In M. Jamashidi, C.C. Nguyen, R. Lumia, and J. Yuh (Eds.), *Intelligent Automation and Soft Computing: Trends in Research, Development and Applications*, Albuquerque, NM, pp. 419–424. TSI Press.

Evolution of adaptive behavior in dynamic environments

Nikolaus Almássy and Erik Vinkhuyzen

AILab, Department of Computer Science, University Zurich-Irchel
Winterthurerstrasse 190, 8057 Zurich, Switzerland
E-Mail: almassy vinkhuyz@ifi.unizh.ch

Abstract

We use genetic algorithms to evolve behavioral properties of simulated autonomous vehicles. The traits of the robots that are being evolved include parameters and internal control mechanisms that drive the following-, approach- or avoidance behavior. A parallel steady state genetic algorithm is used to simulate natural evolution. We argue why we need evolutionary techniques to create adaptive behavior in dynamic environments.

1 Motivation

There is a growing doubt that the traditional artificial intelligence (AI) approach to intelligent behavior, which involves reasoning by manipulating symbolic representations of the world, will ever lead to systems that are interacting robustly in a real world environment (Winograd and Flores 1986; Brooks 1991). This raised a great deal of interest in the study of adaptive behavior and the construction of autonomous agents, that in contrast try to avoid engineered representations of the world and relate more to the ethology of simple animals.

To perform such studies one can either invest in suitable hardware and create control architectures for real robots or, as cheap and easy usable robots are difficult to maintain, rely on robot simulations on a computer. One might argue that simulated environments do not offer interactions that are rich enough to prove the robustness of a specific control architecture. On the other hand, testing the performance of control architectures implemented on real robots is a timely task and the automation of such tests are even more difficult than building the robot itself. Real robots have much more complex interactions, that are only possible in the real world. Hardware sensors and actuators behave in a less predictable way due to noise or hardware problems, also the morphology of the robot can influence its behavior. A control architecture that functions well in a discrete and noise-free simulation environment might fail if implemented on a robot that is equipped with noisy sensors, confronted with slippery or rough surfaces in an unpredictable environment. Such simulations of autonomous robots are then criticized for the simplicity of the possible interactions. Despite these drawbacks simulations are vital to explore unknown aspects of the control architecture.

In this work we use the BUGWORLD simulator (Almássy 1993) that makes the best possible trade-off between simple and fast simulations and a test bed that offers robot environment interactions that are rich enough such that they can be compared with real robots. Simulations always should be combined with studies on real robots to validate the results gained by simulations. The code of the robot's control architecture just has to be modified at the interface to the sensors and actuators. This technique of integrating simulations in the development of control architectures for real robots was successfully used by (Mondada and Verschure 1993).

2 Evolution of behavioral properties

As already outlined in Section 1 we believe that learning autonomous robots will exhibit a more robust behavior in dynamic environments than robots equipped with control architectures designed with traditional AI methods. On the other hand, designing autonomous systems that achieve adaptive behavior becomes overwhelmingly difficult as the number of parameters that define a particular behavior increases and the complexity of the system exceeds human engineering capacity. Therefore we propose a design system that minimizes the amount of human engineering as much as possible. We propose evolutionary techniques that make some crucial parts of the design decisions. Unlike others who used a genetic algorithm (GA) to evolve robot controllers (Vaario and Ohsuga 1992; Harvey, Husbands, and Cliff 1992), we do not start from scratch. As a starting point we use basic mechanisms described in (Verschure and Pfeifer 1992) that are extended by genetically determined behavioral traits, that are implemented by various parameters and low-level rules. Previous work in this direction (Almássy and Verschure 1992) already showed the potential of evolutionary techniques in the development of robot controllers. Genetic programming (Koza 1992), a special form of evolutionary techniques was also used by (Brooks 1992) to achieve a more robust adaptive behavior of both simulated and real robots. Others have used genetic algorithms to design autonomous mobile robots (Vaario and Ohsuga 1992; Harvey, Husbands, and Cliff 1992), but all these attempts were more concerned with the biological plausibility of their models, and did not relate enough to physical robots that can be used to accomplish certain tasks.

Of course there are other search and optimization techniques that would potentially be applicable in our case: *random search*, where points in the search space are chosen at random or relatively unsystematically and evaluated: this method would by no means guarantee any convergence to an optimal point. Gradient methods such as *hill-climbing* are well suited for unimodal functions but likely to get stuck in local optima of multimodal functions. Combinations of these methods are called *iterated search* and start again with a random point when hill-climbing reaches a local optimum. Since starting points will be distributed evenly over the search space, as many trials will be allocated in successful regions as in regions of low fitness. In contrast the GA starts with evenly distributed starting points and allocates an increasing number of trials in regions of high fitness. Another modification of hill-climbing, *simulated annealing* essentially uses one candidate solution at a time, and modifies it randomly over time. As the search proceeds, the probability that a worse solution will be accepted approaches zero. By that local optima can be avoided, but still not enough information is passed between trials to give an "overall picture" of the search space.

Specifying the fitness of an agent

The specification of the fitness of the agent is the crucial part of the design system: we do not want to specify the *right strategy* for successful behavior, but instead set up constraints the agent has to fulfill. To avoid obstacles, to never collide with other (movable) robots, and to collect targets. We will give a small penalty for touching obstacles, but a high penalty for collisions with other robots. We implement these penalties by reducing the number of steps the agent is allowed to perform. The agent's fitness is only determined by the number of targets it collected. Therefore we do not determine a specific strategy the agent has to follow. The minimal set of sensing mechanisms that are used by the agent:

- detection of static obstacles,
- detection of moving obstacles (by compensating own movement)
- sensing of the target

With these mechanisms, and a set of basic reflexes, we want to evolve a set of parameters for the robot's control architecture. The parameters influence aspects of the control architecture like the learning rate of the neural network and strategic behavioral aspects like following or retreating behaviors.

3 A distributed GA simulation

The main focus of our research is the evolution of adaptive behavior in a dynamic unpredictable environment. For that reason we use a parallel distributed simulation environment that lets us simulate a large number of vehicles (agents) in a single environment, therefore offering the possibility of interactions between agents. Since simulations of dynamic environments are computationally very expensive, we implemented a parallel simulator that runs on a local area network of high performance workstations.

The simulation task was separated into the simulator that computes sensory information and maintains the information about all defined objects, and the agent that computes the next action just based upon the sensory feedback that is received from the simulator. No other information about the simulated world is available to the agent. Separate processes define the world with its obstacles and targets and display it on the screen (See figure 1). The simulator (Almássy 1993) has been and is still successfully used in several projects (Almássy and Verschure 1992; Verschure 1992; Mondada and Verschure 1993).

The same concepts that applied to the design of the simulator were also used in the integration of the simulator with a parallel genetic algorithm (GA) (for an introduction to genetic algorithms see (Goldberg 1989; Davis 1991)). Major parts of the simulator software were reused in the implementation, and since workstations in a local area network are not as reliable as a parallel computer, mechanisms that guarantee fault tolerance and load balancing were included.

we don't have enough space so we don't take this figure.

A central process manages a sorted list of abstract descriptions of agent properties (genotypes) selects these (with a linear selection bias towards the best performing) for reproduction and distributes them to the processing elements (the agent process). These agents are then tested in their simulation environments, and report then the performance back to the GA. As soon as a processing element (PE) is finished with the evaluation of a genotype, two new genotypes

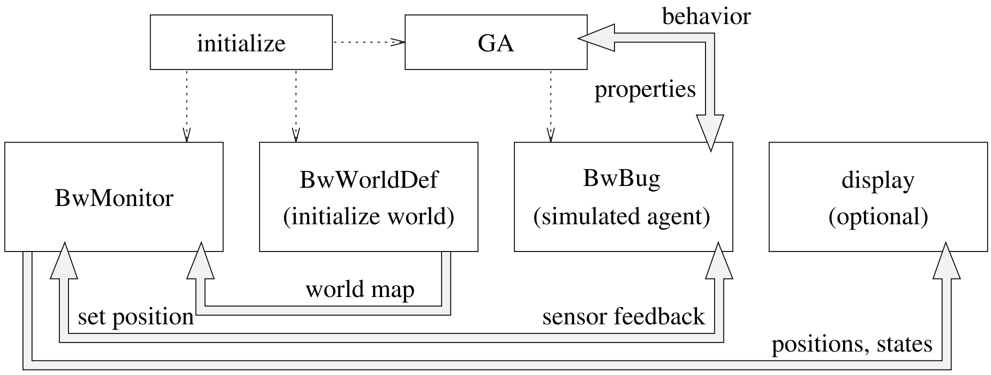

Figure 1: The simulation begins by starting *BwMonitor*. Then, *BwWorldDef* connects to *BwMonitor* to define obstacles and targets. The display can be attached to the *BwMonitor* any time, it opens up a window and constantly maintains the current state of the world with agents, targets and obstacles. Finally agents (*BwBug*) can connect to *BwMonitor* and typically perform the following steps: requesting range-finder and target feedback, computing the next action based on this information, and communicating its new position to *BwMonitor*. Once the simulation is started up it is connected with a genetic algorithm (GA). The GA process sends parameters to the agent to determine behavioral properties. The agent reports the number of targets found after the allotted number of steps to the GA, that uses this information to compute the fitness. The thick arrows represent a communication link and the dotted arrows stand for process calls.

(individuals) are selected from the pool of genotypes, crossed over with probability $p_{cross} = 0.8$ and mutated with probability $p_{mut} = 0.04$, and one of these genotypes is redistributed to the idle PE. Therefore all PEs are always busy, and no time is waisted by waiting for a resynchronization necessary for a generational step. Such a genetic algorithm is called a *steady state* algorithm (Whitley 1989), to be distinguished from a canonical GA (Goldberg 1989), that needs to know the mean fitness of the whole genotype pool (i.e. the current generation) to determine the selection function for the next generation.

The reasons for choosing a steady state GA were (1) its superiority upon canonical GAs, (2) no resynchronization is needed when a generation is finished, and (3) the closer resemblance to natural evolution: no resynchronizations or engineered mechanisms that control selective pressure and population diversity are needed. Steady state GAs can be used to speed up a robust and focused evolutionary search (Whitley 1989).

Integration with a real environment

To verify the control architectures, a robot interface will be built, that allows the execution of the same code of the control architectures on a real robot with hardware sensors and actuators. The agent's process connects to that interface that translates the movement actions given by the control architecture into motor signals of the robot. The sensor data from the robot is scaled to match the range of the simulated range finder. The interface process, is connected with a cable to the robot which can be easily included in a real robot GA experiment.

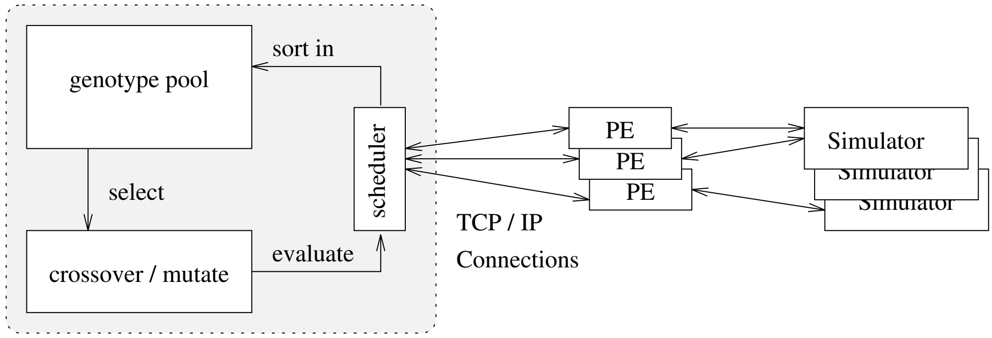

Figure 2: The genetic algorithm maintains a sorted list of genotypes, selects these for reproduction, variates the genotypes and passes them on to the scheduler. The scheduler assigns the genotype to the first free processing element (PE) and sends a message containing the genotype to the PE that usually runs on a different computer connected via internet. As soon as the PE finished the evaluation the genotype is sorted by its fitness back into the genotype pool.

4 Design of the control architecture

The control architecture is the program that determines the movements of the simulated robot from the sensory input. Every simulated robot receives a range finder input on 37 predefined angles with respect to the direction of movement. The robot also has two target sensors, located at $\pm 90°$ from the front of the robot. The robot can only sense the target if it is near enough (a parameter typically set so the simulated robot can sense the target from about a quarter of the total distance of the world). These sensors are connected to a neural network very similar to the one used in (Verschure, Kröse, and Pfeifer 1992). We added a motion detector that detects other agents in the environment. This motion detector is a virtual sensor that predicts the range finder input based upon the previous two range finder inputs and the last step. The motion detector units are connected to the motor units 2 and 3, which stand for approach right and left respectively. The collision units, that are activated when a collision happens, are connected to the motor units 0 and 1, which stand for avoid right and avoid left respectively. The unit I_1 inhibits the output of the target units to avoid an approach to a target that is behind an obstacle. I_2 inhibits the following behavior, if a collision occurred. Note that collision units are activated either by a collision or by the range finder units after hebbian learning of the connections from the range finder units to the collision sensor units. Therefore the agent learns to associate range finder output with collisions. I_3 inhibits following behavior in the presence of targets.

Behaviors

The behavior of the simulated robot is determined in the following way: whenever it has a collision it performs an avoidance reflex and turns away from the object, typically following the wall. The reflex is implemented by not modifiable connections between collision sensor and motor units.

When the bug senses the target, it will turn in that direction. Whenever a target has been consumed, it disappears from the world and reappears at another randomly chosen location. To avoid contradictions in action behavior, avoiding obstacles is more important than finding targets. In fact, the target seeking behavior is suppressed for an extended period of time after the simulated robot has had a collision, to avoid local minima in its behavior, in which the simulated robot would, for example, turn to the right and go forward to aim at a target, but then immediately retreat because of a collision and veer to the left. This decaying inhibitory connection between target seeking and obstacle avoidance, was also used by (Verschure 1992), and implements some kind of "frustration" behavior.

Additionally we have implemented a behavior that reacts to moving objects in the environment. In our simulated world, there are only static obstacles, so anything that moves must be another robot. Whenever a robot sees another robot, it executes one of two behaviors: it either turns towards the moving agent and starts to follow it, or it ignores it. One of the questions that this simulation experiment is trying to answer, is what strategy of behavior is optimal given a certain environment with a particular number of objects and food-sources. We believe this question to be interesting for biologists, who have theories of the advantage of group behavior based on the food-distribution in the world. For example, if food sources are spread out and come in small patches, it is probably advantageous to follow other moving agents. We hope to show that agents that have a solitary strategy (i.e. they do not follow others), will survive better that those that have a group-strategy.

5 Experiments

In the first experiment we used five targets that disappeared when an agent consumed it. Five agents were started at random initial positions. We evolved the "visual width" of the agent, the range within which other agents must be in order to be recognized. Furthermore we evolved the learning rate of the range finder and the collision detector. The performance was defined as the number of targets the agents collected. Targets were either consumed (implying a change of location) or not consumed – the agent had to search for a different target. Since in one experiment the targets disappeared after an agent consumed it, following other bugs would not be a successful strategy, since those followers would always be too late. Moreover, if agents follow other agents, collisions with other agents become more likely. The result of the first experiment did in fact show an evolution of "short sightedness".

In another experiment the penalty for colliding with other agents was set to a tenth of the penalty for colliding with obstacles. The GA quickly evolved a behavioral strategy to follow other agents as close as possible. A bug would search for a target and the followers would safely reach the target just bumping against other agents. No significant improvement in collecting targets was observed.

6 Conclusion

In the experiments we conducted we showed that the optimal behavior of an autonomous agent depends on the environment. The parameter settings are difficult to determine and our results show that they are sometimes even counterintuitive. With a genetic algorithm, that works with a large database of solutions in parallel, the whole space of possible parameter settings can be explored effectively, even in the presence of noise. Due to the dynamics of the system environment interaction, the fitness of parameter settings are very dependent on the behavior of the other agents that co-evolved in the same environment. This appears to the optimizer as noise, but with the large number of steps and a parallel genetic algorithm which exploits the CPU power of many workstations we successfully reduced the noise in the fitness landscape.

Future work

The optimal behavior of an agent depends on the environment and the behavior of the other agents in the same environment. In future experiments we will inspect the different behavioral strategies that emerge from the agent environment interaction. We will also include other types of agent environment interactions what will create more complex types of interactions:

- we approach and consume the target only if we gain (more) energy.
- frustration threshold will be dependent on the energy level.
- anticipation of other agent behavior, and adaptation to these agents.

References

Almássy, Nikolaus (1993). BUGWORLD: A distributed environment for the development of control-architectures in multi-agent worlds. technical report 93-32, Department of Computer Science, University Zurich.

Almássy, Nikolaus and Paul Verschure (1992). Optimizing self-organizing control architectures with genetic algorithms: the interaction between natural selection and ontogenesis. In *Proceedings of the Second Conference on Parallel Problem Solving from Nature*, pp. 451–460. Elsevier.

Brooks, Rodney A. (1991). Intelligence without representation. *Artificial Intelligence*, **47**, 139–159.

Brooks, Rodney A. (1992). Artificial life and real robots. In *Proceedings of the First European Conference on Artificial Life*, pp. 3–10. MIT Press.

Davis, Lawrence (Ed.) (1991). *Handbook of Genetic Algorithms*, New York. Van Nostrand Reinhold.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

Harvey, Inman, Philip Husbands, and Dave Cliff (1992). Issues in evolutionary robotics. In (Meyer, Roitblat, and Wilson 1992), pp. 364–373.

Koza, John R. (1992). *Genetic programming: on the programming of computers by means of natural selection.* Cambridge, Mass.: MIT Press.

Meyer, Jean-Arcady, Herbert L. Roitblat, and Stewart W. Wilson (Eds.) (1992). *From Animals to Animats 2: Proceedings of the 2nd International Conference on Simulation of Adaptive Bahavior.* MIT Press.

Mondada, Francesco and Paul Verschure (1993). Modeling system-environment interaction: The complementary roles of simulation and real world artifacts. In *Proceedings of the ECAL 93: Self organization & life: from simple rules to global complexity*, pp. 808–817. Université Libre de Bruxelles.

Vaario, Jari and Setsuo Ohsuga (1992). An emergent construction of adaptive neural architectures. *Heuristics – the journal of knowledge engineering*.

Verschure, Paul (1992). *Modeling adaptive behavior: the dynamics of system environment interaction.* Ph. D. thesis, University of Zurich, Switzerland.

Verschure, Paul, Ben Kröse, and Rolf Pfeifer (1992). Distributed adaptive control, the self organization of structured behavior. In *Robotics and Autonomous Systems*. Elsevier.

Verschure, Paul and Rolf Pfeifer (1992). Categorization, representations, and the dynamics of system-environment interaction: A case study in autonomous agents. In (Meyer, Roitblat, and Wilson 1992), pp. 210–217.

Whitley, Darrel (1989). The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 116–121. Morgan Kaufmann.

Winograd, Terry and Carlos Fernando Flores (1986). *Understanding computers and cognition: a new foundation for design*. Norwood, N.J.: Ablex.

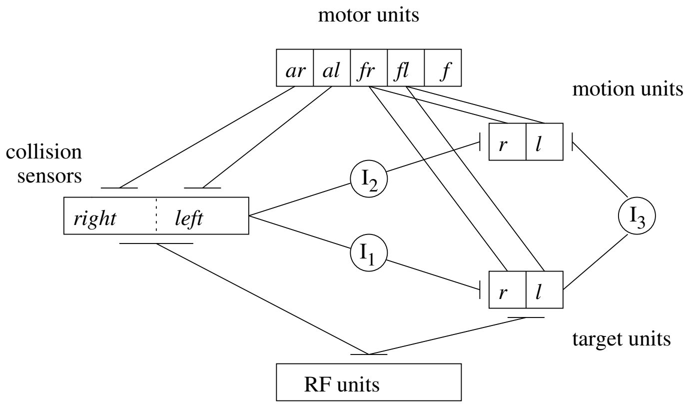

Figure 3: The control architecture of the Robot. All range finder units are connected with all collision sensor (CS) units. The left and the right half of the CS units are connected to the motor unit a_l and a_r respectively. The units I_1, I_2 and I_3 inhibit the output of the units they are connected to (depicted by the solid line at the end of the connection).

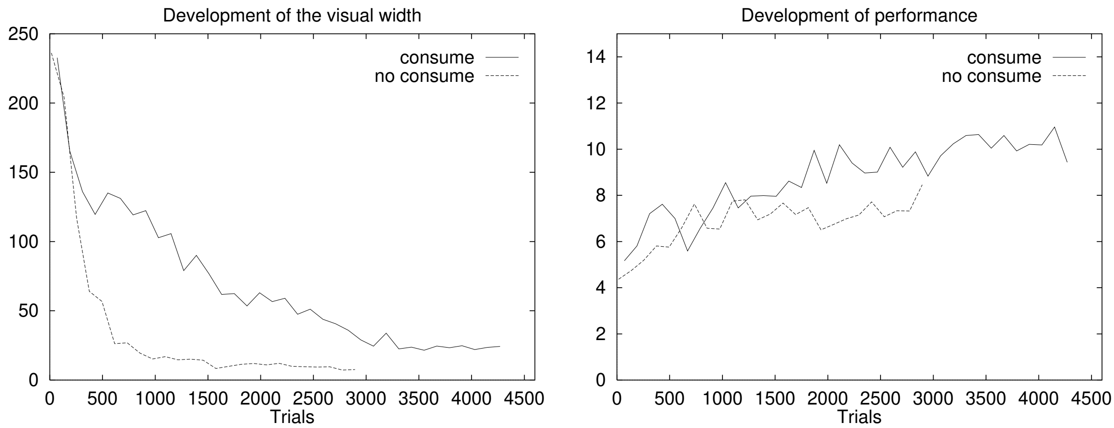

Figure 4: Development of the visual width of the agents during evolution. In one experiment the agents consumed the targets, that reappeared in randomly chosen positions in the world. Due to the design of the environment, the targets were unlikely to be located near obstacles. Therefore less bug to bug collisions happened near targets and the selective pressure towards ignoring other bugs was weaker than in the other experiment, where targets remained on the same place after an agent reached it. In both experiments the performance of the agents could be enhanced.