# MEMETIC LEARNING: A NOVEL LEARNING METHOD FOR MULTI-ROBOT SYSTEMS

Dean F. Hougen, Joseph Carmer, and Mark Woehrer

*Robotic Intelligence and Machine Learning Laboratory, School of Computer Science*

*University of Oklahoma, Norman, OK 73019*[*]

{hougen,jcarmer,markwoehrer}@ou.edu

**Abstract** We need to find innovative ways of expanding the activity repertoire of robots, including having robots learn their own activities. Here, we explore the suitability of well-known methods for learning with multiple robots and introduce a new method: memetic learning. Memetic learning combines individual experience methods with population-based methods in a novel way that allows for expanding or changing candidate solutions in non-random ways, based on evaluations of strengths and weaknesses of current solutions. This results in an intelligent search of the feature-space and, therefore, requires fewer learning trials. This is particularly important if online learning is to be achieved.

**Keywords:** Robot learning, reinforcement learning, eligibility traces, evolutionary computation, genetic algorithms, memes

## 1. Introduction

As multi-robot systems become more widespread, we need to find innovative ways of expanding the activity repertoire of these robots, including having the robots learn their own activities. Possible learning methods include those based on the experiences of individual robots, population-based methods that use analogies to evolution, and methods that combine these concepts.

The suitability of these methods for learning in robotic systems depends critically on the amount of time that the robots must spend gathering experiences in order for learning to take place. In both real robots and high-fidelity simulations, this factor will almost surely be the limiting factor in learning. It

[*]Support for the authors has come from the School of Computer Science, the College of Engineering, the University of Oklahoma, and the National Science Foundation.

makes sense, then, to explore learning methods that attempt to make the most of any experience gained.

For this reason, we introduce a new learning paradigm for robotic systems: memetic learning. Memetic learning combines individual experience methods with population-based methods in a novel way that allows for expanding or changing candidate solutions in non-random ways, based on evaluations of strengths and weaknesses of current solutions. This results in an intelligent search of the feature-space and, therefore, requires fewer learning trials. Not only can activities be learned memetically in a multi-robot environment but, unlike many learning methods (such as individual reinforcement learning), memetic learning benefits from the use of multiple robots.

## 2. Memetic Learning

Memetic learning is based on imitation—people learn, not only from their own direct experiences with the world, but also by following patterns, models, or examples they have observed. This imitation is an active process by which people acquire the components of cultural knowledge. This process provides a rich metaphor on which several related machine-learning methods can be based. These methods will be known collectively as memetic learning.

### 2.1 Related Work

Several related approaches, all falling under the heading of *evolutionary computation*, have been proposed and are the subjects of much active research in machine learning (Back et al., 1997; Yao, 1999). These approaches are based loosely on concepts borrowed from biological evolution. Of these approaches, the best known is *genetic algorithms*. In genetic algorithms, chromosomes (sequences of genes) are represented by sequences of discrete values, typically binary strings. The possible values for each gene at each *locus* (position) on a chromosome are the *alleles*.

Genetic algorithms start with an initial (randomly-determined) population of proposed solutions to some problem and use analogs of the gene-level operations of crossover and mutation to add new candidate solutions to the population. A selection method preserves some solutions and removes others, according to some fitness criteria. Genetic algorithms, therefore, may serve as parallel, heuristic search procedures (Holland, 1962). The application of evolutionary computation methods to learning action policies for robotics and similar applications has been reviewed recently (Moriarty et al., 1999).

While genetic algorithms and other evolutionary computation approaches are able to find solutions to difficult machine-learning problems, they are inefficient in their search. This is because, like biological evolution, standard evolutionary computation approaches are blind. That is, their methods for generating new candidate solutions are random.[1]

Non-random approaches to improving candidate solutions have been proposed, such as *reinforcement learning* (Kaelbling et al., 1996; Sutton and Barto, 1998).[2] In reinforcement-learning methods, an initially random candidate solution is created and tested. If it performs well, it is rewarded; if not, it is punished. Based on its performance, the candidate solution is modified and then re-tested. Unfortunately, working from a single individual candidate solution makes it likely that large parts of the feature space may not be explored unless the modifications are largely random, despite the rewards and punishments received. This is the problem of exploitation vs. exploration in reinforcement learning—if you exploit what you know to get the most reward of the known possibilities, you may be failing to find better possibilities that could be located through exploration.

To overcome the problems with blindness in evolutionary computation approaches and the lack of feature space coverage found in directed approaches, combinations of these approaches have been proposed. This effort has resulted in *evolutionary reinforcement learning* and *learning classifier systems* (Lanzi et al., 2000). Here individuals within the population learn individually, then share their knowledge through random combination methods such as crossover.

Moscato (1989) proposed an approach he called *memetic algorithms*. Memetic algorithms are evolutionary computation methods. However, whereas other evolutionary computation approaches are inspired by biological evolution and its combination of replicators known as genes, memetic algorithms are inspired instead by the evolution of cultural knowledge and are named for the cultural replicators that Dawkins proposed and named *memes* (Dawkins, 1976).

Moscato recognized that there are at least two great differences between biological evolution and cultural evolution: (1) individuals cannot choose their own genes whereas memes can be acquired intentionally, and (2) individuals may modify and improve upon the memes that they acquire, whereas they cannot do so with their genes.

Despite recognizing both of these differences, Moscato only incorporated modification of acquired memes into his memetic algorithms; meme selection was left as future work. Subsequent researchers have followed suit and memetic algorithms have largely become synonymous with the combination of local search heuristics with genetic algorithms using crossover. They are, therefore, sometimes called *hybrid genetic algorithms* or *genetic local search* methods and are closely related to evolutionary reinforcement learning and learning classifier systems.

## 2.2 Proposed Approach

This paper introduces the idea of *memetic learning*. Memetic learning is directly inspired by Dawkins notion of the acquisition of memes through *imitation*. What is needed for *intelligent* imitation is a method to evaluate partial solutions—either individual genes or groups of genes in combination. This evaluation is what is provided by memetic learning and is what differentiates memetic learning approaches from other evolutionary computation approaches.

For robotic tasks, the genes may represent actions to be taken given the state of the system. These state-action pairs can be evaluated using *reinforcement learning* (Kaelbling et al., 1996; Sutton and Barto, 1998). Memetic learning applied to such tasks, therefore, can be seen to combine the best of both reinforcement learning and evolutionary computation. Note that this is different from the way in which these methods are combined in evolutionary reinforcement learning (or learning classifier systems) in which the learning is done by the individual based on its *own* experience, whereas with memetic learning, the individual may look at the reward received by *other* individuals for their actions in the part of the search space in question, to determine which to imitate. Here we introduce one way of doing this, which we call *splicing*.

**2.2.1 Imitation by Splicing.** Splicing involves taking partial solutions from two or more individuals and combining them into a single individual. This operation is similar to crossover in standard genetic algorithms and other evolutionary computation methods but with a great advantage: The replicator sequence is chosen based on its estimated positive contribution to the donor individual rather than selected randomly. This means that it is likely to have a similar positive contribution to the receiving individual. This also means an individual may receive replicator sequences from more than two donors at once. (While this could be done with standard genetic algorithms, the random nature of the selection process in a standard genetic algorithm makes it decreasingly likely that the receiving individual will acquire beneficial components from a donor as the number of donors increases.)

**2.2.2 Memetic Learning Algorithms.** *Memetic learning algorithms* fit Moscato's definition of memetic algorithms. However, we use the name memetic learning algorithms to stress the use of non-random methods for the generation of new solutions, rather than crossover which is so widely associated with previous memetic algorithms.

Memetic learning algorithms are related to genetic algorithms in the form that solutions may take. As with standard genetic algorithms, one defines possible solutions as sequences of discrete values, typically binary strings. For genetic algorithms, each entry in the sequence is considered a gene, whereas with memetic learning algorithms, each entry in the sequence is a meme. The possible values for that entry are the alleles.

Learning in memetic learning algorithms proceeds as follows: A population of individuals with random alleles for each meme is constructed and tested on the task. Individuals observe their own overall fitness values and those of the other individuals in the population. Further they observe the partial fitness values of the partial candidate solutions that they are able to identify, both for themselves and for the others. They then replace their memes for those portions of the solution for which they have low fitness values, using imitation.

## 3. Experiments using Trailer-Backing

As our first test of memetic learning algorithms, we use the task of *trailer-backing*. The object of this task is for the robotic agent to learn to back a truck with a hinged trailer to a stationary goal, as shown in Figure 1. The agent fails if the *hitch angle* (the angle between the spines of the truck and the trailer) exceeds 90° in either direction—known as *jackknifing*—or the *goal angle* (the angle between the spine of the trailer and the goal) exceeds 90° in either direction. The agent succeeds if the rear of the trailer reaches the goal. The agent controls the angle of the steering wheels at the front of the truck.

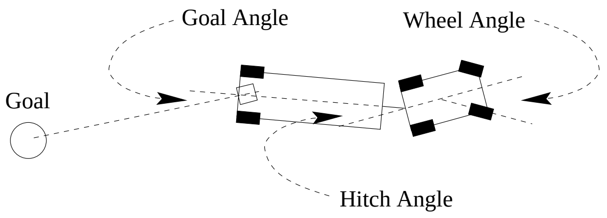

*Figure 1* Our version of the trailer-backing task.

This is a difficult task in part because the agent is provided with very little information from which to learn. It is given current state information on every discrete time-step but the only feedback given is a binary success or failure signal that may come after dozens or hundreds of time steps. This provides a *temporal credit assignment* problem. Moreover, our version of the task introduces severe *perceptual aliasing* of the kind found in many robot learning tasks. Rather than providing the agent with knowledge of the position and orientation of the truck and trailer in global coordinates, as other authors have done (e.g., Koza, 1992), we provide the agent with only information that could be easily sensed by a robotic truck-trailer system. In particular, we give the agent only the goal and hitch angles described above. Significantly missing is any information on distance to the goal. This means that the same decision for the same input is unlikely to produce the same next input.

## 3.1 Learning Algorithms

We compare three basic types of learning algorithms. To exemplify methods that learn from individual experience we use reinforcement learning with eligibility traces. To exemplify evolutionary methods, we use a genetic algorithm. Finally, to exemplify combined methods, we implement memetic learning using splicing.

For all of these methods, we use the same representation for the candidate solutions. The two-dimensional input space is partitioned into eight equal-sized regions in each dimension, for a total of 64 discrete states. For each state, the system learns to turn the wheels either 30° left or right. This representation gives us the ability to create a simple policy table that determines the choice to make when each state is entered. It also magnifies the problem of perceptual aliasing to the point that the states and actions are too gross for the system to learn a policy that always succeeds. This means that some failures are inevitable and the system must be capable of learning despite this fact.

For all methods, the truck-trailer rig is started at a fixed distance from the goal, at a goal angle chosen randomly with a uniform distribution between −15° and 15°, and at a hitch angle randomly with a uniform distribution between −45° and 45°. It backs until it succeeds or fails. This is known as a *trial*.

### 3.1.1 Reinforcement Learning with Eligibility Traces.

The concept of the eligibility trace in brain function was presented by Klopf (1974). Eligibility traces have since been used in several reinforcement learning systems (Singh and Sutton, 1996). Eligibility traces allow for learning when performance is temporally dependent on responses and critical evaluations are available.

The policy table is initialized by randomly choosing left or right as the action for each state. Corresponding to each table entry is a score $s$ ($0 \leq s \leq 1$) that reflects our confidence in that action. Initially, these scores are set to zero. Learning then takes place over a series of trials.

The scores are changed on failure or success, based on their eligibility for adaptation $e$ ($0 \leq e \leq 1$). At the start of each trial, all policy table scores have an eligibility value of zero. When a state is entered, the corresponding score becomes eligible for change according to

$$e^{new} = \begin{cases} e^{old} + I & \text{if } e^{old} + I \leq 1, \\ 1 & \text{otherwise.} \end{cases} \qquad (1)$$

where $e$ is the eligibility and $I$ is the initial additional eligibility just after firing. This provides a *saturating trace* (Hougen, 1998), rather than the more common replacing or accumulating traces (Singh and Sutton, 1996). In these experiments $I$ is 0.1.

At each time step, the eligibility of each score decays, regardless of whether the corresponding state was encountered on that time step, according to

$$e(t + 1) = \delta e(t) \tag{2}$$

where $\delta$ is the rate of eligibility decay ($0 \leq \delta \leq 1$). In these experiments $\delta$ is 0.99041915. (I.e., it takes 72 time steps for an eligibility value to fall to half of its current value. This value is based on the typical length of time it takes the robot to complete a trial.)

When success or failure occurs, it is likely that more recent actions are more responsible than earlier actions and are rewarded or punished to a greater extent. When a success or failure signal is received, the scores of all policy table entries are updated according to

$$s^{new} = s^{old} + e\, f \tag{3}$$

where $s$ is the score, and $f$ is the feedback (+1 for success, −1 for failure). If $s$ becomes negative, the policy decision for the corresponding state is reversed, and $s$ is set to $|s|$.

This reinforcement learning method is quite effective at learning a policy for this task. On average, in fewer than 500 trials, the robot learns a policy that succeeds well over 90% of the time (see Figure 2). This is a tough standard against which to compare other systems.

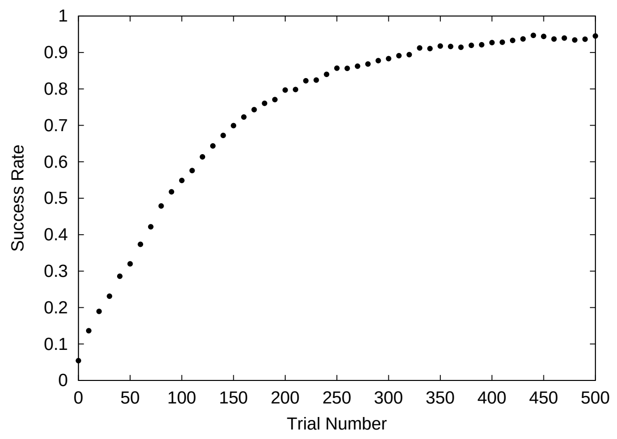

*Figure 2* Success rate, shown every ten trials, for a robot learning to back a trailer using eligibility traces. Results are averaged over 100 runs.

**3.1.2 Genetic Algorithm.** For the genetic algorithm (GA), each policy table is encoded as a one-dimensional chromosome, where each allele at each locus is a left or right action decision. The population, then, is a collection of policy tables. With a population size set at 50, we use proportional probability selection for choosing individuals for reproduction, with 10% copied directly to the next generation (elites) and 90% generated as two-offspring pairs using two parents and two-point crossover followed by mutation with a probability of 0.2% at each locus. For fitness, we have the truck-trailer rig attempt trials from random starting configurations, as described above (Section 3.1).

One might expect the fitness function for an evolutionary algorithm applied to a task such as this to involve several trials, the results of which are averaged together (Moriarty et al., 1999). Indeed, this was done for an evolutionary programming approach to the trailer-backing task (Koza, 1992). Unfortunately, the resulting large number of trials argues for offline learning, rather than online (Moriarty et al., 1999), whereas we are more interested in online learning. Indeed, in the evolutionary programming research, hundreds of thousands of trials were employed (Koza, 1992), making online learning highly impractical.

As our trials provide only minimal feedback (a binary success or failure signal), we have a choice between using many trials to get an accurate evaluation of each policy's fitness or using a rougher fitness measure. We compare fitness measures involving 10, 5, and 1 trial(s) per evaluation. For each trial, a success adds 1 to an individual's fitness value, while a failure adds 0. Because we use proportional probability selection for choosing individuals for reproduction, yet do not want to eliminate any chance that an individual without successes is selected for reproduction (to avoid genetic bottlenecks), we augment the fitness of each individual by 0.25 before performing proportional probability selection.

As expected, the fitness measures involving more trials per evaluation produce more accurate fitness estimates and require fewer generations to achieve the same level of success, as shown in Figure 3. However, one should note not only the relatively minor differences in performance of the GA when using 10 and 5 trial fitness functions but also the surprisingly good performance of the GA when using the 1 trial version. This latter result means that, if we consider the more important criterion for online learning—trials, rather than generations—the 1 trial fitness function performs significantly better than either of the more thorough variations, as shown in Figure 4. For this reason, we will use the 1 trial fitness function for comparison with memetic learning algorithms.

#### 3.1.3 Memetic Learning Algorithms.

For memetic learning algorithms, we use a simple gene-splicing method. As with the GA, each policy table is encoded as a one-dimensional chromosome, where each allele at each locus is a left or right action decision. The population, again, is a collection of policy tables and the population size is set again at 50. However, rather than using a selection mechanism to generate new individuals for successive generations, we retain all individuals—generations are marked by changes *learned* by indi-

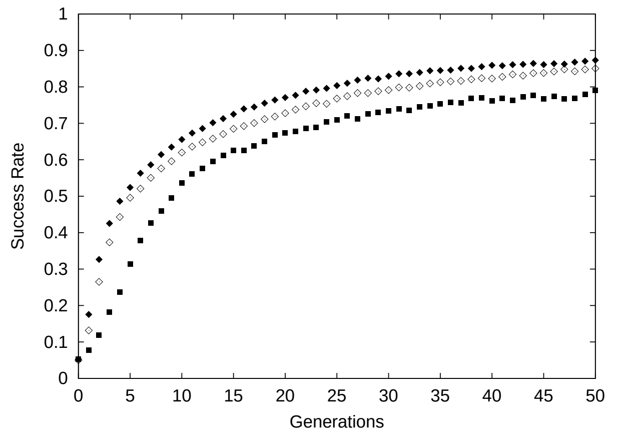

*Figure 3* Success rate, shown every generation, for robots learning to back a trailer using genetic algorithms with three different fitness functions. ◆ is 10 trials per fitness evaluation; ◇ is 5; ■ is 1. Results are averaged over 100 runs.

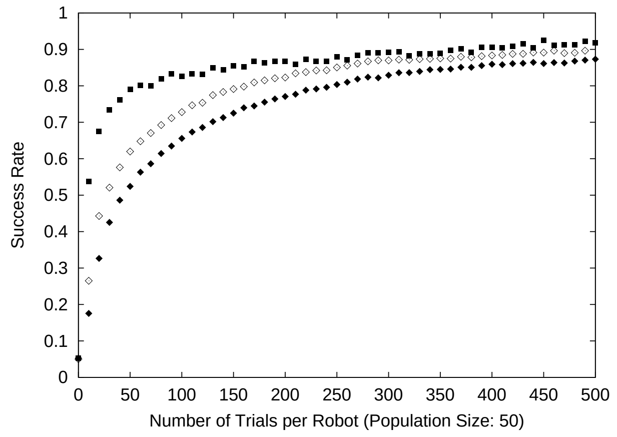

*Figure 4* Success rate, shown every ten trials per robot with a population size of 50 robots, for robots learning to back a trailer using genetic algorithms with three different fitness functions. ◆ is 10 trials per fitness evaluation; ◇ is 5; ■ is 1. Results are averaged over 100 runs.

viduals. Each individual in the population is given a single trial. Based on that trial, individuals learn in up to two ways. First, all individuals learn by direct experience. Second, if an individual fails, it learns by imitation.

**Learning by direct experience.** As with reinforcement learning, each policy table entry has associated with it a score that reflects our confidence in that action and each score has associated with it an eligibility value. The score and eligibility values are updated during and after each trial using Equations 1, 2, and 3.

**Learning by imitation.** If an individual fails, it also learns by imitation by considering each of its loci separately. For each locus, the probability that the current allele is retained is equal to $max(s, 0)$. For alleles that are not retained, a replacement allele is chosen for that locus using proportional probability selection based on the scores of all of the alleles for all individuals for that locus. Note that it is possible for the replacement allele to be the same as the replaced allele. In fact, with only two alleles for a locus, as in the present study, this is quite likely.

Because we do not want to completely eliminate any chance that an allele with a zero score is selected for imitation, we augment each allele score by 0.05 before performing our proportional probability selection. This is similar to what we do in the genetic algorithm at the level of the individual, rather than the level of the allele.

**3.1.4 Comparisons.** As discussed above (Section 3.1.2), there is more than one way to evaluate the learning rate of an algorithm, and trials is a more important measure than generations when we are interested in online learning. Even considering trials, however, we can still use various measures. One measure is total trials. This measure would be appropriate when a single robot is available on which learning is to take place. A second measure is trials per robot. This measure is more appropriate when multiple robots are available.

As shown in Figure 5, the total trials measure greatly favors reinforcement learning over both genetic algorithms and memetic learning algorithms. This is not surprising. Both genetic algorithms and memetic learning algorithms are intended as parallel search methods but counting total trials is essentially equivalent to serializing their implementation, which greatly slows their performance. Reinforcement learning, on the other hand is a serial search method by design.

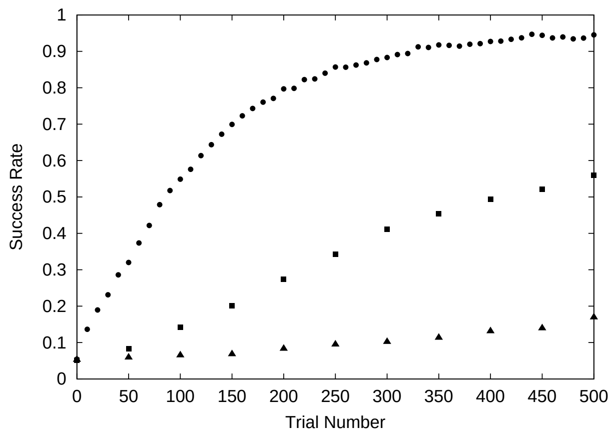

*Figure 5* Success rate for robots learning to back a trailer using three different learning methods. ● is eligibility traces (shown every 10 trials); ■ is genetic algorithm (shown every 50 trials, i.e., every generation); ▲ is memetic learning algorithms (shown every 50 trials, i.e., every generation). Results are averaged over 100 runs.

As shown in Figure 6, with 50 robots available for use, measuring trials per robot greatly favors both genetic algorithms and memetic learning algorithms over reinforcement learning. This, likewise, is not surprising, following the inverse of the logic presented above. Having multiple robots does not speed up reinforcement learning, because reinforcement learning takes place independently on each robot. Contrapositively, having multiple robots *does* speed the learning in both genetic algorithms and memetic learning algorithms, because it makes the theoretically parallel search of these methods parallel in practice.

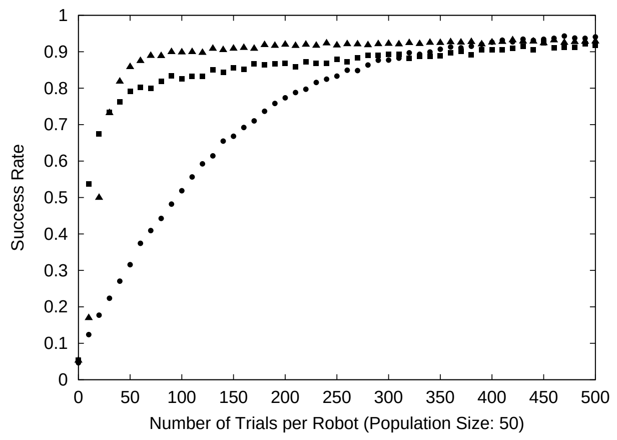

*Figure 6* Success rate for robots learning to back a trailer using three different learning methods. ● is eligibility traces (shown every 10 trials); ■ is genetic algorithm (shown every 10 trials per robot, i.e., 10 generations); ▲ is memetic learning algorithms (shown every 10 trials per robot, i.e., 10 generations). Results are averaged over 100 runs.

Figures 5 and 6 also show that, for this task, memetic learning algorithms begin learning more slowly than genetic algorithms (during approximately the first 30 trials per robot) but quickly surpass them and retain better performance for a substantial period thereafter (well in excess of 300 trials per robot). The fact that memetic learning algorithms quickly surpass genetic algorithms is expected, given the theory behind both methods. The slower memetic learning during the start-up period, however, is unexpected and will be the subject of future research.

## 4. Discussion and Future Work

We consider learning as a function of trials because this is standard in the robot learning community. However, note that both total trials and trials per robot are imperfect measures of learning time, whether one robot is available for learning or multiple robots are available. This is because not all trials may be of the same length. A difference in trial length will tend to favor non-generational learning methods because with these methods a robot can start its next trial as soon as it finishes its current trial whereas with a generation-based method all robots must wait for the slowest robot in the generation to complete before calculating their next set of policies and starting their next trials.

However, we can use population-based methods without generations—a robot may determine a new policy asynchronously based on the most recent completed trial of all robots. Comparisons of the given population-based methods to non-generational variants of them remains as future work.

In fact, much remains as future work. This paper should be seen as merely introducing a concept, rather than laying out a complete methodology. One important next step is to develop and experiment with more sophisticated splicing methods. Another important step is to combine splicing with other imitation methods such as *generalization*, which we have used previously to speed learning (e.g., Hougen et al., 2002). Additional future work will look at different meme encodings, just as different evolutionary computation methods use different gene encodings.

## Notes

1. This is not to suggest that the overall learning mechanisms in evolutionary computation methods are random. The selection of individuals for reproduction is guided by fi tness evaluations. However, the reproduction itself, whether through crossover or mutation, is done at random.

2. We are using the term *reinforcement learning* to denote methods that search either policy space or value-function space. However, we are not including evolutionary methods under this label, although other authors (e.g., Moriarty 1999) have done so.

## References

Back, T., Hammel, U., and Schwefel, H.-P. (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17.

Dawkins, R. (1976). *The Selfi sh Gene*. Oxford University Press.

Holland, J. H. (1962). Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 3:297–314.

Hougen, D. F. (1998). *Connectionist Reinforcement Learning for Control of Robotic Systems*. PhD thesis, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN.

Hougen, D. F., Gini, M., and Slagle, J. (2002). An integrated connectionist approach to reinforcement learning for robotic control: Extension to three dimensions. In *International Conference on Intelligent Autonomous Systems*, pages 134–141.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artifi cial Intelligence Research*, 4:237–285.

Klopf, A. (1974). Brain function and adaptive systems – a heterostatic theory. In *Proceedings of the International Conference on Systems, Man, and Cybernetics*.

Koza, J. R. (1992). A genetic approach to finding a controller to back up a tractor-trailer truck. In *Automatic Control Conference*, pages 2307–2311.

Lanzi, P. L., Stolzmann, W., and Wilson, S. W., editors (2000). *Learning Classifi er System: New Directions and Concepts*. Springer, Verlag, Berlin.

Moriarty, D. E., Schultz, A. C., and Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artifi cial Intelligence Research*, 11:241–276.

Moscato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Yao, X. (1999). Evolutionary computation comes of age. *Cognitive Systems Research*, 1:59–64.