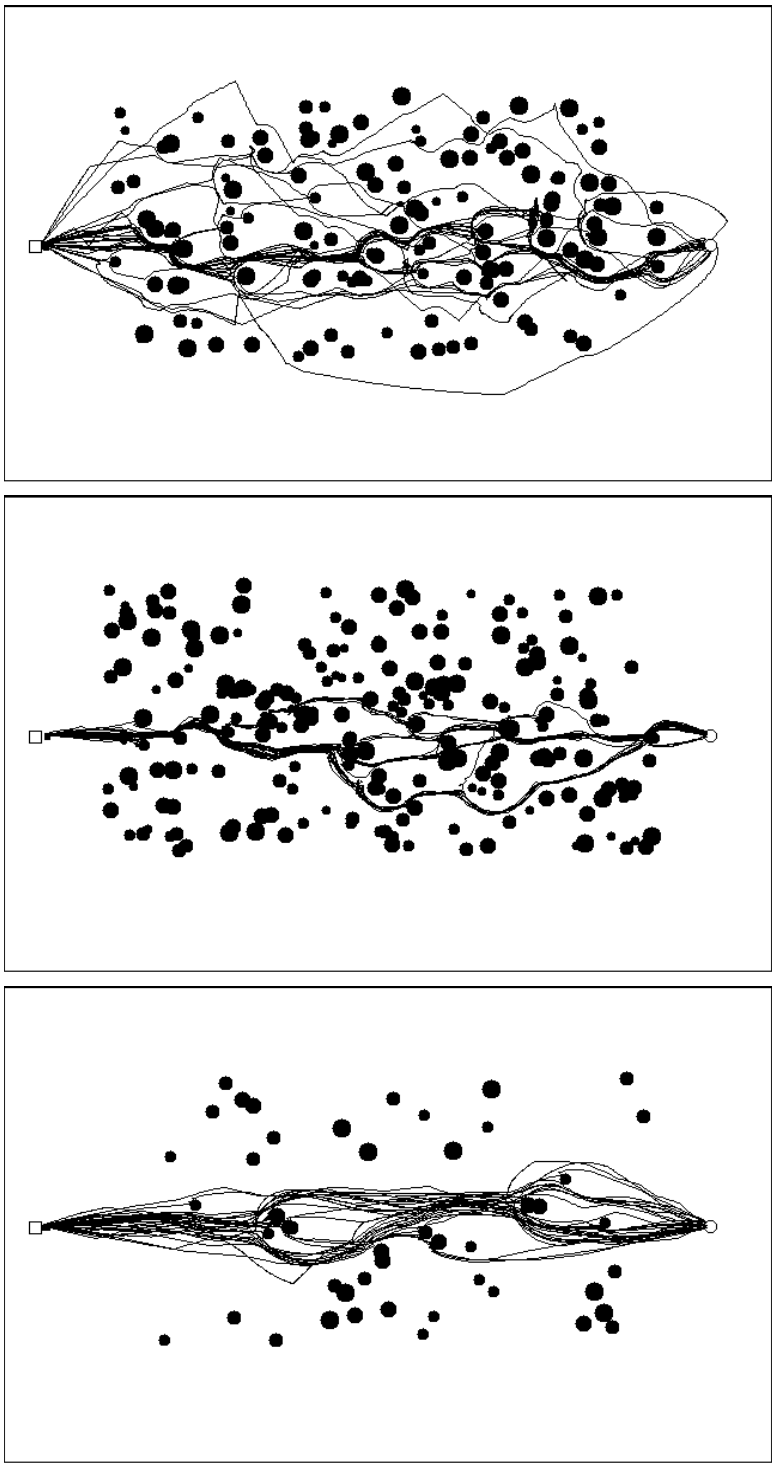

Figure 13: Final paths through 25% cluttered general worlds of safe (top), fast (middle), and direct (bottom) robots.

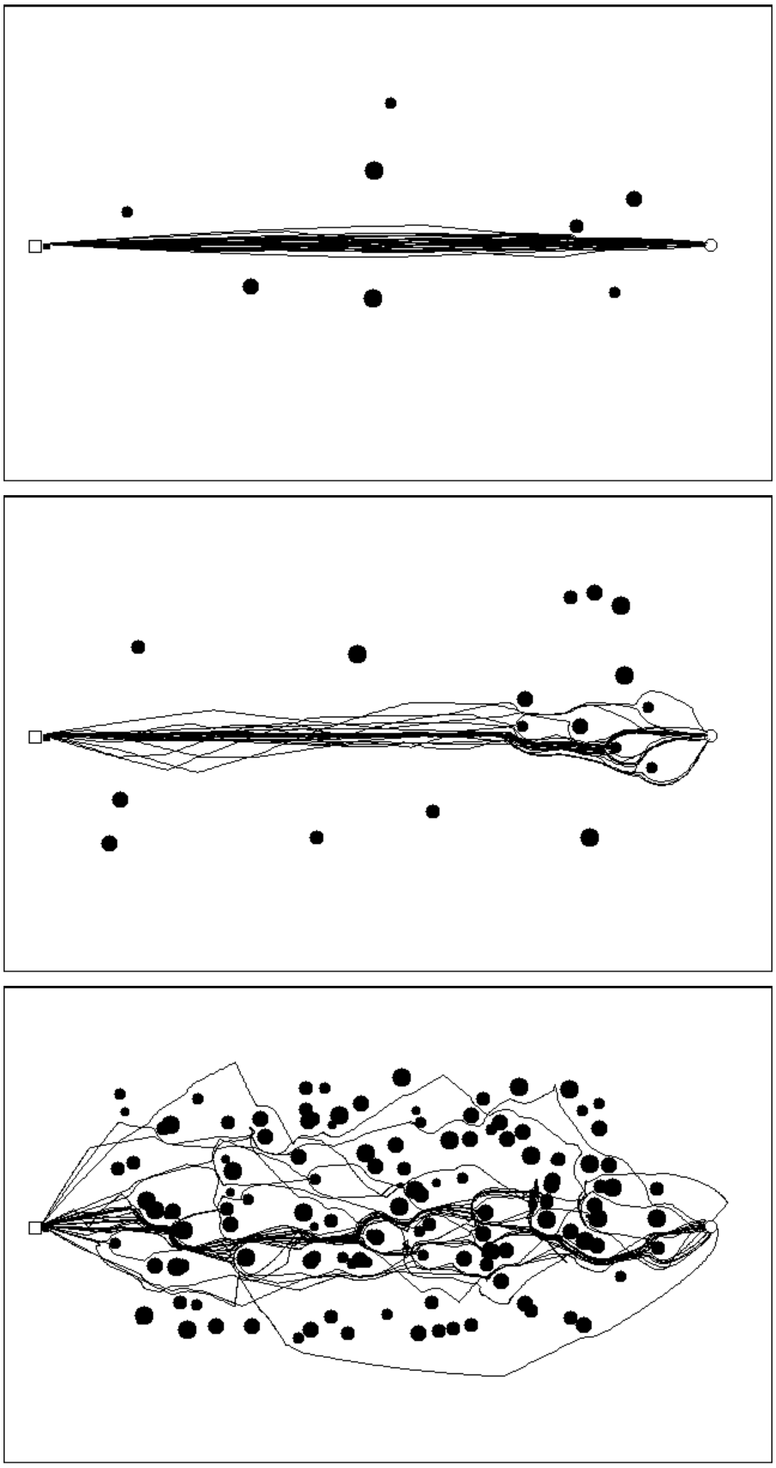

Figure 14: Final paths of safe robots through 1% (top), 10% (middle), and 25% (bottom) cluttered general worlds.

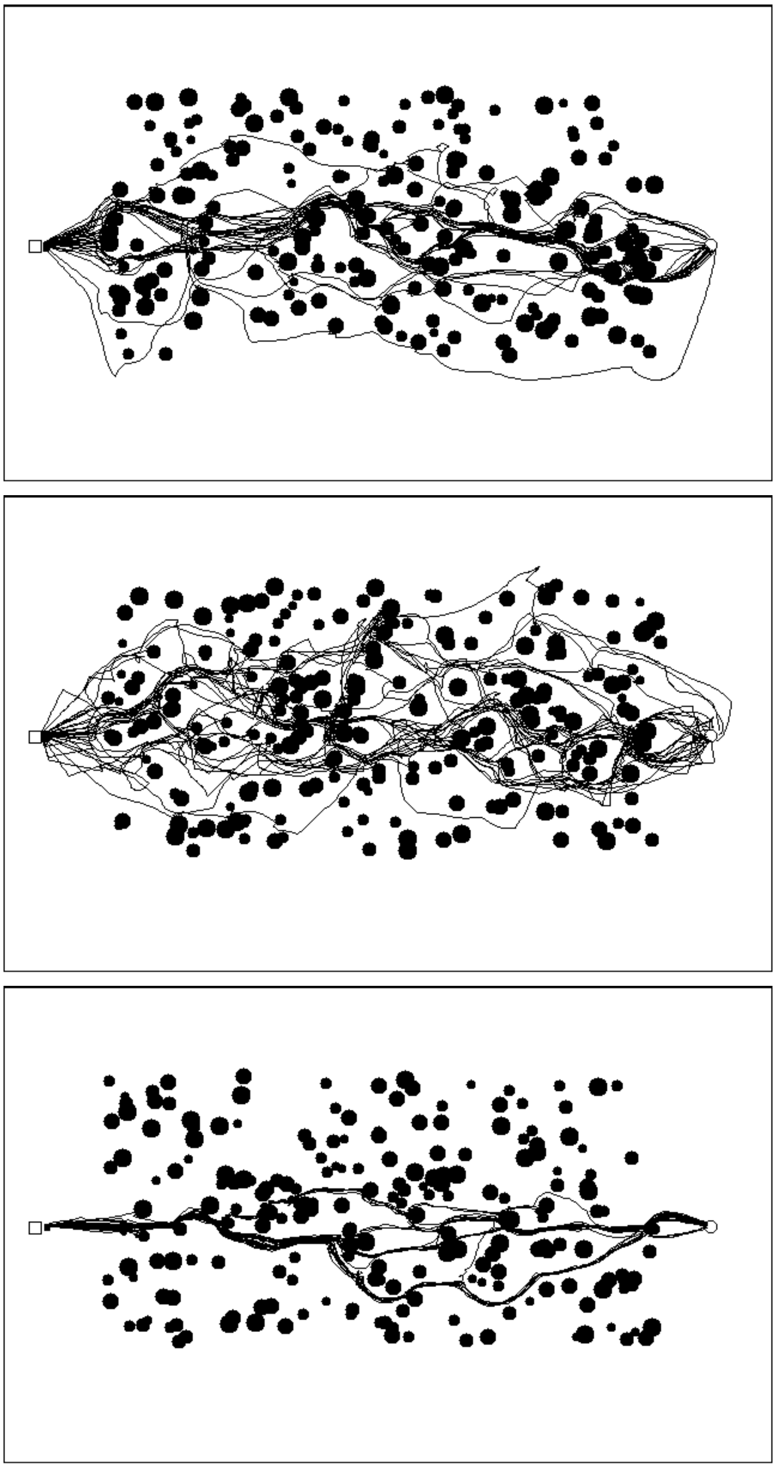

Figure 15: Final paths of fast robots through 25% fixed (top), varying (middle), and general (bottom) worlds.

Tables