EVOLUTIONARY ALGORITHMS IN ROBOTICS[†]

JOHN GREFENSTETTE

Naval Research Laboratory

Washington, DC 20375-5337

ABSTRACT

Evolutionary algorithms incorporate principles from biological population genetics to perform search, optimization, and learning. This article discusses issues arising in the application of evolutionary algorithms to problems in robotics.

INTRODUCTION

The field of robotics offers an endless supply of difficult problems, requiring an equally impressive array of methods for their solution. One class of methods that has shown its utility on a number of relevant problems is called *Evolutionary Computation* [1]. This term applies to computational methods that incorporate principles from biological population genetics to perform search, optimization, and machine learning, and includes a variety of specific formulations with names such as genetic algorithms, evolutionary programming, evolution strategies, and genetic programming. This article will present an overview of evolutionary algorithms, and discuss selected issues concerning their application to robotics.

OVERVIEW OF EVOLUTIONARY ALGORITHMS

Traditional search methods typically start with one candidate solution to a problem and modify it to obtain better and better solutions. In contrast, evolutionary algorithms maintain a *population* of candidate solutions that evolves over time through a process of competition (Fig. 1). What it means to be a "candidate solution" is a function of the application task. Evolutionary algorithms have been designed in which each candidate solution represents a collection of parameter settings [2], an individual rule [3], a collection of rules [4], or a tree-structured computer program [5]. In each case, the algorithm proceeds by evaluating the *fitness* of the candidate solutions in the population, and then selecting parents from the current population for replication or deletion on the basis of fitness. Next, plausible new solutions (*offspring*) are constructed by applying idealized *genetic search operators* (such as *recombination* and *mutation*) to the selected parents. This cycle repeats, producing generally more fit candidates over time. The details of the individual phases of the evolutionary algorithm vary widely among research groups, but comparative studies are beginning to identify some fundamental properties of the alternative approaches [2].

† To appear in the Fifth International Symposium on Robotics and Manufacturing, ISRAM 94.

```
procedure *Evolve*
begin
    t = 0;
    *initialize* population P(t);
    *evaluate* solutions in P(t);
    while an acceptable solution has not been found, do
    begin
        t = t + 1;
        *select parents* from P(t-1);
        *recombine* and *mutate* parent solutions;
        *evaluate* the new solutions;
        *update* population P(t) with new solutions;
    end
end.
```

FIG. 1. An Evolutionary Algorithm.

It should be clear from the above description that evolutionary algorithms represent a very general class of search and learning methods. To apply the approach to a particular problem, the user must specify:

1) A representation of the space of candidate solutions;

2) A method for evaluating any candidate solution in the space;

3) A set of operators that modify and recombine candidate solutions.

In many cases, it requires considerable effort to properly specify these components. For example, the operators used to modify and recombine candidate solutions must be well-suited to the particular representation used. Fortunately, in addition to the extensive experience with evolutionary algorithms of the last twenty-five years [6,7,8], there is a growing body of theory that can guide the user toward successful applications [9]. Evolutionary algorithms employ an intelligent random search in the space of candidate solutions. While cost-effective on difficult problems, they are not the method of choice for problems that can be solved easily through analysis, simple heuristic search, or exhaustive enumeration [10].

ROBOTICS APPLICATIONS

Evolutionary methods have found applications that span the range of architectures for intelligent robotics. For example, evolutionary algorithms have been used to learn rule sets for rule-based autonomous agents [4], topologies and weights for neural nets for robotic control [11,12], fuzzy logic control systems [13], programs for LISP-controlled robots [5], and rules for behavior-based robots [14]. There are at least two different research paradigms evident in the literature of evolutionary algorithms for robotics, which might be called the *artificial life (ALife)* paradigm, and the *genetic knowledge engineering* paradigm.

The ALife paradigm is motivated by a number of issues. First, there is interest in the process of evolution itself. Simulations of robotic agents can be used to study the emergence of complex behavior from the interaction of simple components, operating under conditions of competition for resources [15]. Second, there are concerns about the utility of the traditional model-based reasoning approach to designing intelligent robots. Models used in artificial intelligence studies often fail to reflect the complexities, noise, and errors that arise in real sensors and actuators operating in the real world. Furthermore, traditional AI research usually assumes that the problem-solving module have inputs that have already been correctly translated from analog signals (e.g., sonar, infra-red, etc.) to symbols (e.g., "door-is-open"). In response to such shortcomings, some researchers argue for the development of adaptive robots that evolve behaviors without using a pre-specified model of the world [16]. Finally, there is a growing interest in the possibility of physical robots based on nano-technology, that truly self-replicate and evolve in adaptation to their environment [17].

The genetic knowledge engineering paradigm treats the knowledge acquisition task for intelligent robots as a cooperative effort between the robot designers and the robot itself [18]. Some aspects of the robot's world will be known in great detail to the designer, for example, the size and weight of the robot, the characteristics of its sensors and effectors, and at least some of the physics of the task environment. The robot should have access to the best model of its world that the designer can reasonably provide, possibly including a quantitative simulation model of the robot and its environment. On the other hand, some aspects of the environment will be unknown in advance. These might include the behavior of other agents, the exact physical properties of the environment (e.g., terrain, specularity of walls, etc.), as well as knowledge of the most effective behaviors for performing specific tasks. In this paradigm evolutionary algorithms can be used to explore alternative robot behaviors within a simulation model as a way of reducing the overall knowledge engineering effort. As simulation technology continues to improve, and as computational power to run simulations becomes more available, objections that simulations always include too many unrealistic assumptions will carry less weight. Machine learning methods, such as evolutionary computation, that exploit the power of sophisticated simulations are therefore likely to become increasingly important.

Both of the above paradigms may be expected to provide fundamental insights into the development of flexible and adaptive robots. The remainder of this paper will address some particular issues applying the genetic knowledge engineering approach to problems in robotics, focusing in part on the SAMUEL evolutionary learning system being developed at NRL [4].

USING BACKGROUND KNOWLEDGE

It is natural to classify evolutionary algorithms as "weak methods", that is, methods that rely on few assumptions about the space being searched. However, the approach affords several opportunities for using background knowledge to augment the basic method [19], producing a more efficient overall knowledge acquisition effort.

Choice of Search Space and Representation. The first choice the user of an evolutionary algorithm needs to make is the choice of the search space, along with an appropriate representation. This choice clearly reflects the trade-offs between human and machine effort. For example, if the choice is to search the space of all mappings between the set of all possible sensor readings (digitized to greatest possible resolution) to the set of all possible actuator commands, this clearly places the entire burden of knowledge acquisition on the learning system. A more reasonable trade-off might be to process the raw sensor readings into a set of virtual sensors and to provide a set of behaviors for the robot, leaving the task of learning the mappings between these much more constrained sets. If the designer can specify the desired behavior to within a set of numeric parameters, then evolutionary algorithms can be used to search the space of parameters for the most desirable values.

As an example, in the SAMUEL learning system, candidate solutions in the evolutionary learning system are sets of rules of the form:

```
if (forward-sonar < 50) and (left-sonar > 75)
then set turn = left-45-degrees
```

Each such rule plays the role of a *gene,* with a set of such rules, or *strategy,* treated as a candidate solution to the problem (navigation in this case). In SAMUEL, the user can further limit the search space by defining a set of *constraints* in the form of rules that specify conditions under which certain actions are either forbidden or required [20]. Constraints are intended to limit the robot's actions within physically safe parameters, but still allow freedom to explore a large set of alternative strategies [21].

Initial Population. Evolutionary algorithms often begin with candidate solutions selected at random from the search space. Often, the approach can be sped up by the use of heuristics to select the starting population. This must be done with care, however, since a lack of sufficient diversity in the initial population is almost guaranteed to produced premature convergence to suboptimal solutions.

In SAMUEL, the rule representation was designed to encourage the user to include heuristic strategies in the initial population [22]. In fact, for many complex robotic tasks, it is unlikely that the system will be able to evolve solutions without some initial heuristics. For example, if the task is to track a moving target, the robot is unlikely to perform the task at all given a set of random rules of the form shown above. We are exploring several approaches for exploiting heuristic knowledge, including the automatic generation of several variants from a user-generated rule set. Again, this raises the issue of trade-offs between the effort of specifying good initial rules and the effort of engineering the search space to include only plausible solutions to the task. The optimal trade-off will vary from task to task.

Fitness Function. There are a few important requirements for the design of the fitness function in evolutionary algorithms. We mention the two most prominent ones: First, the fitness function should accurately reflect the desired performance on the task. Evolutionary algorithms are highly opportunistic optimizers and may produce surprising results if the fitness function rewards some behavior that the system designer does not want. Evolutionary algorithms may also exploit unexpected features of the simulation model to maximize performance. Of course, this implies that the model should accurately reflect the conditions in the target environment. To the extent that this is not possible, our studies [23] have shown that it is still possible to learn from limited-fidelity simulations that err on the side of difficulty (e.g., have more noisy sensors that the real robots). In such cases, the learning time increases, but so does the robustness of the learned rules. Second, the fitness function should provide differential payoff so that alternative candidate solutions can be ranked. It should not present a "needle-in-a-haystack" problem in which only the final solution is assigned a positive fitness.

Genetic Operators. Many early studies of genetic algorithms employed simple syntactic operators to alter candidate solutions, such as simple crossover and random mutation. However, some recent studies have used more heuristic operators that make more directed changes based on the learning agent's experience. For example, some genetic classifier systems use *triggered operators* such a creating a new rule to cover a novel situation [3]. In SAMUEL, we use *generalization* and *specialization* operators that are triggered by specific conditions relating the measured utilities of individual rules and the outcome of the task. These may be viewed as Lamarckian forms of evolution [24], and show that artificial evolution need not proceed along purely Darwinian lines.

Hybrid Approaches. Finally, it is often useful to use evolutionary methods in concert with other methods that have complementary strengths. The strength of an evolutionary algorithm is in rapidly identifying the most promising regions of a complex search space. Evolutionary methods are less efficient at fine-tuning candidate solutions. Therefore, a natural hybrid is to use an efficient local optimization method to improve the final solutions found by the evolutionary system [19].

Another hybrid approach is to augment the population with additional long-term memory, to deal with changing environments. We are exploring an approach called *case-based anytime learning* [25]. In this approach, the robot's evolutionary learning module continuously tests new strategies against a simulation model of the task environment, and dynamically updates the knowledge base used by the real robot on the basis of the results. The robot's online execution module includes a monitor that can dynamically modify the simulation model based on its observations of the changing environment. Currently, the system can modify the simulation by altering several parameters according to their observed value in the real world. When the simulation model is modified, the learning process is reinitialized using previously learned strategies from similar cases. Tests using a two-robot cat-and-mouse game as the task environment show that case-based methods provide significant improvement in responding to changes in the task environment [26].

FUTURE DIRECTIONS

We have presented a brief overview of issues arising in the application of evolutionary algorithms to robotics. Within the genetic knowledge engineering approach, many open problems remain, including how to evolve hierarchies of skills and how to enable the robot to evolve new fitness functions as the need for new skills arises. Perhaps the work on ALife approaches will produce new insights for these difficult problems.

REFERENCES

1. De Jong, K. A. and W. Spears (1993). On the state of evolutionary computation. *Proceedings of the Fifth International Conference on Genetic Algorithms,* pp. 618-626, San Mateo, CA: Morgan Kaufmann.
2. Baeck, T. and H.-P. Schwefel (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation 1(1),* 1-23.
3. Booker, L. B. (1988). Classifier systems that learn internal world models. *Machine Learning 3(3),* 161-192.

4. Grefenstette, J. J., C. L. Ramsey and A. C. Schultz (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning 5(4),* 355-381.

5. Koza, J. R. (1992). *Genetic Programming.* Cambridge, MA: MIT Press.

6. Forrest, S. (Ed.) *Proceedings of the Fifth International Conference on Genetic Algorithms,* San Mateo, CA: Morgan Kaufmann, 1993.

7. R. Maenner and B. Manderick (Eds.), *Proceedings of Parallel Problem Solving from Nature-2.* North-Holland, 1992.

8. Fogel, D. B. and W. Atmar, *Proc. Second Annual Conference on Evolutionary Programming.* La Jolla, CA: Evolutionary Programming Society.

9. D. Whitley (Ed.), *Foundations of Genetic Algorithms-2.* San Mateo: Morgan Kaufmann, 1993.

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

11. Whitley, D., S. Dominic, R. Das, C. Anderson (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning 13(2/3),* 259-284.

12. Yamauchi, B. (1994). Dynamic neural networks for mobile robot control. *ISRAM 94.*

13. Karr, C, L. (1991). Design of an adaptive fuzzy logic controller using a genetic algorithm. *Proceedings of the Fourth International Conference on Genetic Algorithms,* pp. 450-457, San Mateo, CA: Morgan Kaufmann.

14. Dorigo, M. and U. Schnepf (1993). Genetics-based machine learning and behavior-based robotics: a new synthesis. *IEEE Transactions on Systems, Man and Cybernetics, SMC-23, 1.*

15. P. Todd and S. Wilson (1993). Environment structure and adaptive behavior from the ground up. *From Animals to Animats 2: Proc. of the Second International Conference on Simulation of Adaptive Behavior,* pp. 11-20. Cambridge, MA: MIT Press.

16. Brooks, R. *Artificial Life and Real Robots.* Cambridge: MIT Press. 1992.

17. Higuchi, T., T. Niwa, T. Tanaka, H. Iba, H. de Garis and T. Furaya (1993). Evolving hardware with genetic learning: a first step towards building a Darwin machine, *From Animals to Animats 2: Proc. of the Second International Conference on Simulation of Adaptive Behavior,* pp. 417-424. Cambridge, MA: MIT Press.

18. Grefenstette, J. and C. Ramsey (1993). Combining experience with quantitative models. Workshop on Learning Action Models at AAAI-93 (National Conference on Artificial Intelligence, 1993)

19. Grefenstette, J. Incorporating problem specific knowledge into genetic algorithms, in *Genetic Algorithms and Simulated Annealing,* L. Davis (Ed.), London: Pitman, 1987.

20. Grefenstette, J. J. and H. C. Cobb (1994). User's guide for SAMUEL. Version 4.0. NRL Report, Naval Research Lab, Washington, DC.

21. Grefenstette, J. (1992). The evolution of strategies for multi-agent environments. *Adaptive Behavior 1(1),* 65-90.

22. Schultz, A. C. and J. J. Grefenstette (1990). Improving tactical plans with genetic algorithms. *Proceedings of IEEE Conference on Tools for AI 90* (pp 328-334). Washington, DC: IEEE.

23. Ramsey, C. L., A. C. Schultz and J. J. Grefenstette (1990). Simulation-assisted learning by competition: Effects of noise differences between training model and target environment. *Proceedings of the Seventh International Conference on Machine Learning* pp. 211-215. San Mateo, CA: Morgan Kaufmann.

24. Grefenstette, J. J. (1991). Lamarckian learning in multi-agent environments. *Proceedings of the Fourth International Conference of Genetic Algorithms* pp. 303-310, San Mateo, CA: Morgan Kaufmann.

25. Grefenstette, J. J. and C. L. Ramsey (1992). An approach to anytime learning. *Proceedings of the Ninth International Conference on Machine Learning* pp. 189-195, D. Sleeman and P. Edwards (eds.), San Mateo, CA: Morgan Kaufmann.

26. Ramsey, C. L. and J. J. Grefenstette (1993). Case-based initialization of genetic algorithms. *Proc. Fifth Int. Conf. on Genetic Algorithms.* pp. 84-91, San Mateo, CA: Morgan Kaufmann.