Combining Robot Control Strategies using Genetic Algorithms with Memory.

Sushil J. Louis and Gan Li

Department of Computer Science
University of Nevada
Reno - 89557
sushil@cs.unr.edu

Abstract. We use a genetic algorithm augmented with a long term memory to design control strategies for a simulated robot, a mobile vehicle operating in a two-dimensional environment. The simulated robot has five touch sensors, two sound sensors, and two motors that drive locomotive tank tracks. A genetic algorithm trains the robot in several specially-designed simulation environments for evolving basic behaviors such as food approach, obstacle avoidance, and wall following. Control strategies for a more complex environment are then designed by selecting solutions from the stored strategies evolved for basic behaviors, ranking them according to their performance in the new complex environment and introducing them into a genetic algorithm's initial population. This augmented memory-based genetic algorithm quickly combines the basic behaviors and finds control strategies for performing well in the more complex environment.

1 Introduction

One of the main concerns in robotics is to plan a path for a robot system moving purposely and safely in an environment filled with known or unknown obstacles. Using the sensor motion planning approach, information about obstacles is assumed to be unknown or only partially known and local on-line information is assumed to come from sensory feedback. Since no detailed model of the environment is assumed, planning is performed continuously based on whatever partial information is available at the moment. The advantages of sensor motion planning are twofold: (1) it can deal with unstructured environments and the uncertainty typical of such environments, and (2) it requires much less memory or computation because relatively little information has to be processed during each step. On the negative side, generality is an elusive goal and optimality is usually ruled out.

A control strategy, mapping sensory feedback into a robot's actuators, is an essential component for a mobile robot under the sensor motion planning model. It can be designed by a human according to both the physical structure and the behavior requirements of the robot. However, human design of control strategies doesn't always work well because sometimes desired behaviors are fuzzy and difficult to explicitly define and not all useful behaviors of an autonomous robot can be determined a-priori, or recognized by humans.

During the last decade, much work has been done to explore the evolution of robot control strategies. A series of technical reports has been published by Cliff, Husbands, and Harvey on using genetic algorithms (GAs) to design neural-network control architectures for a simulated visually guided robot [1]. Koza has used genetic programming to evolve LISP programs that control and guide a variety of simulated robots performing navigation and other tasks [5]. Murray and Louis used genetic algorithms to first design combinational circuits for basic (low-level) behaviors, then used the genetic algorithm to design a switch to choose between these low-level behaviors for performing more complex tasks [10].

We cast low-level robot control strategy design as a search problem in a search space of possible strategies and use a non-traditional genetic algorithm to computationally design control strategies, in the form of a combinational circuit connecting sensor inputs to actuators, for a simulated robot (simbot) which can navigate and eat food in a two-dimensional environment with rectangular obstacles. At first, the simbot learns (and memorizes) basic behaviors such as food approach, obstacle avoidance, and wall following in specially-designed separate simulation environments. The best performing simbot from each environment can be considered an expert at one specific behavior and its control strategies must have some useful building blocks corresponding to this behavior. Next, seed solutions (cases) are selected from these experts by calculating and ranking their performance in a new and more complex target environment. Finally, we inject these cases as seeds into the initial population of another GA running in the more complex target environment. Selecting the "right" number of "appropriate" cases results in speedy design of promising control strategies for the simbot. Our strategy therefore seeks to provide a genetic algorithm with a long term memory in the form of a case-base, borrowing ideas from the field of case-based reasoning [12].

In the next section, we introduce the traditional genetic algorithm and describe our modifications. In addition we provide a brief description of case-based reasoning and the combined system. Section 4 describes the simulated robot and its environment. We present the experimental parameters used by our system in section 5. Experimental results are displayed and analyzed in section 6, followed by conclusions and future work.

2 A Genetic Algorithm

Genetic algorithms (GAs) are stochastic, parallel search algorithms based on the mechanics of natural selection, the process of evolution [4, 3]. GAs were designed to efficiently search large, non-linear, poorly-understood search spaces where expert knowledge is scarce or difficult to encode and where traditional optimization techniques fail. They are flexible and robust, exhibiting the adaptiveness of biological systems. As such, GAs appear well-suited for searching the large, poorly-understood spaces that arise in design problems; specifically designing control strategies for mobile robots.

2.1 The CHC Genetic Algorithm

CHC, the non-traditional genetic algorithm used in this paper, differs from traditional GAs in a number of ways [2]:

1. For a population of size N, it guarantees the best individuals found so far always survive by putting the children and parents together and selecting the best N individuals for further processing. In a traditional GA, the parent population does not survive to the next generation.
2. To avoid premature convergence, two similar individuals separated by a small Hamming distance (this threshold is set by the user) are not allowed to mate.
3. During crossover, two parents exchange exactly oned- half of their randomly selected *non-matching* bits.
4. Mutation isn't needed during normal processing.
5. Instead, an external mutation operator re-initializes the population when the population has converged or search has stagnated.

The CHC genetic algorithm generally does well with small populations [2]. Limited resources and the computational cost of the simulations led to our use of small populations and selection of the CHC genetic algorithm for this work.

2.2 Case-Based Reasoning

Case-based reasoning (CBR) is based on the idea that reasoning and explanation can best be done with reference to prior experience, stored in memory as *cases* [12]. When confronted with a problem to solve, a case-based reasoner extracts the most similar case in memory and uses information from the retrieved case and any available domain information to tackle the current problem. This paper uses the basic tenet of CBR — the idea of organizing information based on "similarity" — to help augment genetic algorithm search.

3 Combining GAs and CBR

Combining genetic algorithms with a long term memory model, like case-based reasoning, combines the strengths of both approaches. The case-base does what it is best at — memory organization; the genetic algorithm handles what it is best at — adaptation. The resulting combination takes advantage of both paradigms; the genetic algorithm component delivers robustness and adaptive learning while the case-based component speeds up the system. Furthermore, in many application areas we confront sets of similar problems. It makes little sense to start a problem solving search attempt from scratch with a random initial population when previous search attempts may have yielded useful information about the search space. Instead, seeding a genetic algorithm's initial population with solutions to similar previously solved problems can provide information (a search bias) that, hopefully, increases the efficiency of the search. If no useful information was obtained or obtainable, a randomly initialized population may be our only choice. Our approach borrows ideas from case-based reasoning (CBR) in which old problem and solution information, stored as cases in a case-base, help solve a new problem [12]. Although we restrict ourselves to genetic algorithms in this paper, we should be able to substitute, with minor modifications, any population based search algorithm for the genetic algorithm. We believe that evolutionary programming, genetic programming, and evolution strategies are especially suitable. Figure 1 shows a conceptual view of a first version of our system.

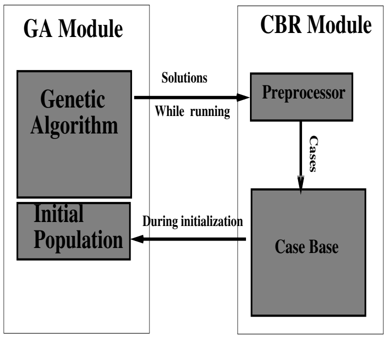

Fig. 1. Conceptual view of our system

Previous work in this area includes Louis, McGraw, and Wyckoff's paper that applied Case-Based Reasoning (CBR) to GAs as an analysis tool for the parity combinational circuit design problem [7]. Ramsey and Grefensttette seeded a genetic algorithm's initial population with cases and reported improved performance on the non-stationary functions that they used [11]. More recent work by Louis, Liu, and Xu addresses the questions of which cases are "appropriate" and how many cases to inject [6, 9] and establishes the feasibility of the method using the open-shop scheduling and rescheduling problem and the combinational circuit design problem.

4 Simulation

For our problem, the environment is a bounded square area with 300 units on a side as shown in Figure 2. There are several obstacles and food sources in the simulation environment and obstacles are modeled by rectangular boxes. The robot cannot enter either the boundary or the obstacles and the locations and the amount of food are fixed in one environment. Food sources produce a sound signal that can be detected by the simbot. A food signal can penetrate obstacles and can be heard by the simbot only if the simbot is within the signal range. If the distance between the robot center and a piece of food is less than or equal to five units, the food is assumed to be "eaten" and disappears from the environment.

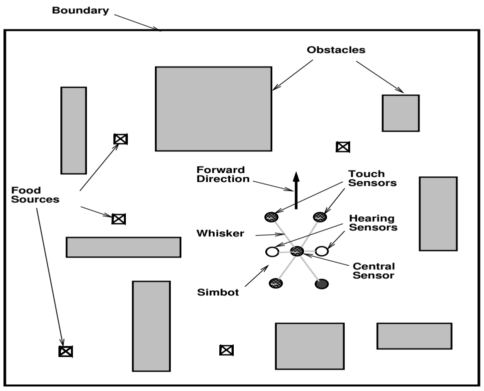

Fig. 2. Simulation environment and simbot

4.1 Simbot

Figure 2 also shows that the simbot has four touch sensors, one central touch sensor, two hearing sensors, and two locomotive tank tracks. Each touch sensor or hearing sensor is fixed on the end of a one-dimensional flexible whisker. The touch sensors and hearing sensors simulate hands and ears letting the robot feel obstacles and hear food in the environment. Each sensor has two states: 0 or 1. A combinational logic circuit maps the sensor states into binary control commands for each locomotive tank track. The tracks have two forward, one stop, and one reverse speed; the four possible speeds need two bits to encode. The simbot moves by coordinating the two tank tracks.

4.2 Encoding

In this paper, the control circuit is a 7×6 gate array that must be encoded into a binary chromosome. There are $(7 \times 6) - 6 = 36$ useful logical gates in the gate circuit because only four out of seven outputs of the control circuit are used for expressing the binary control commands for the two robot tracks. For each gate, four bits are needed for expressing the 16 possible logic gates with two inputs and one output. Therefore the chromosome length will be $36 \times 4 = 144$. We map a two-dimensional logic circuit to a one-dimensional chromosome by concatenating adjacent rows [8].

The simulation process provides an evaluation of a candidate combinational circuit for controlling the robot. The encoded chromosome of the combinational logical gate is obtained from the GA and evaluated in the simulation environment using a fitness function that measures how well the simbot performed its assigned task. The fitness value is returned back to the GA.

5 Experimental Setup

In the first three experiments, the robot was trained to develop the three basic behaviors of food approach, obstacle avoidance, and wall following separately in three different simulation environments. The environments are shown in Fig. 3. The solutions found here serve as seeds for developing control strategies for a navigation task (find all the food) in a complex environment that looks like an office area with rooms (open space) separated by walls (large obstacles). There are four food sources distributed in four of the nine rooms.

Each simulation process consists of $1,000$ time steps. Both the starting position of the robot and the initial sensor values are fixed for each experiment. The performance of the robot was evaluated using specific fitness parameters comprised of seven parts calculated at each time step and summed over all time steps for a final fitness value. The seven parts are listed in Table 1.

Table 1. The components of the fitness function

Parameter	Description
f1	a bonus for hearing food
f2	a penalty for collision
f3	a bonus for long, straight, and forward motion
f4	a bonus for moving along an obstacle or boundary
f5	a penalty for head-on touch against an obstacle or boundary
f6	a bonus for eating all food in less than 1,000 steps
f7	a huge bonus for eating food

The fitness of a candidate circuit is a weighted sum of these seven components and we can emphasize one or more behaviors for the simulated robot by adjusting the weights. We ran the GA ten times for each experiment with different random seeds. In all experiments, the genetic algorithm's population size was 100 run for 100 generation. Each chromosome was 144 bits long. The crossover probability was 1.0 and no normal mutation is needed as mentioned before.

6 Results and Analysis

6.1 Evolving Three Basic Behaviors

In the first three experiments, we use CHC without scaling and the entire initial population is randomly generated. The threshold for the Hamming distance is (length-of-chromosome/4) = 36 as is the norm for CHC. Scaled CHC is used to overcome the possible monopoly of solutions with extremely high fitness (caused by injection) in the complex environment. The fitness functions that were used are shown below:

1. **Food Approach (FA):** $f1 + f2 + f3 + f4 + f5 + f6 + f7$
2. **Obstacle Avoidance (OA):** $f1 + (5 \times f2) + f3 + f4 + (5 \times f5) + f6 + f7$
3. **Wall Following (WF):** $f1 + f2 + f3 + (5 \times f4) + f5 + (2 \times f6) + f7$

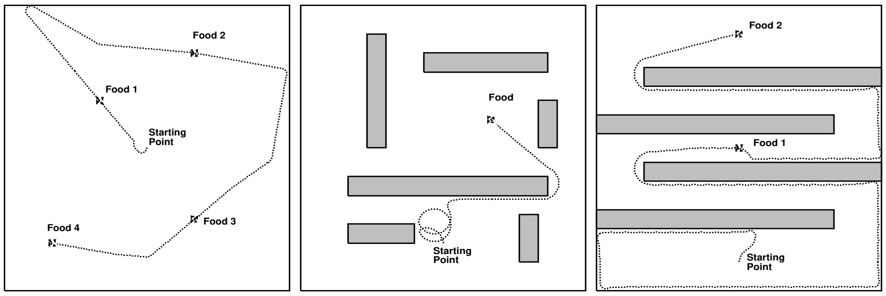

Fig. 3. The simbot's path when learning food approach (left), obstacle avoidance (middle), and wall following (right) behavior

CHC proved to be a reasonable and effective method for designing basic control strategies for a simbot. As shown in Fig 3, the simbots have successfully evolved the expected basic behaviors of food approach, obstacle avoidance, and wall following. This figure depicts the paths taken by the best individual for each of these first three experiments. Note that the control circuits may not be optimal.

6.2 Designing Control Strategies in an Office Environment

In the next set of experiments we designed the control strategies of a simbot in a complex office environment by injecting a suitable number of appropriate cases into the GA's initial population. First, we copied one case corresponding to the best individual for a basic behavior from each of the first three experiments for a total of three cases. Second, five cases were selected from the best 30 candidates of the first three experiments' results according to the candidates' fitness values *in the office environment.* We found that injecting five cases produced better performance than injecting a larger or smaller number of cases. We believe, that injecting a larger number of cases leads to insufficient exploration and injecting a fewer number of cases leads to insufficient exploitation. Five percent is a happy medium. We call the GA injected with these cases the **T**arget **R**anked **G**enetic **A**lgorithm or TR-GA, and the GA injected with the best cases in the basic behavior environments the **S**ource **R**anked **G**enetic **A**lgorithm or SR-GA. We also ran the GA with a randomly initialized population (RI-GA) for comparison purposes.

The maximum performance curves over 10 runs of the genetic algorithm are shown in Fig. 4. As we can see, the TR-GA significantly out-performed its

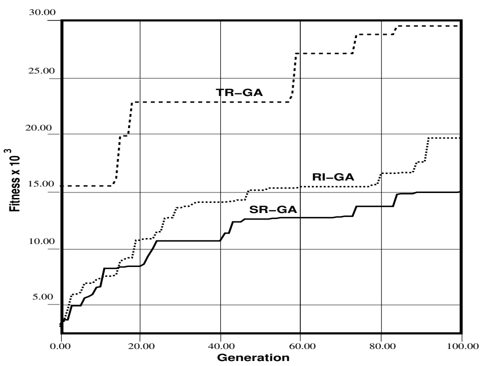

Fig. 4. The genetic algorithm maximum performance curves

competitors. Although Fig. 4 compares a TR-GA with five injected individuals to a SR-GA that used three injected individuals, the TR-GA with three injected individuals also did better than the SR-GA, while not doing as well as the TR-GA with five. Somewhat surprisingly the randomly initialized GA did better than the SR-GA, indicating that the best control strategies for basic behaviors may not contain building blocks that help navigation in the office environment and/or may be of low enough fitness to be eliminated during GA processing. More evidence is presented in Table 2 which compares the fitness, in the office environment, of cases injected into the SR-GA with those injected into the TR-GA. We also noted that solutions with high initial fitness in the target office environment may be ranked low in their source environment.

The results indicate that injecting appropriate cases into the initial population of a genetic algorithm can not only help speed up convergence but also provide better quality solutions. However, this will only work if injected solutions contain useful building blocks for solving the new problem, that is, if injected solutions are similar enough to solutions for the new problem. Assuming that problem similarity implies solution similarity is a pre-requisite for our system

Table 2. Cases used for SR-GA and TR-GA and their fitness in the office environment. FA = food approach, WF = wall following, OA = obstacle avoidance.

Case Source	Case	Fitness in Office Environment
SR-GA	Best of FA	2,807
	Best of OA	2,000
	Best of WF	1,220
TR-GA	FA-1	15,524
	WF-1	13,503
	FA-2	10,414
	WF-2	6,834
	OA-1	6,754

to perform well [6, 9], but when trying to combine several solutions, we had to re-validate this assumption by evaluating and ranking candidates for injection in the new target environment. Previous results had not indicated the need for ranking cases in the new environment before injection [6]. However, we obtained good agreement in our estimate of the number of individuals to inject. Earlier work had shown that injecting only a small percentage of the population led to good performance while injecting larger percentages led to quick convergence to a local optimum [6]. This agreed with the experimental results reported in this paper where we found that injecting five individuals (5% of the population) provided better performance compared to experiments involving the injection of a smaller or larger number of individuals.

In addition, we need to make sure that the injected individuals contain at least one representative of each basic behavior. Otherwise, the missing basic behavior may have to be evolved from scratch – from the randomly initialized component of the population. Once we have individuals representing each of the basic behaviors, the rest of the candidates for injection compete for the remaining slots on the basis of their performance in the target environment. This ensures that the population is initialized with the needed variety of high performance building blocks.

Figure 5 presents the path of a simbot, controlled by a circuit designed by the TR-GA, in the office environment. Note that although the environment contains many traps in the form of rooms with small doorways, the simbot does not get trapped and manages to avoid two unpromising rooms altogether. The TR-GA designed simbot also eats 70% of the food over ten runs compared to only 40% for the randomly initialized GA.

7 Conclusions and Future Work

The paper demonstrates that we can evolve basic behaviors and adapt to the environment using CHC, a non-traditional genetic algorithm. Injecting selected

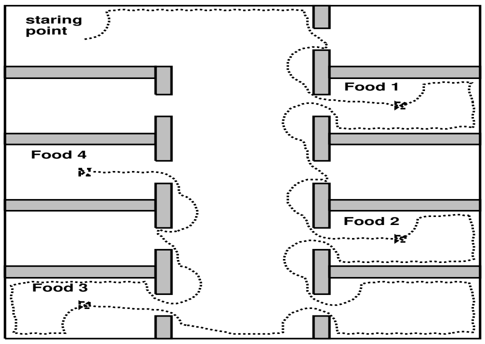

Fig. 5. Simbot path in an office environment for a circuit designed by the TR-GA

solutions stored in a long term memory and corresponding to these basic behaviors into the GA's initial population allows us to quickly and successfully design control strategies for a robot navigating in a complex office environment. The experimental results are promising and the simulated robot is faster and accomplishes more of the task than the robot designed by a randomly initialized GA. We are currently investigating parallelization of the code to handle a larger population size in a reasonable amount of time. This will allow us to handle more complex environments. We are also planning to transfer the evolved circuits to a real mobile robot, thus testing our work on physical hardware with all its concomitant problems. We will be investigating the effect of noise on performance – circuits evolved in the presence of noise may be more robust and better able to handle the noise inherent in a real mobile robot operating in a complex environment.

We have only reported on non-randomly initializing genetic algorithms in this paper. However, the concept is extendable to other population based searches like evolutionary programming, evolution strategies, and genetic programming. In addition, there is no reason why injection of individuals should only take place at initialization – we can inject individuals during the course of GA's run. We believe that investigating the combination of population-based search algorithms with a long term memory promises to be a fruitful area of future research. The hope is that as the number of problems solved by the combined system grows, the time taken to solve a new problem shrinks.

8 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 9624130.

References

1. D. Cliff, P. Husbands, and I. Harvey. Evolving visually guided robots. In *Technical Report CSRP 220*. School of Cognitive and Computing Science, University of Sussex, 1992.
2. Larry J. Eshelman. *The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination*. Morgan Kauffman, San Mateo, CA, 1990.
3. D. E. Goldberg. *Genetic Algorithm in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
4. J. Holland. *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
5. J. R. Koza. *Genetic Programming*. MIT Press, 1992.
6. Xiaohua Liu and Sushil J. Louis. Combining genetic algorithms and case-based reasoning for structure design. In M. E. Cohen and D. L. Hudson, editors, *Proceedings of the ISCA Eleventh International Conference on Computers and their Applications*, pages 103–106. ISCA, 1996.
7. Sushil Loius, Gary McGraw, and Rechard O. Wyckoff. *CBR Assisted Explanation of GA Results*. Technical Report No.361, Indiana University, 1992.
8. Sushil J. Louis and Gregory J. E. Rawlins. Designer genetic algorithms: Genetic algorithms in structure design. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 53–60. Morgan Kauffman, San Mateo, CA, 1991.
9. Sushil J. Louis and Zhijie Xu. Genetic algorithms for open-shop scheduling and re-scheduling. In M. E. Cohen and D. L. Hudson, editors, *Proceedings of the ISCA Eleventh International Conference on Computers and their Applications*, pages 99–102. ISCA, 1996.
10. Andrew Murray and Sushil J. Louis. Design strategies for evolutionary robots. In E. A. Yfantis, editor, *Proceedings of the Third Golden West International Conference on Intelligent Systems*, pages 609–616. Kluwer Academic Press, 1995.
11. C. Ramsey and J. Grefensttete. Case-based initialization of genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, San Mateo, California, 1993. Morgan Kauffman.
12. C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Cambridge, MA, 1989.