Interaction of Culture-based Learning and Cooperative Co-evolution and its Application to Automatic Behavior-based System Design

Amir massoud Farahmand[1,2,3] Majid Nili Ahmadabadi[1,2] Caro Lucas[1,2] Babak N. Araabi[1,2]

Amir@cs.ualberta.ca Mnili@ut.ac.ir Lucas@ipm.ir Araabi@ut.ac.ir

(1) Control and Intelligent Processing Center of Excellence, Department of Electrical and Computer Engineering, University of Tehran, Tehran, Iran

(2) School of Cognitive Sciences, IPM, Tehran, Iran

(3) Department of Computing Science, University of Alberta, Edmonton, Canada

Abstract – **Designing an intelligent situated agent is a difficult task because the designer must see the problem from the agent's viewpoint, considering all its sensors, actuators, and computation systems. In this paper, we introduce a bio-inspired hybridization of reinforcement learning, cooperative co-evolution, and a cultural-inspired memetic algorithm for the automatic development of behavior-based agents. Reinforcement learning is responsible for the individual-level adaptation. Cooperative co-evolution performs at the population level and provides basic decision-making modules for the reinforcement-learning procedure. The culture-based memetic algorithm, which is a new computational interpretation of the meme metaphor, increases the lifetime performance of agents by sharing learning experiences between all agents in the society.**

In this paper, the design problem is decomposed into two different parts: 1) developing a repertoire of behavior modules and 2) organizing them in the agent's architecture.

Our proposed cooperative co-evolutionary approach solves the first problem by evolving behavior modules in their separate genetic pools. We address the problem of relating the fitness of the agent to the fitness of behavior modules by proposing two fitness sharing mechanisms, namely *uniform* and *value-based* fitness sharing mechanisms.

The organization of behavior modules in the architecture is determined by our structure learning method. A mathematical formulation is provided that shows how to decompose the value of the structure into simpler components. These values are estimated during learning and are used to find the organization of behavior modules during the agent's lifetime.

To accelerate the learning process, we introduce a culture-based method based on our new interpretation of the meme metaphor. Our proposed memetic algorithm is a mechanism for sharing learned structures among agents in the society. Lifetime performance of the agent, which is quite important for real-world applications, increases considerably when the memetic algorithm is in action.

Finally, we apply our methods to two benchmark problems, an abstract problem and a decentralized multi-robot object-lifting task, and we achieve human-competitive architecture designs.

Index Terms – behavior-based system design, cooperative co-evolution, reinforcement learning, culture-based method, memetic algorithm, structure learning, behavior cooperative co-evolution.

I. INTRODUCTION

Our main research goal is to develop methods for the automated design of situated agents. A situated agent is an agent that directly observes the environment using sensors and decides which action is most appropriate at any time. This agent does not usually have complete knowledge of the environment; instead, it faces the environment directly and its decisions have immediate consequences. This problem is difficult because the agent must deal with a potentially high-dimensional, unknown stochastic dynamical system.

We propose a bio-cultural approach for designing a situated agent. Though our approach is general, we formulate it for a hierarchical behavior-based architecture (See [Brooks91] or [Ziemke98] for discussion of situated agents and behavior-based systems, and refer to [Prescott99] for a comparison of hierarchical behavior-based systems with their biological counterparts). We decompose the design problem into two sub-problems of developing behavior modules and structure design and use a novel hybrid mechanism for solving both of them. Our hybrid approach has three main elements: (1) Cooperative co-evolution, (2) Reinforcement Learning (RL), and (3) a culture-based memetic algorithm.

A cooperative co-evolutionary mechanism, similar to [Potter00], is responsible for developing the new behavior modules. These modules are cooperatively co-evolved in order to maximize the performance (i.e. fitness; we use these two terms interchangeably) of agents. The role of the reinforcement learning mechanism is to organize these evolved behavior modules in the agent's architecture. Reinforcement learning provides an adaptation mechanism on the timescale of the agent's lifetime. Moreover, to reuse learned knowledge gained by other agents, our approach shares the learned structure of the high-performing agents to the "community" or "society" of agents. This knowledge sharing is accomplished through our memetic algorithm, which is quite different from the current belief of what a memetic algorithm should be (we discuss this issue in Section III.C). In our approach, memes act as a priori knowledge for

the structure learning process. This knowledge sharing increases the expected lifetime fitness of agents. The lifetime fitness, as opposed to the fitness, is the measure of an agent's performance from the beginning of learning to the end of an agent's lifetime. This is an important measure of performance because in many problems we would like our agent to behave well as soon as possible, and the lifetime fitness measures this quality of the agent.

The motivation behind this multi-scale task decomposition is (1) to benefit from the global search capability of an evolutionary mechanism, (2) to take advantage of the fast adaptation of structure learning, and (3) to reuse the previous experiences of other agents in the society to accelerate the learning process. We will further discuss the idea behind our approach in Section III and present its details in Section V.

The idea of designing an agent's controller by decomposing it to several sub-problems and using bio-inspired methods to solve them is potentially applicable to agents with various types of architecture. However, we chose the Subsumption Architecture (SSA) ([Brooks86]) from the behavior-based paradigm as our agent's architecture because of its successful implementations in real-world problems (e.g. [Brooks89], [Wang96], [Matarić98], [Parker98], and [Nili01]).

The organization of the paper is as follows. In Section II, we survey related topics. More specifically, we first introduce the behavior-based paradigm and motivate the desirability of having automatic agent design methods (Section II.A), give a survey of learning and evolutionary approaches for agent design (Section II.B), and briefly introduce reinforcement learning (Section II.C). Afterwards, we detail the main ideas of our approach in Section III. We describe the roles of cooperative co-evolution (Section III.A), learning (Section III.B), and culture (Section III.C) in our framework. After formalizing the agent in Section IV, we describe our design approach in detail in Section V. There, we propose our structure learning (Section V.A), behavior cooperative co-evolution (Section V.B), and memetic algorithms (Section V.C). Thereafter, we apply our approach to the decentralized multi-robot object-lifting task and an abstract general behavior-based problem in Section VI. We review our important results and discuss them in Section VII. Conclusions and future research directions are discussed in Section VIII.

II. RELATED TOPICS AND MOTIVATIONS

The goal of this section is threefold: We present the general motivation behind this research, provide the basic background of behavior-based systems and reinforcement learning needed to understand the proposed method, and review some related work.

A. Behavior-based Paradigm

We chose behavior-based paradigm as the design methodology of the agent's lower level "mind" or controller ([Brooks86] and [Brooks91]). Behavior-based systems are biologically plausible, relatively robust, and fault-tolerant architectures that have been used for several real-world, challenging robotic tasks (e.g. [Brooks89], [Wang96], [Matarić98], [Parker98], and [Nili01]).

Example of Behavior-based Systems: To explain the behavior-based paradigm and the meaning of "behavior" in it, we give a simple example. Consider a robot designer who wants to design a controller for a mobile robot. Her goal is to create a "robot that moves around and avoids obstacles". The description of this task, as stated in the previous sentence, is the subjective behavior of the robot and apparently depends on the eye of the beholder.

To achieve this task, she designs two control modules. The first module gives random movement commands to the mobile robot. This module does not get any input from sensors, but changes its output command from time to time. She names this module "wandering behavior" because she expects that whenever it controls the robot, the robot wanders around. The other module receives input from proximity sensors and rotates the robot whenever it comes close to an obstacle. She names this module "obstacle avoidance behavior" with a similar argument. Thereafter, she arranges these two behavior modules in the agent's architecture to manage their interaction through an arbitration mechanism. The arbitration mechanism determines which of these two behavior modules should control the robot at any time. A robot with these modules and architecture can be considered as an example of a situated agent. It is situated because the controller directly observes the environment and directly commands the actuators. No part of the architecture builds an explicit, abstract-level representation of the world, and there is no planning module for manipulating symbols.

The observable behavior of the robot is the result of interaction of these two "behavior" modules with the environment. It is possible that the interaction of these two modules with the environment will lead to complex behaviors, which have not been anticipated by the designer.

For instance, in a heavily cluttered environment, where there is always at least one object ahead of the robot, the "obstacle avoidance" behavior module alone (without the "wandering" behavior module) may always force the robot to wander around and avoid obstacles (because there is always an object in front of the robot which forces it to move from its current position). In this case, the robot's behavior does not appear as mere "obstacle avoidance", rather it can be interpreted as "moving around the field and avoiding obstacles". This "behavior" of the robot is probably different than the designer's expectation of a pure "obstacle avoidance" behavior module.

Two Notions of Behavior: Note that in the previous discussion we used the term "behavior" in two different meanings. The first is when we refer to each of those internal components of the robot's controller. We call them "behavior modules" because they are *intended* to produce some specific behaviors, though it is possible that they behave differently in practice. The second is the phenomenological interpretation of the behavior. This interpretation refers to the behavior of the agent as a whole when it interacts with the environment. This

behavior is the *emergent* result of possibly complex interactions of those "behavior modules" with the environment. In this paper, we use "behavior" in both senses. Whenever we use the word "behavior" or phrase "behavior module", we intend the former meaning. We use "overall behavior of the agent" when we are referring to the second interpretation.

Behavior-based Systems Design: Because of the complex interactions of behavior modules with the environment and themselves, designing a behavior-based system to achieve a certain goal is not an easy job. Adding a new behavior module may influence other behavior modules and considerably change the overall behavior of the agent in an unexpected manner. Indeed, one main drawback of behavior-based systems has always been the difficulty of their design. In practice, a designer often uses the tedious bottom-up trial-and-error approach to devise a behavior-based system that meets the required performance objectives [Brooks86].

The difficulty of behavior-based system design suggests that we should automate the design procedure to relieve the burden on the designer. In spite of many successful implementations of behavior-based systems (e.g. [Brooks89], [Wang96], [Matarić98], [Parker98], and [Nili01]), most of them are hand-designed, and there is not much work on automatically designing an agent's architecture (all the aforementioned systems are hand-designed). The current work suggests a hybrid approach for facilitating the design procedure. In our approach, we evolve behavior modules and learn the organization of those modules in the architecture. It satisfies all aspects of autonomy mentioned in [Ziemke98], i.e. (1) the behavior modules are self-organized, (2) the arbitration mechanism is learned, and (3) the approach treats internal memory and the input from external sensors in the same way.

B. Evolution and Learning

The design problem can be viewed as an *optimization problem* in which we are looking for an appropriate set of parameters such that our goals are satisfied. For hierarchical behavior-based systems, such as the Subsumption Architecture, we are searching for a set of parameters that describes the internal workings of behavior modules as well as their organization (structure) in the architecture. A good design methodology for situated agents must have the following properties:

- Find a suitable and working set of parameters quickly
- Work with hierarchical and multi-level decision-making architectures
- Cope with non-stationary environments
- Produce modular and reusable components

Most traditional adaptation approaches do not provide this amount of flexibility. For instance, learning algorithms that are based on local search may not find a good solution in a large and bumpy parameter space and can get trapped in a local optimum. This is especially true for policy-gradient reinforcement learning methods (See [Baxter01], [Kakade02], and [Ghavamzadeh06] for a few examples of policy-gradient methods). Nevertheless, learning methods can be relatively fast in finding some solutions even in non-stationary environments, given that the learning parameters are set properly. On the other hand, evolutionary approaches can usually find good solutions for a problem given enough time. Nevertheless, traditional evolutionary methods are slow and do not handle non-stationary environments very well. Therefore, they are not very suitable for a situated agent that has to respond quickly to changes in the environment. Moreover, most of them do not produce modular controllers, which is an important issue for designing a reusable controller.

Learning and Evolution for Behavior-based System Design: There have been some efforts to use learning or evolution to partially automate the design procedure for behavior-based systems. Examples of learning-based approaches for behavior-based system design include Maes *et al.* which used learning to adjust firing precondition of behavior modules [Maes90]. Mahadevan *et al.* proposed a learning mechanism to adjust behavior modules of the fixed-structured Subsumption architecture [Mahadevan92]. In [Matarić92], they developed a method for learning an environment's topological map. In [Matarić94] and [Matarić97], they used shaped reinforcement signal and progress estimator to accelerate learning. In [Michaud98], a memory-based approach was used to select behavior modules. In [Kohl04], they used a policy-gradient method to find a good gaiting for a quadrupedal locomotion. In [Matarić01], one can find a summary of several works that use the learning approach to design behavior-based systems.

Artificial evolution has also been used to design situated agents ([Harvey93], [Floreano96], [Floreano00], [Nolfi00], and [Chernova04]). Two examples of more closely related work are [Koza94] which used genetic programming to evolve SSA-like architectures in the wall following task and [Togelius04] which devised layered incremental evolution in a SSA-like architecture. This latter work tried to evolve modular behavior-based systems. [Floreano08] provides a recent survey of evolutionary robotics.

Taking advantage of the good properties of evolution and learning in a complementary manner is highly desirable. There is some research exploiting the good properties of evolution and learning, but they are not extensive (See [Nolfi99] for a survey of applications of learning in evolutionary robotics). It is worth mentioning that the way *learning* is usually used in evolutionary robotics context is somewhat different from what is meant in the reinforcement learning framework [Sutton98]. In the evolutionary robotics literature, learning mostly refers to neural network weight adaptation in an unsupervised (e.g. Hebbian rule) or supervised manner. The goal of learning in the evolutionary robotics literature is thus not usually explicitly formulated as maximizing some function of received rewards as in reinforcement learning. In our approach, we explicitly try to a maximize reward function, so our learning aspect is more similar to the mainstream reinforcement learning.

Although it would seem that defining a suitable reward signal is not always straightforward, successful applications of

reinforcement learning suggest that we may benefit from explicitly formulating the lifetime goal of an agent as maximizing a function of reward (e.g. the average or discounted sum of reward) and defining the fitness of the agent accordingly. As a similar view, [Whiteson06] used a neural network as a function approximator for reinforcement learning. However, instead of adapting a single neural network, they evolved a population of networks and adjusted their parameters using reinforcement learning. Their idea was that evolution would find a set of neural networks that let the agent learn better. In this paper, we propose a design methodology that benefits from both learning and evolution paradigms in addition to a culture-based knowledge sharing mechanism to develop behavior-based architectures. The feasibility of the proposed approach in its preliminary form is shown in [Farahmand06].

C. Reinforcement Learning

Our structure learning method is formulated as a reinforcement learning (RL) problem. In the following paragraphs, we briefly introduce reinforcement learning without going into details. Interested readers can refer to [Bertsekas96] and [Sutton98].

Reinforcement learning is a mathematical framework for sequential decision-making problems. The goal is deciding the optimal set of actions when an agent is situated in a stochastic dynamical environment. We describe this framework by giving an example. Suppose we want to design a humanoid robot playing soccer. The robot has some sensors, like a camera and a microphone, to observe the environment. We define its objective as playing soccer and scoring goals on the opponent; and we seek a *policy* that leads to this aim.

Reinforcement Learning Framework: This problem can be stated in a reinforcement learning framework. The robot is the *agent*, and the soccer field is its *environment*. The agent perceives the state $s_t \in S$ of the environment at time t (the position of all players, the position of the ball, commands or requests from coach, etc.) and selects *action* $a_t \in A$ that is the robot's movement command. This action is selected according to the *policy* $\pi : S \rightarrow A$ of the agent. The policy is a function that decides which action should be selected and executed at any state.

The environment has stochastic dynamics that depend on the current state and the executed action. In our example, the dynamics are the way a given action (e.g. go forward) changes the position of the robot on the soccer field. These dynamics are generally probabilistic because of noise and other unknown effects and can be described as the *state-action transition model* $P(s_{t+1} \mid s_t, a_t)$. This model describes the probability of going to state s_{t+1} if the agent chooses action a_t in state s_t.

Meanwhile, the agent receives reward r_t showing the merit of the action it was executing in that state. For instance, if the agent scores a goal, it will receive a positive reward +1, and if it scores an own goal, it will receive a negative reward -1, etc. We can state our desiderata as a reward (or reinforcement) function that generally depends on the current state s_t, the executed action a_t, and the next state s_{t+1}, i.e. $r(s_t, a_t, s_{t+1})$. Note that in general the reward can be delayed and the agent does not immediately receive it after executing an action. For example, the decision to move toward the opponent's penalty area does not yield reward immediately; the reward is only given much later if the agent scores a goal.

The goal of reinforcement learning is to find a policy π that maximizes a function of the received rewards. This function, named the *value function* V, should be defined based on our needs. For instance, we can try to maximize the expected average reward during a finite period of time or we may prefer to maximize the expected discounted sum of rewards whenever rewards in the near future are more important than rewards in the distant future (a stock market is an example of this kind of problems where the value of a money decreases over time). A reinforcement learning agent gradually improves its policy by interacting with the environment and getting (s_t, a_t, r_t, s_{t+1}) samples.

The important property of the reinforcement learning framework, which makes it different from dynamic programming, is that it does not assume the dynamics of the environment $P(s_{t+1} \mid s_t, a_t)$ and the reward function $r(s_t, a_t, s_{t+1})$ a priori. Because of this, one can say that reinforcement learning is a sample-based version of dynamic programming.

In this paper, we estimate the value of a structure (instead of the value of a state) by interacting with the environment, and try to find the structure that maximizes that value. Here, the structure acts like the policy.

Hierarchical Reinforcement Learning: Because our structure learning method applies to hierarchical architectures, we review several works in the hierarchical RL literature. There exist several RL methods for hierarchical architectures such as Feudal RL [Dayan93], Options [Sutton99], MaxQ value decomposition [Dietterich00], Hierarchies of Abstract Machines (HAM) [Parr98], and a policy-gradient approach to hierarchical RL [Ghavamzadeh03]. However, there are some important differences between our approach and more common methods for hierarchical RL. First of all, our architecture has a behavior-based nature. Each component of our architecture is a complete behavior, which is produced by the evolutionary mechanism. It decides on its own without getting any command from a central coordinator or from other behavior modules. Each behavior module is thus a direct map from the perception space to the action space. The distributed nature of the architecture enables the agent to work properly even if a few of its behavior modules become faulty. This is not common in most hierarchical RL methods, such as Feudal RL, MaxQ, and HAM, where there is usually some central or higher-level coordinator. The other important difference between our approach and traditional hierarchical RL methods is that our learning method tries to find an optimal or suboptimal structure (hierarchy), whereas other hierarchical RL methods are not directly concerned with hierarchy, but learn each component's optimal or suboptimal mapping. In

our approach, a cooperative co-evolutionary process adapts the behavior modules. See [Barto03] for a survey of hierarchical RL methods.

III. THE MAIN IDEA

Our approach to the hierarchical behavior-based system design decomposes the problem into two sub-problems and benefits from learning, cooperative co-evolution, and culture-based methods to solve those sub-problems. Our first sub-problem is developing a suitable set of behavior modules, and the second one is organizing those behavior modules in the architecture.

The suggested approach for solving this problem has three main components: (1) behavior cooperative co-evolution that solves the behavior development problem, (2) reinforcement learning for structure learning that finds the organization of behavior modules during the agent's lifetime, and (3) a culture-based memetic algorithm that accelerates the structure learning method and enhances lifetime performance of the agents (Fig. 1 shows the relationships between the cooperative co-evolutionary mechanism, the learning agent, and the culture).

One may ask the reason for selecting the evolutionary mechanism for designing behavior modules and reinforcement learning for organizing them, since other combinations are also possible. Our choice is based on the special form of hierarchical behavior-based architecture that we deal with in this paper. The main reasons for this choice are (1) the larger space of behavior modules compared to the space of structures in our architecture, and (2) the effects of a small change to the behavior modules/structure on the overall behavior of the agent. The practical consequence of this difference is that if one tries to considerably change the overall behavior of the agent and to adapt it to new conditions of the environment, it is easier to change the structure of the agent than its behavior modules. This may be helpful in cases when behavior modules have not been well-adapted to the current environment and we need a new, acceptable (but not necessarily optimal) solution to our problem quickly. Also, evolutionary methods are usually more effective in finding good solutions for high-dimensional optimization problems than local search methods. Therefore, in this research, we chose to evolve behavior modules (which has a larger search space) and to learn the structure of the agent (See [Farahmand05A] and [Farahmand05B] for examples of using learning for adaptation of both behavior and structure).

We discuss the main components of the proposed methods in the following.

A. Cooperative Co-evolution of Behaviors

To solve the problem of developing a suitable set of behavior modules, we use a cooperative co-evolutionary mechanism. The proposed cooperative co-evolutionary method evolves behavior modules, which are the basic components of the behavior-based agent's architecture.

In our cooperative co-evolutionary method, we have several different behavior (genetic) pools in each of which a specific type of behavior modules evolves. Individuals in these behavior pools encode instances of behavior modules (notice behavior pools in Fig. 1). The agent is composed of several behavior modules where each of them comes from a separate population. This type of encoding is sometimes called phenotypic co-evolution (as opposed to genotypic co-evolution described in [Krawiec07]).

The representation of individuals and the genetic operators depends on the problem and the designer; e.g. one may choose a neural network to describe the behavior modules, so the designer encodes the topology/weights of neural networks in a specific way, and defines crossover and mutation operators accordingly.

As an example of this cooperative co-evolutionary method, consider generating "obstacle avoidance" and "light seeking" behavior modules for a mobile robot. We need two behavior pools. In one pool, we evolve behavior modules that receive sonar readings and output the movement direction of the robot (obstacle avoidance) and in the other pool we evolve behavior modules that receive the input from vision sensors and output the motor command (light seeking).

The goal of the behavior co-evolution is to evolve several behavior modules such that if they are put together in the architecture, the agent's fitness (which depends on its performance in the environment) will be high. To achieve this goal, behavior modules should be *cooperatively* evolved in a way that their fittest solution is close to the optimum solution for the agent as a whole. This is not a trivial task, and we later discuss *Uniform* and *Value-based Fitness Sharing Mechanisms* for forcing this desideratum in Section V.B.1.

Our cooperative co-evolutionary mechanism, as described in Section V.B., is similar to what is known as cooperative co-evolutionary algorithms in the evolutionary computation literature (See [Potter00] and [Wiegand04]). In cooperative co-evolutionary algorithms, each population evolves a sub-component of the solution separately. Those sub-components are evaluated together and the sub-component's fitness is assigned based on this performance measure. In our agent development framework, those sub-components are behavior modules, and *Uniform* and *Value-based Fitness Sharing Mechanism* are two ways to assign fitness to sub-components.

B. Individual Learning

Now, suppose we have an appropriate set of behavior modules. An important question is how to organize these modules in the agent's hierarchical architecture. This problem has a combinatorial optimization nature. Some combinations of behavior modules lead to acceptable performance and some do not. For instance, suppose we have two behavior modules: "wandering" and "obstacle avoidance". Furthur assume that the architecture is multi-layer and the higher layers can suppress the lower layers whenever the higher behavior modules become activated (we will precisely define our architecture and the meaning of "activation" in Section IV). If

the "wandering" module is higher than the "obstacle avoidance" module, the agent always executes the "wandering" behavior and never executes the "obstacle avoidance". This specific organization of behavior modules likely leads to crashing into obstacles. On the other hand, if the "obstacle avoidance" behavior module has a higher priority than the "wandering" behavior module, the agent would wander in the world and avoid obstacles when they are confronted.

The goal of the individual learning component of our agent design methodology is finding an appropriate ordering of behavior modules, i.e. the agent's structure. The proposed structure learning method is based on reinforcement learning principles. It seeks the organization of behavior modules that maximizes the received reinforcement signal.

The interplay of learning and evolution is important in our framework. Learning has a multi-level effect on the agent and its society. It adapts the agent to the current environment. This is especially helpful whenever the environment changes faster than what evolution can track. Moreover, learning can indirectly influence the genetic material of a species through the Baldwin effect (as opposed to the Lamarckian viewpoint which says that learned traits can directly be inherited). An individual that learns can find a local optimum even if the phenotype induced by the genotype was not close to the optimum. This effect increases its chance of survival. However, learning takes precious lifetime of the individual. This generates a selection pressure toward individuals that have the correct phenotype from birth - without any need for learning (See [Hinton87] for a computational model on the effects of learning and evolution, and [Nolfi99] for a survey of the mutual effect of learning and evolution).

C. Culture-based Memetic Knowledge Sharing

Learning not only changes the fitness landscape, but also affects the *culture* of the society the agent is living in. This culture can, in return, have an effect on learning, too. Culture, as we interpret it, is *a medium for sharing experiences and solutions to previously encountered problems*. Each individual may use this common knowledge as its initial knowledge and then refine this knowledge for its own special needs. The individual may share this new solution with other members of society through social interactions and incrementally change the culture.

If we assume the problems that individuals face are more or less similar, and also assume that they have similar tools and means for solving them, we can conclude that agents can benefit from previously experienced ways of success or failure by sharing this learned knowledge through the described cultural mechanism.

The latter description of knowledge transfer in the society bears resemblance to the *meme* metaphor. A meme is "a unit of information that reproduces itself as people exchange ideas" [Dawkins76]. *Memetic algorithms* [Moscato92], which are considered a hybridization of local and global search in the evolutionary computation community, are a promising meme-inspired approach for solving difficult optimization problems efficiently ([Radcliffe94], [Merz99], [Merz00], [Buriol04], [Krasnogor04], [Ong04], and [Zou04] are a few examples of traditional memetic algorithms. See also [Moscato03], [Krasnogor05], and [Smith07] for reviews of memetic algorithms). The idea of this interpretation of memetic algorithms is that local search can guide us to a local optimum quickly while global search methods, such as those common in evolutionary algorithms, increase the chance of finding a very good or even optimal global solution. Nevertheless, the idea of a meme can be interpreted and implemented in other ways rather than as a simple local search before/after a genetic operator - as is common in the evolutionary computation community (See [Federici03] for a sample of other interpretations of memes). For instance, here, we consider a meme as the tradition or the cultural belief for solving a problem common to many agents in the society. Those beliefs act as good a priori knowledge for solving a common problem, while the individual fine tunes that knowledge for its special needs. The individual can transfer this knowledge back to the culture or the *meme pool* - as we call it in this paper. In summary, in this interpretation, memes act as a priori knowledge for the learning process. We use this interpretation of memes alongside the traditional one in this paper.

IV. MATHEMATICAL MODEL OF THE AGENT

Our behavior-based architecture consists of a set of behavior modules parallel to each other with different priorities, see Fig. 2. Behavior modules that are placed higher in the structure have priority over lower modules. As an example of this architecture, consider our designer's mobile robot with three behavior modules "wandering", "light seeking", and "obstacle avoidance". The agent with this set of behavior modules observes the combined state s of its environment and itself and proposes action a. The agent's state changes according to the dynamics of the environment, and it receives a reinforcement signal r.

Suppose we have a set of n behavior modules $\{B_i\}; i = 1,...,n$, defined as the following map between state and action (See Fig. 3):

$$B_i : S_i' \to A_i' \qquad \mathrm{i} = 1,...,\mathrm{n}$$
$$A_i' = A_i \cup \{\text{No Action}\},\ S_i' = \{s_i' | s_i' = P_i(s); \forall s \in S_i\} \quad (1)$$
$$S_i \subset S,\ A_i \subset A,\ P_i : S \to S_i'$$

where S is the state space (which consists of the inputs from sensors and possibly the internal memory of the agent), S_i is the subset of the state space observable by behavior module B_i ($S_i \cap S_j \neq \emptyset$, in general), A is the set all possible actions, and A_i is the set of B_i's output actions, i.e. actuators. P_i is the mapping that projects the agent's state S to the behavior B_i's perception. As the formulation indicates, all behaviors' inputs are not necessarily the same. This is reasonable since in real-world problems different behavior modules observe the world

differently. For instance, "obstacle avoidance" uses sonar sensors as its input, and "light-seeking" behavior module just uses the output of light-sensitive sensors.

In our architecture, each behavior module B_i's action space is augmented with "No Action" (*NA*). *NA* is a virtual action that even if selected does not change actuators. The role of *NA* is allowing behavior modules not to activate in some regions of their excitation space. If *NA* is selected appropriately, it would enable the agent to achieve higher performance. To see why, suppose the agent has no *NA*. If we add *NA*, it can achieve the same performance level by simply ignoring the newly added *NA* actions. However, it is possible that for some states, the agent chooses a *NA* and gains more reward by letting lower behavior modules take control. If the algorithm can find the global optimum, the performance of the agent with *NA* would be certainly equal or greater than the one without. Nevertheless, there is a trade-off between extra flexibility coming from this additional virtual action and the difficulty coming from the slightly enlarged search space (the increase in the search space would be negligible if the size of A_i is already large).

In order to see how a behavior module works, suppose the world state is $s \in S$. Each behavior module B_i senses S through P_i-projected subspace S_i'. If $s_i \in S_i$ (or $s_i' \in S_i'$ in the behavior's internal representation), behavior B_i will be **excited**. Behavior module B_i will be **activated** and output a_i if $a_i = B_i(P_i(s_i)) = B_i(s_i') \neq NA$. Behavior module B_i does nothing if it is not excited or it is excited but selects $a_i = NA$. It is the job of the designer to specify these excitation subspaces –which is equivalent to the design of sensory system for a controller. However, the agent itself will "figure out" when to become activated through a co-evolutionary mechanism.

As a clarifying example, suppose our mobile robot is in the middle of a large room with a lamp in a corner. The robot has eight sonars and there are four light-sensitive sensors; so a 12-dimensional vector describes the state of the system $s = \left[sonar_{1\times 8} \quad light_{1\times 4} \right]^T$. In this situation, two of the robot's light sensors turn on because they observe the lamp. Also, sonar sensors report a large number because no object is close to them. The "Obstacle avoidance" behavior module only uses sonars information to decide. Therefore, it just observes the first eight dimensions of the state space (i.e. $P_{obstacle} : \Re^{12} \rightarrow \Re^8$). It also becomes excited whenever any of the sonars report a distance closer than some threshold, say two meters. The light-seeking module observes the last four dimensions of the state vector and becomes excited if any of those values is non-zero. In the aforementioned situation (the robot in the middle of the room), the "obstacle avoidance" behavior module does not become excited because it is far from any objects. However, the "light-seeking" module becomes excited since two of its dimensions are non-zero. Depending on the behavior module, it may suggest a real "action" (such as a "move forward") or *NA*. Here, suppose the "light-seeking" module suggests an action that leads the robot toward the light. After a while, our robot gets close to a wall. Now, the sonar readings are showing distances smaller than two meters. In this situation, the "obstacle avoidance" module becomes excited in addition to the "light-seeking" module.

The agent cannot simultaneously execute the suggested action of all behavior modules, and thus a conflict arises. Deciding which action should control the agent depends on the agent's architecture.

In this paper, we consider a special but important case of the Subsumption Architecture and call it the Purely Parallel Subsumption Architecture (PPSSA), see Fig. 2. PPSSA is a hierarchical behavior-based architecture in which all behavior modules are parallel to each other. In PPSSA, a higher behavior module has the priority to suppress the lower ones. Suppressing means that the higher activated behavior does not let the lower activated (and suppressed) behaviors put their actions on the actuator's bus and control the agent. In this situation, no matter what the actions of lower behaviors are, the agent is controlled by the action of suppressing behavior. Whenever a behavior module becomes activated and suggests some action and is not suppressed by any other behavior, that behavior module becomes the **controlling** behavior of the agent.

To formalize our architecture, assume that we have a set of n behavior modules $\{B_i\}$ and an m-behavior architecture T consisting of m of those behaviors ($m \le n$). The set $\{B_i\}$ shows what behavior modules are available in the architecture. The vector T describes the organization of behaviors in the architecture. The element $T(i)$ denotes the behavior module in the i^{th} layer of T where the numbering starts from the lowest layer, e.g. $T(1)$ is the index for the lowest behavior in the architecture. We can define the architecture by knowing $\{B_i\}$ and T as follows:

$$T = [B_{ind(1)}\ B_{ind(2)} \ldots B_{ind(m)}]^{\mathrm{T}} \quad m \le n$$
$$ind(i): j \quad (\text{that indicates } B_j \text{ is in the } i^{th} \text{ layer}) \qquad (2)$$

As an example, our mobile robot's controller is described by the set of behavior modules $\{\text{Avoid obstacle, Light Seeking, Wandering}\}$ and the structure $T = [\text{wandering} \quad \text{light seeking} \quad \text{obstacle avoidance}]^{\mathrm{T}}$. This structure shows that the highest behavior in the architecture is "obstacle avoidance" and the lowest one is "wandering". The goal of this paper is to learn this structure and to co-evolve those behavior modules automatically.

V. PROPOSED METHOD

In this section, we describe our hybrid behavior-based system design framework in detail. Consider that we want to develop a behavior-based agent. We need to find a suitable behavior module repertoire $\{B_i\}$ and an appropriate organization of them in the architecture T such that our objectives are met. We define our objectives as a function of the reinforcement signal that the agent receives. Having r_t as the reinforcement signal received at time t (which may depend on the system's

state, selected action, etc.) and defining R as a random variable that indicates the episode's return for the agent, the value of the whole system with structure T and set of behavior modules $\{B_i\}$ $(i=1,...,n)$ is

$$V_T = \underset{\pi}{E}\left[\frac{1}{N}\sum_{t=1}^{N} r_t \,\middle|\, \begin{array}{l}\text{the agent with structure } T \\ \text{and set of behaviors } \{B_i\}(i=1,...,n)\end{array}\right] = \underset{\pi}{E}\left[R \,\middle|\, \begin{array}{l}\text{the agent with structure } T \\ \text{and set of behaviors } \{B_i\}(i=1,...,n)\end{array}\right]. \quad (3)$$

in which expectation is taken over all possible trajectories of the agent during its lifetime. If the task is continual, the state distribution may become stationary and this expectation will be with respect to the induced state distribution. The probability distribution of trajectories (or the stationary distribution) depends on the state transition probability of the environment ($P(s_{t+1} \mid s_t, a_t)$) and the policy π of the agent. The policy of the agent is a function of the state space to the action space. For PPSSA, it depends on the behaviors $\{B_i\}$ and their organization T in the way described in Section IV. We may omit the explicit dependence of the value function on the agent's policy in the rest of the paper to simplify our notation.

The design goal is finding a set of behaviors and the structure that maximize the value of the agent. This can be formalized as

$$\left\{\left\{B^*_i\right\}, T^*\right\} = \underset{\{B_i\},T}{\arg\max} V_T. \quad (4)$$

where T^* is the optimal structure and $\{B_i^*\}$ is the set of optimal behaviors. Note that the result of optimization problem depends on the way we define the space of behaviors. For example, the result would be different depending on whether we define it as a set of look-up tables or linear function approximators.

As equation (4) shows, there are two different parameter sets for the optimization task that are dependent on each other: parameters that describe each behavior module and parameters that describe the organization of behavior modules in the architecture. One may take different approaches to solve this optimization problem: evolving behavior/evolving structure separately or together, learning behavior modules and evolving structure, evolving behavior modules and learning structure, or learning both behavior modules and the architecture.

If we want a modular system with re-usable behaviors, we should not change behavior modules very fast. If we do so, we need to develop a new set of behavior modules for every new situation the agent may face. Instead, we can re-organize behavior modules in the structure in order to change the overall behavior of the agent whenever it faces change in the environment or the goal.

In the structure learning, we assume having a behavior module repertoire $\{B_i\}$ and our goal is choosing an ordered sequence T with m out of n behaviors that maximizes (3):

$$T^* = \underset{T}{\arg\max} V_T. \quad (5)$$

On the other hand, the behavior co-evolution problem considers adjusting each behavior B_i's mapping from its own state space (S_i') to the augmented action space (A_i') using evolutionary process. In other words, a behavior module must choose an appropriate action a_i^* in each s_i' that maximizes its fitness in T^*.

To develop a behavior-based agent, we co-evolve a set of behavior modules in their own genetic (behavior) pools. Each pool has its own genetic properties and in general, different behavior modules have different genotypes. The algorithm randomly selects a behavior module from each pool to make a set of behaviors $\{B_i\}$ and gives it to the agent (Fig. 1). By interacting with the environment and receiving a reinforcement signal, the agent tries to find an ordering of behavior modules that maximizes the received reward. This means that the agent tries to find the best possible architecture for the given set of behaviors. Thereafter, the agent is assigned a fitness based on its performance. The fitness is based on some weighted average of the agent's received reinforcement signal during its lifetime. Noting that we need to evolve behavior modules to increase the agent's fitness, we need a mechanism to relate the fitness of the agent to the fitness of behavior modules –this is called credit assignment in multi-agent learning literature [Harati07]. This mechanism must force the cooperation between behavior pools so that the agent can maximize its fitness. We discuss this mechanism in Section V.B.1.

The aforementioned optimization procedure is similar to the traditional notion of memetic algorithms as there is a local search stage during evolution. Despite this similarity, the learning is not directly performed in the space of behavior modules, but instead it is done in the space of structures that uses those genetically inherited behavior modules as its components. In our approach, the search space for evolution is not the same as the search space for learning, though they are coupled through the agent's performance in the environment.

After the agent's lifetime, it returns the best learned structure to the culture of its society (meme pool). Based on the agent's fitness (which depends on both the learned structure and the set of available behavior modules), the meme would survive in the culture or diminish. This meme pool acts as a guide to newborn agents. A "baby" agent receives one of those memes from the meme pool as its initial structure. This initial knowledge helps the agent to perform better from its early stages of life.

We should emphasize that two different meme-like concepts are used in our framework:

- Local search after evolution (structure learning)
- A culturally-induced a priori knowledge based on the best surviving structures

Pseudo-code of the proposed approach is outlined in Fig. 4. Details of each component of the method are described in the next subsections.

A. Structure Learning

In this subsection, we propose a method for structure learning. Our aim is finding T^* that satisfies (5) assuming that we have $\{B_i\}$ (a set of behavior modules given to the agent by the designer or developed by the co-evolutionary mechanism). To solve this problem, we need to find solutions to the following sub-problems:

Representation: How should the agent represent knowledge gathered during learning?

Hierarchical Credit Assignment: How should the agent assign credit to different behavior modules and layers in its architecture?

Knowledge Updating: How should the agent update its knowledge when it receives reinforcement signal?

The agent must have a data structure where its lifetime experiences can be stored in a meaningful and compact way. This representation is used to infer the correct organization of behavior modules. An appropriate representation must be capable of defining a large class of possible combinations of structures, have a small representation space and, use information gathered during learning wisely. The other important issue is determining the responsibility of behavior modules for the received reinforcement signal and assigning appropriate credit to them, i.e. the hierarchical credit assignment problem. A good choice of representation facilitates this task. The last issue is the way we should update the knowledge representation using clues from the credit assigner.

We solve these problems constructively starting from defining an appropriate representation named the Zero-Order representation. Thereafter, we show how to decompose the whole system's value function V_T (3) into simpler components that can be estimated online. This decomposition allows us to take advantage of the system's architecture to assign credit to its components and to update values effectively.

A.1) Zero-Order Representation

In this representation, we store the expected value of each behavior module in each layer. In other words, the merit of being in a layer for each behavior module is stored. We write the value of the structure T as

$$\begin{aligned} V_T &= E_\pi[R] = E_\pi\left[\frac{1}{N}\sum_{t=1}^{N} r_t\right] \\ &= E_\pi\left[\frac{1}{N}\sum_{t=1}^{N}\{r_t \wedge (\text{"}L_1 \text{ is controlling"} \vee \text{"}L_2 \text{ is controlling"} \vee ... \vee \text{"}L_m \text{ is controlling"})\}\right] \\ &= E_\pi\left[\frac{1}{N}\sum_{t=1}^{N}\{r_t \wedge \text{"}L_1 \text{ is controlling"}\}\right] + ... + E_\pi\left[\sum_{t=1}^{N}\{r_t \wedge \text{"}L_m \text{ is controlling"}\}\right] \\ &= E_\pi\left[\frac{1}{N}\sum r_t \mid L_1 \text{ is controlling}\right]\cdot P(L_1 \text{ is controlling}) \\ &\quad + E_\pi\left[\frac{1}{N}\sum r_t \mid L_2 \text{ is controlling}\right]\cdot P(L_2 \text{ is controlling}) \\ &\quad + ... + E_\pi\left[\frac{1}{N}\sum r_t \mid L_m \text{ is controlling}\right]\cdot P(L_m \text{ is controlling}) \end{aligned} \tag{6}$$

where $E_\pi\left[\frac{1}{N}\sum r_t \mid L_i \text{ is controlling}\right]$ is the expected reward following the agent's policy when the i^{th} layer takes control and $P(L_i \text{ is controlling})$ is the probability of a layer L_i being the controlling one. Note that this decomposition is possible because different time instances $\{t \mid \text{"}L_i \text{ is controlling" in timestep } t\}$ are mutually exclusive. Here, we assume that at least one of the behavior modules is active at every moment, so we can write the third equality.

Defining $V_{ZO}(i,j)$ -Zero-Order value- as

$$V_{ZO}(i,j) = V_{ij} = E_\pi\left[\frac{1}{N}\sum r_t \,\middle|\, \begin{matrix} B_j \text{ is the controlling} \\ \text{behavior in the } i^{th}\text{layer} \end{matrix}\right] \tag{7}$$

we have

$$\begin{aligned} &E_\pi[\frac{1}{N}\sum r_t \mid L_i \text{ is controlling}] \\ &= \sum_{j=1}^{n} P\{B_j \mid L_i\}\, E_\pi\left[\frac{1}{N}\sum r_t \,\middle|\, B_j \text{ is the controlling behavior in } L_i\right] \\ &= \sum_{j=1}^{n} P\{B_j \mid L_i\} V_{ij} \quad i = 1,...,m \end{aligned} \tag{8}$$

where $P\{B_j \mid L_i\}$ is the probability that B_j is the controlling behavior whenever L_i is the controlling layer. Altogether, V_T can be written as

$$V_T = \sum_{i=1}^{m}\sum_{j=1}^{n} P\{B_j \mid L_i\} V_{ij} P(L_i \text{ is controlling}). \tag{9}$$

The cardinality of the representation space is

$$\text{cardinality(ZO)} = n \cdot m. \tag{10}$$

which is much smaller than the cardinality of a representation that stores all possible behavior combinations.

Equation (9) relates the value of the agent to the organization of behavior modules in its structure. Therefore, we can estimate the value of a structure by evaluating this summation provided that we know all terms used in the summation. In order to find the optimal structure, we should find the structure that satisfies (5). To do so, we must have an estimate of V_{ij}, $P(L_i \text{ is controlling})$, and $P\{B_j \mid L_i\}$.

According to the definition of V_{ij} (7), credit assignment is straightforward: V_{ij} must be updated whenever B_j is the controlling behavior module in the i^{th} layer. Therefore, if layer i is the controlling layer and B_j has been activated in it while the system receives the reinforcement signal r_k, V_{ij} must be updated similar to what is common in reinforcement learning:

$$V_{ij_{k+1}} = (1-\alpha_{k,ij})V_{ij_k} + \alpha_{k,ij} r_k \quad \begin{pmatrix}\text{whenever } B_j \text{ is the controlling} \\ \text{behavior in the } i^{th} \text{ layer}\end{pmatrix} \tag{11}$$

with $0 < \alpha_{k,ij} \leq 1$. Based on the value of $\alpha_{k,ij}$, we put more or less emphasis on the old experiences.

Estimation of $P(L_i \text{ is controlling})$ and $P\{B_j \mid L_i\}$ is not difficult and one can set a counter for these variables and increase them accordingly. However, there is another elegant method to estimate all of them at once. Instead of updating components of (9) separately, it is possible to estimate all of them together by defining $\tilde{V}_{ij}$ as

$$\tilde{V}_{ij} = P\{B_j \mid L_i\} V_{ij} P(L_i \text{ is controlling}). \tag{12}$$

Therefore, we have

$$\tilde{V}_{ij_{n+1}} = (1-\alpha_{n,ij})\tilde{V}_{n,ij} + \alpha_{n,ij} \begin{bmatrix} "B_j \text{ is active at time step } n" \times \\ "L_i \text{ is controlling at time step } n" \times \\ r_n \end{bmatrix} \tag{13}$$

with $0 < \alpha_{n,ij} \leq 1$.

Hence, it is now possible to find the arrangement of behavior modules that satisfies (5) and maximizes reward. Different methods can solve the combinatorial optimization problem of finding T^*. In our implementations, we have used simple stochastic search. That is, calculating values for many different randomly chosen structures and selecting the one with the highest value.

There is another way to represent the value of structure. Similar to what we did in this subsection, one can develop a structure learning method based on the First Order representation [Farahmand05A]. In that representation, we store the value of the *relative ordering of behavior modules in the structure* (as opposed to the value of a behavior module being in a specific layer of the structure in the Zero-Order representation). Nevertheless, the current representation is sufficient for our agent design task and we do not use the First Order representation in this paper.

One final note about the structure learning method is worth mentioning. When we want to find an architecture that maximizes (9), we need to have estimates of V_{ij}, $P(L_i \text{ is controlling})$, and $P\{B_j \mid L_i\}$. These estimates, however, depend on the current structure of the agent, and therefore, may not be an accurate and unbiased estimate for any arbitrary structure. The result would be that we are actually solving an approximate version of the original optimization problem. This problem has resemblance to off-policy policy evaluation in the conventional reinforcement learning framework, which is not easy in general. Most likely, the estimated error would be very small whenever we are evaluating the same structure that the agent uses for decision-making (this case is like on-policy evaluation as in SARSA [Sutton98]). This error would be larger for structures that lead to totally different selection patterns of behavior modules. An analysis precisely showing this effect is not presented in this paper and the mathematical description of the convergence behavior of the provided structure learning algorithm remains an open problem. However, by randomly selecting structures during the agent's lifetime, we can help the agent find the optimal structure. In practice, the behavior of the structure learning in our experiments was satisfactory. Thorough experimental evaluation of the structure learning is provided in [Farahmand05A].

B. Cooperative Behavior Co-evolution

In this section, we discuss different aspects of our cooperative co-evolutionary behavior development scheme. In Section V.B.1, we present the general framework of cooperative co-evolutionary mechanism. In Section V.B.2, we give a specific example of behavior representation that we will use in our experiments, and show how genetic operators are defined in this case. We should note that one might define different kinds of behavior module representations (e.g. neural networks) and various types of genetic operators based on the specific problem with which one is dealing.

B.1. Behavior Pools and Fitness Sharing Mechanisms

In contrast to most evolutionary approaches in the evolutionary robotics community, we do not use a single population of monolithic behavior modules (or controllers). We discriminate behavior modules by their *supposed* role and make them evolve in their own genetic (behavior) pools. In this framework, all behavior pools evolve cooperatively to increase the fitness of agents. Fitness of each behavior pool directly or indirectly depends on the fitness of the agent. The relation between these two values is determined by Uniform/Value-based Fitness sharing mechanisms.

The designer determines the supposed role of each behavior module by specifying the input and output spaces of each behavior module. For instance, when one wants to have an "obstacle avoidance" behavior for the mobile robot, one may believe that the proximity sensors provide an appropriate type of input for this task. As mentioned before, it is possible that the final "overall behavior" of the agent and the way this specific behavior module contributes to it may be completely different from what was had expected.

Each genetic pool consists of many individuals. The "type" of behavior modules between different pools is not the same, but all individuals in each of those pools describe the same type of behavior modules. For instance, in one pool we have individuals for "obstacle avoidance" behavior and in another pool, we have different individuals evolving to produce "wall following" behavior. The evolution process for each pool has no direct interaction with other populations. Therefore, to have a set of n different possible behaviors, it is necessary to make a set of n different genetic pools with a separate evolutionary process in each of them (Fig. 2).

We need a mechanism to encourage cooperation between behavior modules in the direction of increasing the fitness of the agent. The mechanism we use to cooperatively co-evolve behavior modules is inspired by the Enforced Sub-Population (ESP) [Gomez97] algorithm, and is similar to what are commonly known as cooperative co-evolutionary algorithms ([Potter00] and [Wiegand04]). Cooperative co-evolutionary algorithms must be differentiated from competitive co-evolutionary algorithms where populations explicitly compete with each other, e.g. predator and prey scenarios [Rosin97].

First, a random behavior module is selected from each population to make a behavior set $\{B_i\}$. The agent uses this set to interact with the environment. The learning procedure organizes those behavior modules in the architecture according to the received reinforcement signal (See Section V.A). It is evident that the performance (i.e. fitness) of the agent depends on the correctness of its behavior modules. By "correct", we mean that each of those behavior modules performs such that by appropriately arranging them in the architecture, the agent can do something useful. If all behavior modules are correct, the agent will find a suitable organization and perform well. On the other hand, if some behavior modules act incorrectly and output misleadingly, the fitness of the agent will degrade.

Nevertheless, the exact contribution of each behavior module is not known a priori because the agent's fitness (which we can measure directly) is the result of the complex interaction between all the behavior modules and the structure with the environment. In this paper, we propose two mechanisms for estimating the contribution of each selected behavior module based on the fitness of the agent itself: **uniform fitness sharing** and **value-based fitness sharing**.

Uniform Fitness Sharing: As the performance of the agent is highly coupled with the performance of each of its behavior modules, we may estimate the fitness of that behavior module as the average fitness of all agents in which that specific behavior module contributes. Thus, if a single instance of a type of behavior module takes part in the architecture of several agents, say 10, the fitness of that behavior module is defined as the average fitness of those 10 agents. More precisely, if we define the fitness of an agent with behavior set $\{B\}$ as

$$V_{\{B\}}\Big|_{\text{Last K episodes}} = f_{\{B\}}\Big|_{\text{Last K episodes}} = E\left[\frac{1}{K}\sum_{t\in\text{Last K episode}} r_t \,\middle|\, \begin{matrix}\text{the agent with } \{B\},\\ t\in \text{Last K episode}\end{matrix}\right], \quad (14)$$

each behavior module's fitness $f^u\left(B_i^j\right)$ according to the uniform fitness sharing mechanism is (u in f^u is for *Uniform*)

$$f^u\left(B_i^j\right) = \frac{1}{N}\sum_{\{B\}_i^j} V_{\{B\}_i}\Big|_{\text{Last } k \text{ episodes}} \quad (15)$$

where B_i^j is the j^{th} individual from the i^{th} population and $\{B\}_i^j$s are N uniform randomly chosen sets of behavior modules in which $B_i^j \in \{B\}_i^j$. Behavior modules $\{B\}_i^j$ are chosen randomly, so it is possible for a specific behavior module to be selected more often than others. However, by having enough trials, we become sure that each behavior module has been involved in a sufficient number of trials and that the estimated fitness is a good measure of performance. Note that, like ESP, in our uniform fitness-sharing scheme, we share fitness to its components according to the overall fitness of the agent (in our case, the components are behavior modules, and in ESP they are neurons).

Value-based Fitness Sharing: The uniform fitness sharing mechanism does not differentiate between different behavior modules in the architecture when assigning fitness to the behavior modules. Instead, it considers the fitness of the whole system and then estimates the fitness of each behavior module by averaging over the fitness of several agents. However, if we have some reliable information about the contribution of each behavior module to the agent's performance, we might have a better estimate of each behavior module's fitness. Fortunately, this kind of information is readily available in our structure learning framework. Considering the way we have defined the Zero-Order representation, we can express the contribution of each behavior module in the architecture as:

$$V_{T(B_i)} = \sum_{q=1}^{m} P\{B_i \mid L_q\} V_{qi} P\left(L_q \text{ is controlling}\right) = \sum_{q=1}^{m} \tilde{V}_{qi}. \quad (16)$$

This equation summarizes the contribution of a specific behavior module B_i in all layers of the architecture based on its conditional value, probability of becoming the controlling behavior module, and the probability of a layer becoming controlling. Given that a behavior module may behave differently in different architectures due to the effect of other behavior modules, it is wise to average over several agents with different architectures and sets of behavior modules $\{B\}_i^j$. In other words,

$$f^{v}\left(B_i^j\right) = \frac{1}{N}\sum_{\{B\}_i^j} V_{T(B_i)} \qquad (17)$$

with the same definition of notation and parameters as (15) (v in f^{v} is for the *Value-based fitness sharing*).

These performance measures are flexible. The designer can change them in order to accommodate one's needs. For instance, one may change K in (14) to include all episodes, so the measure would be the indicator of lifetime performance, or one can change it to include only the last few episodes and it would be the indicator of the ultimate performance of the agent. The same is true for the value-based fitness sharing mechanism by changing the way $\tilde{V}_{ij}$ is updated. If we use $\alpha_{n,ij} = 1/n$ in (13), it would be an estimate of the average performance of that behavior module in the architecture. Moreover, it is possible that fitness estimation follows a different updating rule from what the learning and structure selection is based on.

Although our cooperative co-evolutionary method is similar to usually practiced cooperative co-evolutionary algorithms ([Potter00] and [Wiegand04]), there are a few differences. First, we evaluate the fitness of all sub-components of the solution (i.e. behavior modules of the agent) simultaneously, as opposed to evaluating the fitness of a sub-component while fixing other populations. This simultaneous evaluation and co-evolution is crucial for tasks where experience is expensive. The other important difference is the way we assign fitness to those sub-components. The uniform fitness sharing mechanism assigns the fitness of the whole solution (i.e. agent's fitness) to sub-components (and can be considered more similar to the usual practice of cooperative co-evolutionary methods), while the value-based fitness sharing mechanism tries to extract the contribution of each sub-component in a more elegant way.

B.2. Genetic Operators

Genetic operators should be designed in a compatible way with the internal representation of each behavior module. In this work, we use a look-up table representation for each behavior module and define appropriate genetic operators. Nevertheless, our approach is not limited to this special look-up table representation of behavior modules or suggested genetic operators, and other representations and/or genetic operators can be used too.

Behavior module B_i (and also B_i^j which is an instance from that family) is defined as (See (1))

$$B_i(s') : S_i' \rightarrow A_i'. \qquad (18)$$

The input space (perception/internal memory) and output space (action) dimensions and the number of partitions of each dimension determine the size of this table in our representation. These dimensions are determined by the designer.

The function $B_i(s_1, s_2, ...)$ can take integer values between 0 to $|A_i|$. Whenever it takes zero (i.e. $B_i(s_1, s_2, ...) = 0$ for some $(s_1, s_2, ...) \in S_i'$), the output of that behavior module at that point of input space is *NA*; the output is the usual action of the behavior module whenever $B_i(s_1, s_2, ...) \neq 0$. As an illustrating example, suppose we have a behavior module that has a two-dimensional input space and can output two different actions alongside *NA*. Assume that one dimension of the input space can take two values and the other takes three. A typical instance of this class of behavior modules is:

$$B^1 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

We use two different mutation operators for this specific representation: *hard* and *soft*. Hard mutation, which is selected with probability $p_{m_{hard}}$, replaces a behavior module's matrix with a totally new random one. Soft mutation perturbs each matrix and changes some of its elements in order to search nearby points in the solution space. This change is in the form of randomly assigning a new action to some random position of the matrix with probability $p_{m_{soft}}$. For instance, in our example it can be done by changing the value of position (2,3) of B^1 from '1' to '0'. The result is the following behavior module:

$$B^2 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

The crossover operator is defined as

$$B_i^{j^{new}} = XB_i^{j^{old}} + \overline{X}B_i^{k^{old}} \qquad (19)$$

where X is a random binary matrix with an appropriate dimension, $\overline{X}$ is its binary complement, and $B_i^{j^{old}}$ and $B_i^{k^{old}}$ are two behavior modules selected by the selection mechanism based on their fitness. The elements of matrix X are coming from a Bernoulli distribution with probability p_c, i.e. $P\left(X_{ij...l} = 1\right) = p_c$. This crossover operator generates offspring that inherit some elements from both parents.

To show the effect of the crossover operator, take behavior module B^3:

$$B^3 = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

The crossover operator first selects a matrix X. The elements of X are selected randomly with a probability of p_c of being 1. Therefore, if p_c is 0.5, we may get a matrix X:

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

and the result of the crossover operator between B^1 and B^3 would then be:

$$B^{new} = XB^1 + \overline{X}B^3$$
$$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 1 \end{bmatrix}$$

The crossover operator can be symmetric in the sense that it produces two offspring using the same formula but with X for the first child and $\overline{X}$ for the second one, or it can be asymmetric with just one child from both parents.

It is notable that there is no substantial difference between our proposed genetic operators and those that are usually used in genetic algorithms. The conventional operators usually work on string representation of chromosomes, but ours operate on matrix representation. For instance, it is easy to see that the crossover operator is quite similar to the conventional uniform crossover.

C. Memetic Algorithm

In this section, we propose a culture-based method for sharing learned knowledge between agents based on a new interpretation of the memes metaphor. Here, we consider a meme as a possibly useful piece of knowledge about the structure of the agent. This knowledge is the result of learning process of all agents in the society. Based on how well they have learned the structure, that knowledge is the optimal or close to the optimal structures. We store these memes in a place shared by all agents. This place is called *culture*. In the experiment section, we see that our memetic algorithm can improve lifetime performance of agents.

As stated in the Section III.C, memes can be interpreted and implemented in different ways. The common interpretation of a meme in the evolutionary computation community is a local search before/after applying genetic operators in an evolutionary algorithm (e.g. [Krasnogor05]). This approach is similar to our hybrid learning/co-evolution method that was described in Section V.B. Nevertheless, the idea of a meme "as a unit of information that reproduces itself as people exchange ideas" [Dawkins76] may be interpreted and implemented in other ways too.

In our new interpretation, we consider a society of evolving behavior modules and corresponding agents (as discussed in Section V.B). We give a set of behavior modules to an agent and let it start a life. During its lifetime, the agent learns how to organize those behavior modules in its architecture. The final learned structure is usually the optimal or close to the optimal solution for that set of behavior modules. Instead of throwing away this learned structure, we can store it in a commonly shared place called *culture*. Other agents pick their initial structure from this set of possibly useful solutions, and then start fine-tuning that structure by the usual learning procedure (Section V.A). This initial knowledge may potentially accelerate the learning process by giving the agent a good initial guess about the possible optimal structure.

A priori knowledge provided by the culture can be helpful if all the agents use almost the same sets of behavior modules. This is not an unrealistic assumption as all the agents try to find a solution to the same problem. Nevertheless, we should point out that if the solution space is multi-modal and the population is diversified around distant optima, a good structure for a specific set of behavior modules might no longer be a good guess for another set of behavior modules, and reusing a previously helpful structure might not be very helpful anymore.

To be more precise, consider a set of agents $\{agent_i\}$. Each agent in our architecture is defined by two sets of components: behavior modules and structure, i.e. $agent_i : (\{B_i\}, T_i)$. During the agent's lifetime, the agent tries to find a structure that maximizes its received reward (5). After the agent's lifetime, the result of learning is an ordered pair $(\{B_i\}, T_i^*)$ in which T_i^* is the optimal (or close to the optimal) structure.

We define the culture as a meme pool $\mathbf{M}$ where several different learned structures (memes) T_i^*s for different agents' experiences are stored. Not all memes of a culture are the same. Some of them lead to fitter agents (by giving them a better initial knowledge) and some do not. We desire that fitter memes, which are more likely to be better initial guesses for our agents, have a greater chance to dominate the culture. Those memes should be more stable and be selected more often as the initial structure of the agents. Therefore, we associate a measure of fitness f_{T_i} to each meme T_i. In summary, our meme pool is defined as the following set:

$$\mathbf{M} : \left\{ \left(T_i^*, f_{T_i^*} \right) \right\} \tag{20}$$

A meme's fitness f_{T_i} is defined as the fitness of the agent with the final structure T_i (14). To calculate f_{T_i}, we may average over all the agents that have the same resulting structure over different trials and generations:

$$f_{T_i} = \frac{1}{N} \sum_{A_i \{\{B_i\}, T_i\}} f(A_i) \tag{21}$$

in which N is the number of agents with the structure T_i. Adding a forgetting factor helps us deal with the intrinsic non-stationary nature of the evolutionary approach and/or the environment:

$$f_{T_{i_{n+1}}} = (1 - \alpha_{T_i}) f_{T_{i_n}} + \alpha_{T_i} f(A) \qquad A : (\{B_i\}, T_i) \tag{22}$$

The meme pool has a limited size – the cultural diversity is finite. If it has enough capacity, adding newly generated memes is trivial. If the result of the learning is already in the meme pool, its fitness is updated according to (22). Finally, if there is a new meme with a high fitness, it replaces the least fit meme, i.e. a meme T_i with the lowest f_{T_i} in $\mathbf{M}$. Metaphorically, this is somewhat similar to human culture

where old and unsuccessful traditions are gradually replaced by new and successful ones. Moreover, as time passes, the prominence of a tradition changes gradually according to its usefulness to the current condition of the society.

If the meme pool $\mathbf{M}$ is not empty, the newborn agent acquires a meme according to the fitness f_{T_i} of each meme. Otherwise, the agent picks a randomly chosen structure. In this paper, we have implemented a proportional scaled fitness selection mechanism as our meme selector, i.e. scaled roulette wheel selection mechanism. The agent starts interacting with the environment with the selected meme as its initial structure. If the agent receives reward, the hypothesis that this structure is suitable strengthens. If it receives punishment, the agent changes the structure using our proposed structure learning method (Section V.A.). Also the agent explores its structure space by occasionally selecting randomly chosen structures. In this way, the chance of getting trapped in a local optimum of the structure space decreases.

The culture-based meme transferring mechanism conveys the learning experience of a generation to the next one. This may induce some unwanted bias. Nevertheless, our approach has several stochastic elements that help with escaping from local minima. One element is the randomness in the structure learning procedure that we have just described, and the other is randomness in the genetic operators. These reduce the chance of undesirable effects of the culture-based approach.

To summarize, in our behavior-based system design framework the agent uses three sources of adaptation for its development process. The first is a cooperative co-evolutionary mechanism in which agents' behavior modules are evolved in different genetic pools. Genes compete with each other according to the fitness of the resulting agents. In the second mechanism, the culture acts as initial knowledge to the newborn individuals by providing memes. The meme is the result of the agent's ancestors' experiences. The third mechanism is the structure learning process. This process adapts each individual structure in the direction of maximizing the agent's performance given a set of inherited behavior modules. The agent starts its search from a point that its culture suggests.

VI. EXPERIMENTS

To show the effectiveness of our methods, we apply them to two benchmark problems. One of them is a carefully designed general abstract problem and the other is a simulation of a real-world multi-robot object-lifting task that has been solved and tested empirically using a hand-designed SSA [Nili01]. The abstract problem is used to show the effectiveness of the proposed methods in a controlled, and well-defined situation, while application of our methods in the object-lifting task shows their power in real-world and complex situations. In both problems, we compare the performance of behavior co-evolution/fixed structure with behavior co-evolution/structure learning. Whenever we use structure learning, we compare the performance when (1) there is a meme-induced initial knowledge and when (2) the agent starts from a random initial structure. Also we examine the effect of our two proposed fitness sharing mechanism on the performance of the agent. Before discussing the results, it should be mentioned that we have not optimized parameters of the learning and evolutionary methods (e.g. learning rate, exploration rates, population size, etc.); therefore, even better results are possible.

A. Abstract Problem

The goal of the abstract problem is to show the ability of the agent to learn an optimal policy by interacting with a general random environment. The abstract problem is specified by a random finite state/finite action "desirable" policy $a^* = \Pi(s)$ in which $s \in \{1,2,...,m\} \times \{1,2,...,n\}$ and $a^* \in \{1,2,...,q\}$. This desirable policy is essentially a randomly generated integer matrix.

The goal of the agent is to find a policy π^{agent} that has the minimum distance to $\Pi(s)$ by interacting with the environment. Since the agent's policy depends on the evolved behavior modules and learned structure, this problem can be thought of as finding the right structure and set of behavior modules by structure learning/behavior co-evolution processes to estimate an integer-valued matrix.

A.1. Performance Measure and Reinforcement Signal

The distance in the action space is defined as

$$\left\|\Pi(s) - \pi(s)\right\| = \begin{cases} 0 & \text{if } \Pi(s) = \pi(s) \\ 1 & \text{otherwise} \end{cases} \tag{23}$$

and the total distance over all the state space is

$$\left\|\Pi - \pi\right\| = \sum_{s \in S} \left\|\Pi(s) - \pi(s)\right\|. \tag{24}$$

A normalized measure of distance between the learned policy and the optimal solution can be defined as

$$P(error) = \frac{1}{|S|}\left\|\Pi - \pi\right\| \tag{25}$$

which is a good indicator of the system's performance whenever the probability distribution of visiting states in the problem space is uniform. Note that the agent does not know the optimal policy and must learn it, i.e. it cannot calculate (25).

The agent starts with a randomly chosen architecture T in which behavior modules are selected from a behavior module repertoire. The repertoire consists of behavior modules selected from corresponding behavior (genetic) pools. The agent confronts with a random state s and responds with action a based on the behaviors' organization and each behavior module's policy $B_i(s_i)$. If the agent chooses the correct action compared to the optimal solution $\Pi(s)$, it receives +1 reward and otherwise it receives a -1 punishment:

$$r(s) = \begin{cases} +1 & \text{if } \Pi(s) = \pi^{agent}(s) \\ -1 & \text{otherwise} \end{cases} \quad (26)$$

where $\pi^{agent}(s)$ is the selected action of the agent considering all behavior modules and the organization of them in the architecture. Each such interaction defines an episode in the abstract problem. The goal of structure learning problem is finding an agent that minimizes (25). This problem is equivalent to maximizing the average received reward. Therefore, we define the agent's fitness as the average of the reinforcement signal received during several interactions with the environment.

A.2. Experiment Setup

In our experiments, we have chosen a 5x5 problem space with seven actions. This means that the optimum solution is an element of a space with size $7^{25}(\approx 10^{21})$. We define seven behavior modules B_i $(i = 1,...,7)$ each of them excites everywhere in the problem space. Each behavior module B_i's output is selected from the set of $\{a_i, NA\}$ (i.e. behavior module B_i can only selects either a *NA* or a_i). It is evident that to find Π, each behavior module must evolve and find the correct mapping $B_i(s'): S_i' \rightarrow A_i'$ which selects a_i whenever $\Pi(s) = a_i$. Note that it is not always necessary for behavior module B_i to select *NA* in cases that $\Pi(s) \neq a_i$. For instance, if B_j is in an upper layer of B_i and $\Pi(s) = a_j$, no matter what $B_i(s')$ is, the agent outputs correctly if B_j become the controlling behavior. Note also that the search space of our behavior module decomposition is much smaller than the original space (order of $2^{25}(\approx 10^8)$ for each behavior module).

As all behavior modules have the same input space and all of them can become activated in all states of that space, there exists a mapping $B_i(s'): S_i' \rightarrow A_i'$ for each combination of behavior modules T that solves the problem optimally. In other words, the correct mapping can ultimately be found using a proper evolutionary mechanism regardless of the organization of those behavior modules - if all necessary behavior modules are in the architecture. However, this mapping depends on the organization of behavior modules and if the structure frequently changes, the performance of the agent would degrade. Therefore, a good way to solve the co-evolutionary problem in this experiment is fixing the structure and evolving behavior modules. Note that the method is not aware of this special property of the abstract problem.

In our simulations, we have seven behavior pools for our behavior modules $\{B_i\}$; $i = 1,...,7$. Each behavior module B_i has a 25-dimensional input space. The population size of each behavior pool is 30. In order to estimate the fitness of any individual in each behavior pool, we pick a bunch of behaviors from all behavior pools according to their fitness, and then evaluate the agent's fitness. We do this procedure 300 times in each generation. Having these agents' fitness, we estimate behavior modules' fitness using equations (15) or (17) for the uniform fitness sharing mechanism and valued-based fitness sharing mechanism, respectively. To evaluate each agent's fitness, we let it interact with the environment. Here interaction means that a random state from the optimal policy matrix is given to the agent, and depending on its guess, the agent will receive a reward +1 or punishment -1. The agent has a lifespan of 100 episodes. During those episodes, it learns the correct structure using the proposed structure learning method in the experiments where the structure learning is present, or it just evaluates the performance of the current behavior modules set. In cases that we use our memetic algorithm, we set the culture size (the cardinality of $\mathbf{M}$) to 5. Results reported in this experiment are the aggregation of three runs. Other details of the experiment are shown in Table 1.

A.3. Experiment Results

Uniform vs. Value-based Fitness Sharing Mechanism: Our first result is depicted in Fig. 5. It compares the performance of uniform and value-based fitness sharing mechanism. The reported performance measures are the average fitness of the population and the fitness of the best individual in the population. The optimal solution, which selects the correct answer every time, and the random solution, in which all behavior modules B_i select randomly between action a_i and *NA* with the same probability, are also shown. In this figure, we do not use a meme as an initial knowledge of the structure and all structures are selected randomly at the beginning of the agent's lifetime. Note that all results of this paper report the fitness of the agent and not the fitness of behavior modules. This makes comparison between different fitness sharing mechanism meaningful.

It is seen that the evolutionary method that uses the value-based fitness sharing performs better than the one that uses the uniform fitness sharing. The best solution of the value-based method reaches close to the optimal solution. The average fitness of the random behavior architecture is much lower than others and is close to the performance of the first generation of evolving systems.

The fitness of the random case can be calculated in two ways: analytical and numerical. In the numerical method, we simulate an architecture with random behavior modules and structure several times. The result shown in the figure is obtained in this way. The average fitness can also be calculated analytically. If we define random behavior module as a behavior B_i which selects *NA* and a_i with the same probability:

$$P(a_i = B_i(s)) = P(NA = B_i(s)) \quad \forall s \in S \quad (27)$$

the expectation of the received reinforcement signal is

$$\underset{\pi_{random}}{E}[r] = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2^i}\times(1)+\left(1-\frac{1}{2^i}\right)\times(-1)\right) = \frac{1}{m}\left[2\sum_{i=1}^{m}\frac{1}{2^i}-m\right] \quad (28)$$

where m is the number of layers (behavior modules). The first term in the parentheses (including $1/m$) shows the average received reward from the i^{th} layer. Now, consider the i^{th} layer. In order to have the correct response, its higher behaviors must choose *NA* (each with probability of $1/2$) and the i^{th} layer must select a_i (with probability of $1/2$). The second term in the parentheses shows the amount of punishment signal received by each layer. This average fitness is -0.7165 for 7-layer architecture.

Superiority of the Value-based Fitness Sharing Mechanism: The reason for the superiority of the value-based fitness sharing method to the uniform counterpart in this problem is the ability of the former method in extracting more useful knowledge from the agent's experiments. In this problem, $V_{T(B_i)}$ (16) estimates the contribution of each behavior module in the overall performance of the agent precisely. The uniform fitness sharing neglects this knowledge by assigning the same fitness to all behavior modules.

This is possible because the agent's fitness is the summation of each behavior module's contribution to the fitness (which is captured by the value-based fitness sharing mechanism). In this specific problem, any increase or decrease in the behavior module's fitness has direct effect to the agent's fitness. In other words, if the fitness of a behavior module, as defined in (17), were increased/decreased, the fitness of agents that have this behavior in their architecture would increase/decrease in average (14). In this problem, it is not possible that the increase in the fitness of a behavior module B_i decreases the fitness of another behavior B_j, i.e. they do not compete with each other. This property is not common in all problems, so this superiority of the value-based method may not be seen in all the cases. We see that our object-lifting problem is an example of such a problem (Section VI.B).

The question of which of these two fitness sharing mechanism works better depends on the problem and the way behavior modules are designed. If they do not usually compete with each other for resources (i.e. they become excited in different regions of state space or their output space are separate), we can expect that the value-based fitness sharing mechanism perform better. On the other extreme that they always compete with each other, the uniform fitness sharing mechanism may perform better.

Learning and Co-evolution (Uniform) - Meme vs. no Meme: We perform a series of experiments to investigate the effect of hybridisation of learning and co-evolution. In Fig. 6, we compare the performance of fixing the structure (hand-designed structure), learning the structure, and learning the structure with the help of our memetic algorithm when behaviors are co-evolved using the uniform fitness sharing mechanism. It is important to note that the fixed structure case needs designer's prior knowledge about a good solution of the behavior organization problem. Therefore, the fixed structure case and the learning structure cases are not directly comparable because they do not use the same amount of a priori knowledge. In this paper, we report the result of fixed structure case along the learning cases because it shows the extent to which the structure learning alone can come close to the knowledge the designer might have a priori in some cases.

Based on Fig. 6, the fixed structure case performs better than structure learning methods in early generations but its performance is lower after generation ~110. Note that in this problem, the correct structure is not unique, but it depends on the ordering of behavior modules in the architecture. Therefore, fixing a structure confines behavior modules to be evolved for that specific configuration and reduces the unnecessary (in this case) exploration in the solution space. As a result, the fixed structure performs better than the other methods in the early stages of evolution. However, after decreasing the high amount of diversity in the behavior pools, the correct structure for the given set of behavior modules is learned. It is also seen that the case that uses meme-induced prior knowledge performs better than the case without that knowledge in the later generations. This shows that estimating the correct structure and starting from the culture in the learning process is beneficial for the agent. In addition, it shows that the behavior pools have almost converged to similar solutions; so sharing the structure knowledge through the meme pool can usually help in increasing the agent's fitness.

In Fig. 7, the histogram of the probability distribution of the agents' overall fitness in several sample generations is shown when the uniform fitness sharing is used. This figure shows the empirical distribution of fitness among all agents that are produced using evolved behavior modules in a generation. In other words, if one randomly picked a set of behavior modules from genetic pools in a generation, say generation number 100, and made an agent with them and let the agent learn the correct ordering of behavior modules based on the received reinforcement signal, the fitness of the agent would follow this distribution. For instance, the probability of having a fitness greater than 0.5 is a little less than 0.1 for the meme-induced method and a little more than 0.1 for the fixed structure. These results are based on the aggregation of agents' fitness in three runs.

The empirical cumulative probability distribution of the agent's fitness is shown in Fig. 8 (formally, it is $P(\text{Agent's Fitness} \le \varphi)$). The right-side tendency of these diagrams shows the higher chance of having high performing evolved/learned agents. It is seen that in the first generation, all agents are performing poorly. Those agents that use learning are slightly better than those with the fixed structure because they can find a structure that gains more from those randomly generated behavior modules. In the later generations, the diagrams tend to the right-hand side, which shows that more capable agents are being produced. In early generations (e.g. generation 20), the fixed structure case is

superior, but after a while all of them become almost the same and eventually agents that use structure learning perform better (generation 200). As before, meme-induced agents outperform all others in the late stages of evolution. The reason is that it can benefit from previous experience of other agents with similar set of behavior modules. These results are based on the aggregation of agents' fitness in three runs

Learning and Co-evolution (Value-based) - Meme vs. no Meme: A similar comparison for the value-based fitness sharing is shown in Fig. 9. It can be seen that the case with a fixed structure reaches very close to the optimal solution. Those cases that learn the correct structure, and do not benefit from designer's a priori knowledge about the special structure of this problem, perform somewhat worse than the fixed case, but their performances are still significant. Meme-induced agents perform better than those without it. Fig. 10 and Fig. 11 show the empirical probability distribution and the empirical cumulative probability distribution of the agents' fitness during some sample generations respectively. It is seen from those figures that the fixed structure case is superior in the early generations, but the difference reduces in the latter generations. In addition, the performance difference between with-meme and without-meme agents is noticeable.

Steep plots in Fig. 11, especially in later generations, show that the produced agents' fitness is very close to one. As the fitness of "one" corresponds to the global optimal solution of the abstract problem, this figure shows that most of the produced solutions are very close to the global optimal of the problem.

Summary: In this experiment the value-based fitness sharing mechanism estimates the fitness of each behavior module precisely. Having such conditions is not common for all problems. Also for this problem, we can always find a behavior module's mapping for every ordering of behavior modules in the structure. In this case, whenever the structure learning method suggests a change in the structure, it would disturb co-evolution of behavior modules. Meme transfer reduces this disruptive effect of learning by suggesting initial structures that had performed better with the current sets of evolved behavior modules. This effect is more prominent in the later stages of evolution where all behavior modules in a single behavior pool are more or less similar.

B. Multi-Robot Object Lifting

Although the abstract problem shows the efficiency of our proposed methods, it is desirable to tackle a real-world problem by our methods and study their performance. We chose a cooperative multi-robot object-lifting problem as our second test-bed. Hand design of behavior-based controller for those robots was very difficult, but it resulted in a very successful controller for both simulations and real-world experiments [Nili01]. Nili *et al.* hand-designed the behavior modules and structure by exhaustive trial and error on a real-world setup. In this experiment, we simulate the cooperative object-lifting task based on the kinematics of the robots and the object as introduced in [Nili01]. Before discussing the details of our method, we briefly introduce the problem. For more information, see [Nili01].

B.1. Introduction to Multi-Robot Object Lifting

Imagine a situation in which a group of robots must lift a bulky and large object (Fig. 12). The object is of such a size and shape that none of the robots can grasp it directly. Then, a fork lifting mechanism is suitable for handling the object. When lifting the object, based on the relative position of each robot to the object's centre of gravity, the required force may vary from one robot to another, which introduces heterogeneity in the robots' team. Consequently, each individual robot must have the ability to do its job under a range of external loads. In addition, when the object is tilting, each robot must move to prevent sliding at its contact point with the object. If some compliance is provided at the end effector (in the plane parallel to the object's lower surface) and the tilt angle of the object is kept small enough, there is no need for the robot to move while lifting the object. Keeping the inclination angle of the object within a specified range will also prevent collision between the object and the lifting robots.

To keep the object stable when lifting or moving fast on a rough or curved path, the object configuration must be such that the Zero Moment Point (ZMP) remains in the closed area having the object/robot contact points as its vertices. Considering the object's maximum acceleration and possible position of its centre of gravity, one can find the object's angle at the point when the ZMP comes to the border of the supporting area. If the robots keep the object's angle in the range obtained from the above estimation, the object will be stable. Therefore, if the tilt angle of the object is maintained within a specified value, then the robots are not required to move, the object will not hit the robots, and the system is stable. Moreover, it has been assumed that each robot is capable of measuring the object's angle in its own coordinate system, which can be estimated by each robot.

In the followings, we denote $z(k)$ as the height of the robot-object contact point, $v(k)$ as its elevation velocity, and $\tau(k)$ as the object's tilt angle at time step k (all of these quantities can be measured locally, see [Nili01] for details). Hand-designed behavior modules and structure, whenever they are used, are selected similar to [Nili01] (See Table 2 for the description of hand-designed behavior modules). Results of this paper are obtained with $\Delta v = 1, v_{\max} = 5$, $z_{goal} = 3$, $\tau_0 = 5^\circ$, and $\Delta T = 0.005$ as the simulation time step.

In summary, the problem is cooperative lifting of an unknown object to a set-point while keeping the object's tilt angle small with no central control or communication among the robots. We compare cooperative co-evolution of behavior modules with a priori known structure and without that knowledge (the structure learning case). In the latter case, the agent should learn the structure of behavior modules during its lifetime. In our experiments, we study the performance of both fitness sharing methods (uniform and value-based). Moreover, the effect of the proposed memetic algorithm on the performance of learning is investigated too.

B.2. Performance Measure and Reinforcement Signal

There are two major methods for formulating the optimal solution for a learning system. The first one is defining the solution directly –as it is done in supervised methods- and the second method is coding it in the reinforcement signal. The first method is not possible in many practical situations. Therefore, indirect formulation of desired properties of the solution in the reward function is the only practical way. Nevertheless, one of the most important problems in reinforcement learning is the problem of reinforcement signal design ([Dorigo94], [Dorigo97], and [Ng99]). The problem can be quite difficult whenever the task under investigation is complex and the agent must satisfy a few objective functions together. However, reinforcement signal design is in general much simpler than defining the goal state. There is no general rule to design an appropriate reinforcement function that satisfies the designer's desiderata; and in most problems it is selected by trial and error. We have found that the reinforcement signal (29) (in Table 3) satisfies our goals[*]. This reinforcement signal is designed in a way that reflects the objectives of our problem, and is similar to the way a designer would design the hand-designed controller.

It is notable that the reinforcement learning community generally believes that the reward function is the most robust and transferable definition of the task [Ng00]. Therefore, spending some time to find a suitable reinforcement signal and then letting the agent learn the correct policy seems to be a better choice than trying to explicitly design a non-transferable policy.

Explaining the Reinforcement Signal: In order to clarify the reinforcement function (29), we discuss its terms. Relation (30) rewards reducing tilt angle and punishes a movement that increases it. This part of the reinforcement signal specifies the goal of keeping the object's tilt angle small. Relation (31) rewards robots being in small tilt angle and punishes large tilt angles. Note that it rewards low tilt angle in early stages of the episode more than in late stages and punishes high angles in the later times more than in the beginning in order to enforce quick convergence to a satisfactory angle. Relation (32) rewards the robot's closeness to the goal and punishes the robots far distance from the goal and relation (33) punishes the robot's passing the goal. It is obvious that these two parts of the reward function force satisfying the other goal of our problem which is moving the object to a specified height and keeping it there. And at last, relation (34) punishes a behavior that makes the robot move too fast.

Fitness and Lifetime Fitness: To evaluate the performance of different methods, we compare their obtained reinforcement signals. Two different overall fitness measures are defined: 1) fitness of the agent in the last few episodes of its lifetime and 2) fitness of the agent in all episodes (*lifetime fitness*). The first measure shows the ultimate performance of the agent while the second one shows its performance during the whole lifetime. It is desirable that these two measures have large values and be close to each other. If the lifetime fitness is much smaller than the other, it shows that the agent performs badly in its early lifetime, but ultimately the agent learns how to organize its behavior modules correctly.

We described the way each behavior module's fitness is calculated using one of the proposed fitness sharing methods in Section V.B.1. In the uniform fitness sharing method, the average fitness of the agent in the last few episodes (14) is given to all behavior modules (15). In the value-based fitness sharing, $V_{T(B_i)}$ (16) is given to behavior B_i according to (17). In computing $V_{T(B_i)}$, we use a weighted average of reinforcement signal weighted by discount factor $\alpha_{episode}^{str}$ like the one we defined in (12). In this way, all episodes have effect on the behavior module's fitness, but the effect of later episodes are more important. Note that with our choice of $\alpha_{episode}^{str}$, this measure is not exactly the same as the average reinforcement signal of the last few episodes but is close to it. Nevertheless, the agent's fitness is calculated as before, i.e. the average performance of the agent over the last few episodes.

B.3. Experiment Setup

We make five genetic pools, each for a family of behavior modules, i.e. {"push more", "do not go fast", "stop at goal", "hurry up", "slow down"}. State and action spaces of behavior modules are defined as in Table 4.

The crossover rate p_c is fixed in our experiments, but mutation rates ($p_{m_{hard}}$ and $p_{m_{soft}}$) are decayed during evolution. We turn mutation off in the last few generations in order to reduce the noise in the fitness values.

In our experiments, each genetic population has 20 individuals, and we pick 200 random sets of behavior modules in each generation to evaluate the fitness of behavior modules (the distribution of this random selection is according to the fitness of behavior modules). This means that each instance of behavior module is involved in an average of 10 different agent architectures. This is the number that is used in equations (15) and (17) (though we use the exact number of times each behavior instance has been involved and not the average number).

Each learning agent has 25 episodes to find a suitable structure. The fitness is defined based on the average of the reinforcement signal for the last five episodes. This fitness is directly used for the uniform fitness sharing mechanism. For the value-based fitness sharing mechanism, we use values that are obtained during learning ($V_{T(B_i)}$) as in (16) to approximate the fitness. In the case that the agent does not learn the structure (i.e. hand-designed structure), we give 5 episodes for estimating the fitness of the agent. In this case, fitness and lifetime fitness are the same. Note that this is just fitness (and not lifetime fitness) that enforces selection pressure.

[*] Reinforcement signal applies to the current robot. Therefore, values like $z(k)$ and $v(k)$ must be interpreted as the value of the position and velocity of the current robot's contact point, respectively.

For our memetic algorithm, we use a small culture size of five. Also as a reminder, the results of this experiment are the aggregation of fitness during several runs with different random seeds. Details of the experiment setup can be found in Table 1.

B.4. Experiment Results

Our first result on the performance of our methods for object-lifting task is depicted in Fig. 13. The figure shows the average of the population's fitness (vertical axis) through generations (horizontal axis). Remember that the agent's fitness is the sum of reinforcement signals that it receives in its last five episodes. It is notable that in this figure, we do not present the best agent of each generation as they are very close to the hand-designed cases from the early generations.

General Comparisons (Co-evolution vs. No Co-evolution – Uniform vs. Value-based Fitness Sharing Mechanism – Learning vs. No Learning): In Fig. 13, we compare the performance of the agent that uses (1) hand-designed behaviors/hand-designed structure, (2) hand-designed behaviors/learning structure, (3) behavior co-evolution with the uniform fitness sharing mechanism/learning structure, and (4) behavior co-evolution with the value-based fitness sharing mechanism/learning structure. We expect that if the co-evolutionary mechanism performs well, the fitness will increase through generations.

Let's emphasize that the hand-designed behaviors and/or structures use extra knowledge provided by the designer. This knowledge, which is a *partial solution* of the problem, is not usually available. Therefore, directly comparing these cases without noting their fundamental difference is not fair. Nevertheless, as a reference measure, we provide the result of hand-designed behavior module and hand-designed (fixed) structure case as a flat line. In this case, nothing is evolved or learned. This flat line shows what we would achieve if we rely on our human designer. Subjectively, a human observer evaluates the performance of this setting as quite compelling since it always achieves our goal smoothly. This design is the same as the one that has been done for the same task with real-world robots in [Nili01].

Learning vs. Hand-Designed Structure: The other case is where we use the set of hand-designed behavior modules, but we let the agent learn the structure itself. Here, we do not have any evolutionary mechanism, so the result is again shown as a flat line. We observe that the average fitness of the learning agent (with pre-designed behavior modules) is very close to the hand-designed agent. This is interesting because even though the agent has less prior knowledge (the knowledge of "correct" structure is not given to it), it can perform competitively. Note that the results' variance is higher in the structure learning/hand-designed behaviors compared to the fixed structure/hand-designed behaviors. The source of the extra variance is the learning procedure. The learning procedure starts from an initial structure that is most probably not the same as the optimum one; so the agent suffers some punishments before learning the optimum or close to optimum structure. This produces variance in the fitness. Also the structure learning does not necessarily find the optimum structure by the end of the agent's lifetime, and this, too, adds some extra variance.

Co-evolution and Learning vs. Human Design: By studying behavior of co-evolution/structure learning in Fig. 13, we observe that even though they use less prior knowledge compared to the hand-designed cases, they achieve comparable fitness. Indeed, the performance of co-evolutionary method with the uniform fitness sharing mechanism approaches the performance of the hand-designed solution very fast.

Superiority of the Uniform to the Value-based Fitness Sharing Mechanism: The other observation is that in contrast with the abstract problem, the performance of the uniform fitness sharing is better than the value-based one. The reason will be clearer if we give an example.

Suppose at some moment of time, the agent's structure is T = [... PushMore Stop] ("Stop" at the top layer, and "Push more" in the second layer). Also assume that by initialization or by the process of cooperative co-evolution, these two behavior modules are the same as the hand-designed behavior modules as defined in Table 2.

The definition of "Push more" behavior module in Table 2 implies that it is always activated, so it does not let lower layers' behavior modules become the controlling behavior of the agent. In this special structure and behavior modules setting, either "Push more" or "Stop" would control the agent no matter what other behavior modules there are in the lower layers of the agent. The consequence is that the value of all behavior modules $V_{T(B_i)}$ except these two would not change during learning. Therefore, value-based fitness sharing mechanism would not assign a meaningful fitness to any behaviors except these two (See (17)).

Whenever the initial tilt angle of the object is not very large, these two behavior modules can indeed stabilize the object and receive a good amount of rewarding signal (though not the maximum possible amount), which leads to rather high fitness values for them. Because they are not punished and even they are rewarded, they do not try to evolve in such a way that other behavior modules find the possibility to become activated in this specific structure (e.g. this can be done by some changes in "Stop" behavior module). The The result is that there would be no selection pressure for all other behavior modules, and no teamwork would be encouraged.

In summary, the value-based fitness sharing leads to somewhat selfish behavior modules. If this selfishness produces conflicts among them, it may hurt the agent's performance. In some cases, however, all behavior modules can cooperate to solve a problem with little adversarial effect on others (the abstract problem is an example of this case). The same effect is seen in multi-agent credit assignment with AND-Type and OR-Type tasks [Harati07].

Uniform Fitness Sharing Mechanism - Meme vs. No Meme: Now we study other aspects of our methods in more details. In Fig. 14, we study the effect of using/not using memes whenever behavior fitness is assigned by the uniform fitness sharing mechanism, and we compare the average

fitness and lifetime fitness of the agent in different situations. As before, we depict hand-designed behavior modules with/without structure learning for comparison purposes (the fitness and lifetime fitness of hand-designed behavior/structure is the uppermost solid line; the fitness of the hand-designed behavior/learning structure is the second uppermost solid line, and its lifetime fitness is the dashed line).

We see in Fig. 14 that in the last five episodes, average fitness of all behavior evolution cases is almost the same and is close to the hand-designed behaviors/structure case (fitness curves are shown in solid lines). This shows that the co-evolutionary mechanism can actually find good solutions for the behavior optimization problem. Also the average performance of the cases "with structure learning" is almost the same as the average performance of the hand-designed structure. This shows that the structure learning can find a good solution to behavior organization problem.

Effect of Meme on Lifetime Fitness: Comparing lifetime fitness of different methods, we expect a drop in lifetime fitness whenever the agent should learn the structure. This is because whenever we are learning, we are exploring different structure possibilities (especially in early stages of learning), and many of those possibilities may not be so good. However, comparing the lifetime fitness of "with meme" with "without meme" cases show an interesting phenomenon: the meme-induced case performs much better than the case without meme (these are shown in dashed lines). In fact, having a meme pool prevents wasting useful knowledge learned by other agents. Note that in the meme-induced case, the designer does not use any a priori knowledge about the correct structure of the agent, and the method itself benefits from other agents' previous experiences.

In Fig. 15, we show the histogram of the empirical probability distribution of the agent's fitness in a few sample generations when we co-evolve the behavior modules and use (1) hand-designed structure, (2) structure learning without meme, and (3) meme-induced structure learning. In this figure, we use the uniform fitness sharing mechanism. Note that the agent's fitness may be different from behavior modules' fitness when we use the value-based fitness sharing mechanism, but in this case both of them are the same[†].

We see that in the early generations (e.g. Generations 1 and 5 in Fig. 15) those cases that learn the structure perform a bit better than the one with a hand-designed (fixed) structure. The reason is that the structure learning method can exploit any available information that comes from mere randomness (in Generation 1) or diverse behavior modules (in Generation 5) by re-arranging them in the structure. In later generations, however, a hand-designed structure case produces more highly-fitted agents compared to the structure learning cases. This is intuitive because the hand-designed structure case has more prior knowledge about the problem and also it is easier for the co-evolutionary mechanism to find a suitable behavior for a fixed structure. In spite of that, the performance of agents with structure learning is comparable to the hand-designed structure agents. In Fig. 15, we observe that the meme-induced case generates slightly more highly-fitted agents compared to the case without using a meme.

In Fig. 16, the same kind of histogram is shown for the agents' *lifetime* fitness. The superiority of meme-induced cases to the without-meme case is evident: memetic algorithm increases the chance of producing high-performing agents (lifetime fitness). These results show where our memetic algorithm can be most helpful: *increasing the lifetime performance of the agent which is very important when dealing with real systems*.

The empirical cumulative probability distributions of agents' fitness (solid lines) and agents' lifetime fitness (dashed lines) are shown in Fig. 17[‡]. In the first generation, agents that use structure learning tend toward the right-hand side compared to the hand-designed structure case (for both fitness and lifetime fitness measures). This means that in the first generation, agents that learn the structure (both with and without meme) perform better than the hand-designed structure agents on average.

Comparing the fitness measure, the hand-designed structure gradually outperforms the other two by a slight margin. The difference is much larger for the lifetime fitness since the lifetime fitness considers all early learning episodes where the agent tries to find the correct structure and fails. However, this difference is much smaller for the "with meme" approach compared to "without meme" case. Also we see that the lifetime fitness of meme-induced structure learning agents is much better than those without a meme. This, again, confirms our intuition that we can benefit from using a memetic algorithm to transfer learned knowledge.

Value-based Fitness Sharing Mechanism - Meme vs. No Meme: We do the same kind of comparisons for the value-based fitness sharing mechanisms in Fig. 18-21. In Fig. 18, we show the population average of fitness during generations when the value-based fitness sharing mechanism is used. Again, we compare the fitness of our evolutionary mechanisms with/without meme-induced knowledge with the hand-designed behaviors/hand-designed structure, and hand-designed behaviors/learned structure (without meme). Note that in the following figures we report the agent's fitness and not the behavior's fitness, which is a different measure.

The average fitness of the value-based fitness sharing mechanism is generally smaller than that of the uniform fitness sharing mechanism. We discussed this point when we explained results shown in Fig. 13. Nevertheless, we see the same pattern here: average fitness of meme-induced case, especially lifetime fitness, is higher than the case where we do not use memes.

[†] For more information about this type of figure, refer to the explanation of Fig. 7 in the Abstract Problem section (Section VI.A.3).

[‡] For more information about this type of figure, refer to the explanation of Fig. 8 in the Abstract Problem section (Section VI.A.3).

In Fig. 19, the histogram of the empirical probability distribution of the agent's fitness in a few sample generations is shown. We do comparisons similar to what we did for Fig. 15. The same kind of histogram for the lifetime fitness of the agent is depicted in Fig. 20. Here, we see similar patterns. The performance of the learning agent in early generations is better than the performance of a hand-designed structure agent. Ultimately, the hand-designed structure agents outperform the agents that learn the structure.

We show the empirical cumulative probably distributions of agents' fitness (solid lines) and agents' lifetime fitness (dashed lines) in Fig. 21. The pattern of results is almost the same as in the uniform fitness sharing mechanism case. We observe, however, some slight differences too. For instance, compare the distribution of fixed structure between Fig. 16 and Fig. 20. In Fig. 16, we observe an increasing concentration of probability toward the high-end of the fitness distribution. This means that almost all agents have very high fitness values. On the other hand, in Fig. 20, it seems that there are two peaks of fitness probability (one close to the high-end around 300 and the other around 240). This implies that not all agents have the highest achieved fitness, but there are two groups of agents with significantly different amount of fitness. Similar result can be observed by comparing Fig. 17 and Fig. 21. The curve for the fixed structure is more or less convex in Fig. 17 (especially in later generations), but it is not in Fig. 21. This may suggest that the value-based fitness sharing mechanism leads to clusters of individuals: one cluster with a very high fitness individuals and the other with a somewhat lower fitness values. Whether this observation is statistically meaningful or not needs further investigation.

Learning and Fast Adaptation to the Environment's Changes: In the introductory part of Section III, we explained that changing the structure leads to faster adaptation of the overall behavior of the agent compared with changing the behavior modules. This is very important whenever the agent deals with non-stationary environments. Although we have not intentionally changed the dynamics of the environment or the agent's goal, we can still observe this property in our results.

Consider Fig. 17 and/or 21. These figures show the cumulative probability distribution of fitness in several sample generations. The agent does not have any previous knowledge about the environment in the first generation since all behavior modules are assigned randomly. This situation is like an abrupt change in the environment where the function of behavior modules is almost irrelevant to the appropriate function for the current environment. If we expect that learning the structure is helpful for fast adaptation to a new environment, the agent with structure learning should in general perform better than an agent without it. In Fig. 17, we see that the probability distribution of the case with fixed structure (which is the hand-designed structure) has tendency toward the lower values of fitness compared to the agent with learning. This shows that even where behavior modules are completely irrelevant to the current situation (they are random), the structure learning can extract some useful information by reorganising behavior modules. Similar phenomenon is observable in Fig. 21.

Phenomenal Behavior: Finally, a sample trajectory of robot-object contact positions, the object's tilt angle, and the controlling behaviors are depicted with respect to time in Fig. 22 when Robot 1 is initialized higher than the two other ones. The behavior co-evolution with the uniform fitness sharing and structure learning with meme-induced initial knowledge are used to generate the architecture. In this sample trajectory, Robot 2 and Robot 3 execute the "Hurry up" behavior module in early steps while Robot 1 selects "Slow down" behavior module. However, due to the constraints of that behavior's action space ($A'_{\text{Slow}} = \{\max(v(k) - \Delta v, 0), NA\}$), the robot-object contact point's speed cannot become negative. Thus, Robot 1 stands still until the other two move up and the object tilt angle is reduced. Afterwards, it selects the "Hurry up" behavior module. When the robots reach the goal they execute the "Stop" behavior module.

As discussed in Section II.A, the name of a behavior module comes from the designer's expectation of that module. She determines what kind of sensory information goes to it and what kind of output actions are available to the behavior module. The overall behavior of the agent and the behavior module's contribution to it, however, is not predetermined and depends on the complex interaction of the environment, learning and co-evolutionary processes, and the initial design specifications.

Cooperative co-evolution and learning change behavior modules and the structure to maximize the overall performance of the robot. The emergent behavior of the agent is not necessarily the same as the designer's prior expectation. For instance, Fig. 22 shows that the "Slow down" behavior module (which its input/output spaces are defined in Table 4) actually stops "Robot 1" instead of slowing down any robot. In this case, the designer had expected that the robot would have needed the "Slow down" module when it wanted to move downward. Therefore, the design of the "Slow down" module was such that it could not produce any upward movement. This initial design limitation emerges as a behavior module that can stop the robot even when it was not designed for. This suggests that associating a "behavior name" to a module does not necessarily mean that the module should behave in the same way.

Summary: The results of these experiments show that our hybrid co-evolution/learning behavior-based system design methods can develop a competitive or even superior agent's "mind" (or controller) in comparison to a human-made agent. In addition, our proposed culture-based memetic algorithm significantly increases the agent's lifetime performance, which is crucial for online evolution/learning of interactive agents.

VII. DISCUSSION

In this section, we summarize the important results and discussion scattered throughout the paper.

We used two simulated setups to evaluate our proposed methods and compared them with each other. We also compared our methods to hand-designed or partially guided solutions that came from the designer's knowledge about the problem, which is not fully available in general.

In the abstract problem, our methods performed very well, and the performance was close to the optimal. The performance was particularly better for the value-based fitness sharing mechanism compared to the uniform fitness sharing mechanism.

In the abstract problem, the contribution of each behavior module to the total fitness of the agent is direct, i.e. increasing the fitness of one behavior module increases the fitness of the agent. Therefore, the valued-based fitness sharing mechanism that assigns the performance of behavior module (and not the performance of the agent) as its fitness results in a more accurate fitness evaluation compared to the uniform fitness sharing mechanism. This is the reason for the superiority of the value-based fitness sharing mechanism in this problem. Note that this property is not common to all problems, e.g. the object-lifting problem.

The results for the abstract problem also showed that those agents that use learning are slightly better than those with a fixed structure. This is especially interesting given that for this problem, every structure's arrangement can lead to an optimal solution with an appropriate set of behavior modules. This shows that learning can find the most rewarding structure based on the set of available suboptimal behavior modules. Nevertheless, knowing that any fixed structure is sufficient to solve the problem helps behavior cooperative co-evolution in the early generations. The reason is that fixing a structure produces a selection pressure to specific behaviors' solutions and prevents unnecessary diversity in genetic pools.

Finally, the results of the abstract problem indicated that better agents could be obtained with meme-induced prior knowledge.

In the second experiment, we evaluated our methods in a robotic task. Hand design of this robotic problem was very difficult, but resulted in a very successful controller (see [Nili01]). We use this hand-designed controller as our gold standard. In this experiment, we used a reinforcement signal that reflected our beliefs about the objective of the problem. Needless to say, our method was not especially designed for this specific problem.

The fitness of all behavior module co-evolution and/or structure learning methods with/without meme-induced prior knowledge was satisfactory and comparable to the human designed solution. This suggests that the proposed methods could give us good solutions.

Our results showed that, in contrast with the abstract problem, the uniform fitness sharing was better than the value-based mechanism. The reason is that, in this problem, maximizing each behavior module's fitness does not necessarily lead to the best possible agent, and the value-based fitness sharing mechanism may lead to selfish behavior modules.

In the object-lifting problem, we showed that the final fitness (which is calculated based on the last five episodes of learning) of all methods is almost the same. However, the story was different for the lifetime fitness. The meme-induced cases helped to considerably increase the lifetime fitness of agents. This phenomenon shows that having a meme pool prevents wasting useful knowledge learned by previous agents. This effect was more noticeable in the uniform fitness sharing method.

Finally, we noted that naming a behavior module does not necessarily mean that the behavior is precisely performing our anticipated behavior.

VIII. CONCLUSIONS AND FUTURE WORK

We proposed a general, hybrid, bio-inspired optimization approach for designing modular agents. This approach combines the cooperative co-evolution, reinforcement learning, and a new interpretation of memetic algorithms. Although the main concept of our approach is applicable for different kinds of intelligent agent architectures, we formulated it for the automatic development of hierarchical behavior-based architectures. The co-evolutionary process evolves new behavior modules, the structure learning organizes them in the agent's architecture, and the memetic algorithm shares learned knowledge among the agents.

The key idea of our approach is decomposing the problem into some *potentially* easier sub-problems. Instead of adapting a complete monolithic controller, we co-evolve sets of behavior modules separately and organize them in the architecture. Each of these tasks is potentially easier because the input space dimension of those modules is much smaller than the joint space of all sensors and internal memories of the agent. In our modular approach, the designer can use her prior knowledge for defining the input (state)/ output (actuator) spaces of behavior modules.

Another possible benefit of our approach is the ability of the agent to adapt to changes in the environment. Although we have not explicitly changed the environment to study this phenomenon, it showed itself during our experiments. In the first generation where behavior modules were totally random, the learning agent performed better than a fixed design. This shows that the learning agent can quickly adapt to the environment. If learning was not available, the agent would need to wait for at least one or two generations before the evolutionary process could increase the fitness of the agent.

There are many potential research directions that can be pursued based on our approach. The most evident step is trying it on other interesting and more complex problems. As a proof of concept, we have provided two benchmark examples in this paper and the results are competitive to human-designed solutions. Since our approach has not been especially designed for these problems, we also expect to get good results for other problems. Studying how this approach would scale up to other problems would be fruitful.

Another important research direction is studying the current fitness sharing mechanism more thoroughly and investigating other possible approaches. The value-based

fitness sharing mechanism worked well for the abstract problem because maximizing the fitness of behavior modules was highly correlated to the task of maximizing the agent's fitness. As we discussed in our experiments (Section VI.B.), this is not true for all problems. One possible research direction is studying these two approaches on other benchmarks. A relevant question here is the possibility of devising other fitness sharing mechanisms. The uniform fitness sharing mechanism does not exploit the special architecture of the agent and assigns the same fitness to all behavior modules involved in the agent's architecture. The value-based mechanism tries to benefit from the architecture and learning experience of the agent to assign the fitness of behavior modules. A possible approach to study the fitness sharing problem is the *Collective Intelligence* (COIN) framework. COIN addresses the problem of "designing collective of computational processes to maximize a provided world utility function when each process tries to maximize its own payoff utility function" [Wolpert04]. Here, the world utility function is similar to the agent's fitness and those computational processes are each of our behavior modules. Additionally, tools and approaches like what have been used in [Wiegand02], [Ficici05], and [Popovici05] can be helpful for better understanding of the effect of fitness evaluation methods on the dynamical behavior of our co-evolutionary method. Nevertheless, the analysis would be more difficult because of the effects of the learning and memetic algorithms. Finally, approaches that try to solve *relative overgeneralization* pathology of co-evolutionary methods might be helpful for having more efficient co-evolutionary mechanism [Panait06]. They change the fitness function by biasing it toward optimal collaboration. It is not, however, completely clear how one can estimate the fitness of the optimal collaborator in our framework. This issue needs further investigation.

Estimating the generalization capability of the agent and dealing with noise are two important and relevant issues for situated agents. Because a situated learning agent deals with different types of uncertainties, including the stochastic nature of the environment and the exploratory phase of the learning process, any performance measure would be contaminated by noise. When we want to select an agent that performs well in average, we need to remember that the evaluated fitness is not exactly the same as the agent's expected fitness. More precisely, the empirical fitness has deviations from the expected fitness. A technique like multiple evaluations of the fitness function is considered in this paper (see equations (15) and (17)). There are several methods for handling this uncertainty in evolutionary optimization [Jin05]. A potential research direction is adaptively selecting the number of evaluations based on our confidence on the fitness.

Another important issue is that we want to be sure that if our agent performs well at the training phase, it will maintain a similar performance in the future. This is the problem of generalization and has long been studied in the machine learning community and studied in [Chong08] in the evolutionary optimization context. They provide a probabilistic upper bound on the generalization performance of an individual based on its performance on a validation set. In our method where we evaluate each behavior module for N times, the error in the empirical fitness as compared to the expected fitness decays with a rate of $O\left(\frac{1}{\sqrt{N}}\right)$. See also [Mnih08] for a method called the Bernstein Racing algorithm that suggests an evaluation schedule to select the individual with the highest expected performance among a pool of individuals based on their empirical performance Nevertheless, the precise analysis of the effects of learning and the randomness in fitness evaluation needs further investigation.

Our new interpretation of memetic algorithm was based on the idea that culture is a means to transfer learned knowledge from old individuals to new individuals in the society. Our mechanism of storing and sharing knowledge was simple and minimal. Nevertheless, one may benefit from more sophisticated approaches of defining culture and the way it interacts with individuals in the society.

Extending behavior modules that work with real-valued state/action spaces is another important research direction. Our benchmark problems had discrete state/action spaces. In many problems, however, we are dealing with continuous real-valued sensors and actuators, see [Mobahi07] for detailed discussion, and see [Antos07], [Farahmand08A], and [Farahmand08B] for some mathematically rigorous reinforcement learning-based approaches to deal with continuous states/discrete actions problems. In these cases where we need to work with continuous states and/or actions, we cannot represent the mappings defined by behavior modules in an exact form and we need to use function approximators. These function approximators can be anything from generalized linear models to multi-layer feedforward neural networks. Fortunately, using function approximators does not considerably change our architecture design approach. The only change is the way we need to encode behavior modules as genetic material.

Another possible extension is automating the way state and output spaces are determined. In the proposed approach, we assume that the state and output spaces of each type of behavior modules are given by the designer. She determines which sensors and internal memories are relevant features for achieving a certain agent's behavior. Nevertheless, automating this feature selection mechanism is a very interesting and difficult research direction. One approach for dealing with this problem is through an attention control framework [Fatemi07].

One issue that needs further investigation is the benefit of our special problem decomposition for adapting to abrupt changes in the environment. In the monolithic controller design, the whole controller should be changed so that the agent adapts to the new environment. On the other hand, we showed that, in our approach, structure learning finds the most rewarding organization of behavior modules even if those modules were not specifically designed for the new

environment. This gives us a hint that this phenomenon can be actually true.

We did not observe any divergence in the structure learning. However, an important open question is whether the structure learning mechanism converges to the optimal structure when using estimation of a behavior module's values in other structures as the initial estimation of its value in a new behavior organization, see Section V.A.1.

We should emphasize that the scope of our approach is not limited to PPSSA architecture. One may adopt the concept of decomposing the problem and benefiting from co-evolution/reinforcement learning/memetic algorithm and the presented methods to other agent's architectures. However, the modification needs careful attention, as the same mathematical formulation, especially the structure learning part, may not apply directly in detail anymore.

The last point is that, we showed how interaction of co-evolution, individual learning, and culture helps a society of agents to improve its average fitness. A theoretical analysis on the conditions where these mechanisms would be helpful for solving optimization problems is interesting and important. Moreover, one may benefit from more accurate models of sociological and evolutionary processes for tackling hard optimization problems.

ACKNOWLEDGEMENTS

We greatly appreciate the anonymous associate editor and reviewers for their helpful comments and constructive suggestions; and Elliot A. Ludvig, Volodymyr Mnih, Varun Grover, and Yasin Abbasi for proofreading the paper; and Ramin Mehran for designing 3D graphic model of the object-lifting robots. This project was supported in part by the School of Cognitive Sciences, Institute for Studies in Theoretical Physics and Mathematics. Amir massoud Farahmand was partially supported by the Alberta Ingenuity Centre for Machine Learning.

REFERENCES

1. [Antos07] A. Antos, Cs. Szepesvári and R. Munos, "Value-iteration based fitted policy iteration: learning with a single trajectory," *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 330-337, 2007.
2. [Barto03] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Systems Journal: Special Issue on Reinforcement Learning*, vol. 13, pp. 41-77, 2003.
3. [Baxter01] J. Baxter and P. J. Bartlett, "Infinite-horizon gradient-based policy search," *Journal of Artificial Intelligence Research*, vol. 15, 2001.
4. [Bertsekas96] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dyanmic Programming*, Athena Scientific, 1996.
5. [Brooks86] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robotics and Automation R.A-2*, pp. 14-23, 1986.
6. [Brooks89] R. A. Brook, "A robot that walks: emergent behavior from a carefully evolved network," *Neural Computation* 1(2), 1989, pp. 252-262.
7. [Brooks91] R. A. Brooks, "Intelligence without representation," *Artificial Intelligence*, 47, pp. 139-159, 1991.
8. [Buriol04] L. Buriol, P. M. Franca, P. Moscato, "A new memetic algorithm for the asymmetric traveling salesman problem," *Journal of Heuristics*, 10, pp. 483-506, 2004.
9. [Chernova04]S. Chernova and M. Veloso, "An evolutionary approach to gait learning for four-legged robots," *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2004.
10. [Chong08] S. Y. Chong, P. Tino, and X. Yao, "Measuring generalization performance in coevolutionary learning," *IEEE Transactions on Evolutionary Computation*, 12(4), 2008.
11. [Dawkins76]R. Dawkins, *The Selfish Gene*, Oxford University Press, 1976.
12. [Dayan93] P. Dayan and G. Hinton, "Feudal reinforcement learning," *In Advances in Neural Information Processing Systems (NIPS)*, 5, Morgan Kaufmann, San Francisco, CA, pp. 271–278, 1993.
13. [Dietterich00]T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *Journal of Artificial Intelligence Research*, 13, pp. 227-303, 2000.
14. [Dorigo94] M. Dorigo and M. Colombetti, "Robot shaping: developing autonomous agents through learning," *Artificial Intelligence*, 71(4), pp. 321-370, 1994.
15. [Dorigo97] M. Dorigo and M. Colombetti, *Robot Shaping: An Experiment in Behavior Engineering*, MIT, CA, 1997.
16. [Farahmand05A] A. M. Farahmand, M. Nili Ahmadabadi, B. N. Araabi, "Behavior and hierarchy development in behavior-based systems using reinforcement learning," *Technical Report, Department of Electrical and Computer Engineering, University of Tehran*, 2005.
17. [Farahmand05B] A. M. Farahmand, "Learning and evolution in hierarchical behavior-based systems," *MS. Thesis, Department of Electrical and Computer Engineering, University of Tehran*, 2005 [in Persian].
18. [Farahmand06] A. M. Farahmand, M. Nili Ahmadabadi, C. Lucas, and B. N. Araabi, "Hybrid behavior co-evolution and structure learning in behavior-based systems," In the *Proceedings of Congress on Evolutionary Computation* (CEC), Vancouver, Canada, 2006.
19. [Farahmand08A] A. M. Farahmand, M. Ghavamzadeh, Cs. Szepesvári, and Sh. Mannor, "Regularized policy iteration," accepted for publication in *Twenty-Second Annual Conference on Neural Information Processing Systems (NIPS)*, 2008.

20. [Farahmand08B] A. M. Farahmand, M. Ghavamzadeh, Cs. Szepesvári, and Sh. Mannor, "Regularized fitted q-iteration: application to planning," *European Workshop on Reinforcement Learning (EWRL)*, to appear in *Lecture Notes in Artificial Intelligence,* 2008.
21. [Fatemi07] H. Fatemi, and M. Nili Ahmadabadi, "Biologically inspired framework for learning and abstract representation of attention control," To appear in *Springer's Lecture Note in Artificial Intelligence,* 2007.
22. [Federici03] D. Federici, "Combining genes and memes to speed up evolution," In the *Proceedings of the Congress on Evolutionary Computation (CEC),* 2003.
23. [Ficici05] S. G. Ficici, O. Melnik, and J. B. Pollack, "A game-theoretic and dynamical-systems analysis of selection methods in coevolution," *IEEE Transactions on Evolutionary Computation,* 9(6), pp. 580-602, 2005.
24. [Floreano96]D. Floreano and F. Mondada, "Evolution of homing navigation in a real mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26, pp. 396-407, 1996,
25. [Floreano00]D. Floreano and J. Urzelai, "Evolutionary robotics: the next generation," in: T. Gomi, Ed., Proceedings of Evolutionary Robotics, III, Ontario, pp. 231-266, 2000.
26. [Floreano08] D. Floreano, P. Husbands, and S. Nolfi, "Evolutionary robotics," *in Handbook of Robotics, Springer-Verlag*, 2008.
27. [Ghavamzadeh03]M. Ghavamzadeh and S. Mahadevan, "Hierarchical policy gradient algorithms," *In Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pp. 226-233, Washington, D.C., August 2003.
28. [Ghavamzadeh06] M. Ghavamzadeh and Y. Engel, "Bayesian policy gradient algorithms," *In Advances in Neural Information Processing Systems (NIPS),* 2006.
29. [Gomez97] F. Gomez and R. Miikkulainen, "Incremental evolution of complex general behavior," *Adaptive Behavior*, 5, pp. 317-342, 1997.
30. [Harati07] A. Harati, M. Nili Ahmadabadi, and B. N. Araabi, "Knowledge evaluation for credit assignment among independent q-learners," *IEEE Systems Journal,* vol 1, no1, pp. 55-67, 2007.
31. [Harvey93] I. Harvey, P. Husbands, and D. Cliff, "Issues in evolutionary robotics," in: From Animals to Animats II: Proceedings of *the Second International Conference on Simulation of Adaptive Behavior*, J. Meyer, H. L. Roitblat, and S. W. Wilson, (Eds.), MIT Press-Bradford Books, Cambridge, MA, 1993.
32. [Hinton87] G. E. Hinton and S. J. Nowlan, "How learning can guide evolution," *Complex Systems*, 1, pp. 495-502, 1987.
33. [Jin05] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments – a survey," *IEEE Transactions on Evolutionary Computation*, 9(3), pp. 303-317, 2005.
34. [Kohl04] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.
35. [Kakade02] S. Kakade, "A natural policy gradient," *In Advances in Neural Information Processing Systems (NIPS)*, 2002.
36. [Koza94] J. Koza, "Evolution of a subsumption architecture that performs a wall following task for an autonomous mobile robot via genetic programming," In *Computational Learning Theory and Natural Learning Syste*ms, vol. 2, S.J. Hanson, T. Petsche, M. Kearns, and R.L. Rivest, Eds., The MIT Press, pp. 321-346, 1994.
37. [Krasnogor04]N. Krasnogor and S. Gustafson, "A study on the use of 'Self-Generation' in memetic algorithms," *Natural Computing* 3 (1), pp. 53-76, 2004.
38. [Krasnogor05]N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: model, taxonomy, and design issues," *IEEE Transactions on Evolutionary Computation*, 9(5), 2005.
39. [Krawiec07] K. Kraiwec and B. Bhanu, "Visual learning by evolutionary and coevolutionary feature synthesis," *IEEE Transactions on Evolutionary Computation*, 11(5), pp. 635-650, 2007.
40. [Maes90] P. Maes and R. A. Brooks, "Learning to coordinate behaviors," in *Proc. AAAI-90*, pp. 796-802, 1990.
41. [Mahadevan92]S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, 55, pp. 311-365, 1992.
42. [Matarić92] M. J. Matarić, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, 8(3), pp. 46-54, 1992.
43. [Matarić94] M. J. Matarić, "Reward function for accelerated learning," in: W. W. Cohen and H. Hirsh, (Eds.), Proc. 8th Int. Conf. Machine Learning, Morgan Kaufmann, pp. 181-189, 1994.
44. [Matarić97] M. J. Matarić, "Reinforcement learning in the multi-robot domains," *Autonomous Robots*, vol. 4, no. 1, pp. 73-88, 1997.
45. [Matarić98] M. J. Matarić, "Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior," *Trends in Cognitive Science*, vol. 2, no. 3, pp. 82-87, 1998.
46. [Matarić01] M. J. Matarić, "Learning in behavior-based multi-robot systems: policies, models, and other agents," *Cognitive System Research - special issue on multi-disciplinary studies of multi-agent learning*, Ron Sun, ed., 2(1), pp. 81-93, 2001.
47. [Merz99] P. Merz and B. Freisleben, "A Comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem," in *Proceedings of the Congress on Evolutionary Computation (CEC)*, 1999.
48. [Merz00] P. Merz and B. Freisleben, "Fitness landscape analysis and memetic algorithms for the quadratic assignment problem," *IEEE Transactions on Evolutionary Computation*, 4(4), pp. 337-352, 2000.

49. [Michaud98]F. Michaud and M. J. Matarić, "Learning from history for behavior-based mobile robots in non-stationary conditions," *Autonomous Robots and Machine Learning* AR:5, ML:31, AR:335-354, ML:141-167, 1998.
50. [Mnih08] V. Mnih, Cs. Szepesvári, and J-Y. Audibert, "Empirical bernstein stopping," in *25th International Conference on Machine Learning (ICML)*, 2008.
51. [Mobahi07] H. Mobahi, M. Nili Ahmadabadi, and B. N. Araabi, "A biologically inspired method for concept imitation using reinforcement learning," *Applied Artificial Intelligence*, 21 (3): 155-183, Feb 2007.
52. [Moscato92]P. Moscato and M. Norman, "A Memetic approach for the travelling salesman problem – implementation of a computational ecology for combinatorial optimisation on message-passing systems," In *Proceeding of the International Conference on Parallel Computing and Transputer Applications*, IOS Press, Amsterdam, 1992.
53. [Moscato03]P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," In F. Glover and G. Kochenberger (Eds.), *Handbook of Metaheuristics*, pp. 105-144. Kluwer Academic Publishers, Boston MA, 2003.
54. [Ng99] A. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: theory and application to reward shaping," *Proc. 16th International Conf. on Machine Learning (ICML),* Morgan Kaufmann, San Francisco, CA, 1999.
55. [Ng00] A. Ng, S, Russell, "Algorithms for inverse reinforcement learning," In *Proc. of the Seventeenth International Conference on Machine Learning* (ICML), 2000
56. [Nili01] M. Nili Ahmadabadi and E. Nakano, "A constrain and move approach to distributed object manipulation," *IEEE Transactions on Robotics and Automation*, 17(2), pp. 157-172, 2001.
57. [Nolfi99] S. Nolfi and D. Floreano, "Learning and Evolution," *Autonomous Robots* 7(1), pp. 89-113, 1999.
58. [Nolfi00] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, MIT Press, Cambridge, MA, 2000.
59. [Ong04] Y.S. Ong and A. J. Keane, "Meta-lamarckian learning in memetic algorithms," *IEEE Transactions on Evolutionary Computation*, 8(2), pp. 99-110, 2004.
60. [Panait06] L. Panait, S. Luke, R. P. Wiegand, "Biasing coevolutionary search for optimal multiagent behaviors," *IEEE Transactions on Evolutionary Computation,* 10(6), pp. 629-645, 2006.
61. [Parker98] L. Parker, "ALLIANCE: an architecture for fault-tolerant multi-robot cooperation," *IEEE Transactions on Robotics and Automation*, 14(2), 1998, pp. 220-240.
62. [Parr98] R. E. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," In *Advances in Neural Information Processing Systems (NIPS): Proceeding of the 1997 Conference*, Vol. 10, Cambridge, MA. MIT Press, 1998.
63. [Popovici05]E. Popovici and K. De Jong, "Understanding cooperative co-evolutionary dynamics via simple fitness landscapes," *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2005.
64. [Potter00] M. A. Potter and K. A. De Jong, "Cooperative coevolution: an architecture for evolving coadapted subcomponents," *Evolutionary Computation*, 8 (1), pp. 1-29, 2000.
65. [Prescott99] T.J. Prescott, P. Redgrave, and K. Gurney, "Layered control architectures in robots and vertebrates," *Adaptive Behavior*, 7, pp. 99-127, 1999.
66. [Radcliffe94]N. J. Radcliffe and P. D. Surry, "Formal memetic algorithm," In T. Fogarty (ed.), *Evolutionary Computing: AISB workshop*, vol. 865 of Lecture Notes in Computer Science, pp. 1-16, Springer-Verlag, Berlin, 1994.
67. [Rosin97] C. D. Rosin and R. Belew, "New methods for competitive co-evolution," Evolutionary Computation, 5(1), pp. 1-29, 1997.
68. [Smith07] J.E. Smith, "Coevolving memetic algorithms: A review and progress report," *IEEE Transactions on Systems, Man, and Cybernetics*, 37(1), Feb 2007.
69. [Sutton98] R. S. Sutton, and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.
70. [Sutton99] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, 112, 1999, pp. 181-211.
71. [Togelius04]J. Togelius, "Evolution of a subsumption architecture neurocontroller," *Journal of Intelligent and Fuzzy Systems, 15:1,* pp. 15-20, 2004.
72. [Wang96] Z. D. Wang, E. Nakano, and T. Matsukawa, "Realizing cooperative object manipulation using multiple behavior-based robots," in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, vol. 1, Osaka, Japan, pp. 310–317, 1996,
73. [Whiteson06] S. Whiteson and P. Stone, "Evolutionary function approximation for reinforcement learning," *Journal of Machine Learning Research*, pp. 877-917, 2006.
74. [Wiegand02]R. P. Wiegand, W. C. Liles, and K. A. De Jong, "Analyzing cooperative coevolution with evolutionary game theory," *In Proceedings of the Congress on Evolutionary Computation,* pp. 1600-1605, 2002.
75. [Wiegand04]R. P. Wiegand, "An analysis of cooperative coevolutionary algorithms," *Ph.D. Thesis*, George Mason University, 2004.
76. [Wolpert04] D. H. Wolpert, "The Theory of Collectives," in K. Tumer and D. H. Wolpert (Ed.'s), Collectives and the Design of Complex Systems, Springer-Verlag, 2004.

77. [Ziemke98] T. Ziemke, “Adaptive Behavior in Autonomous Agents,” *Presence,* Vol. 7, No. 6, pp. 564-587, 1998.
78. [Zou04] P. Zou, Z. Zhou, G. Chen, and X. Yao, “A novel memetic algorithm with random multi-local-search: a case study of tsp,” In Proceedings of the *Congress on Evolutionary Computation* (CEC), pp. 2335-2340, 2004.

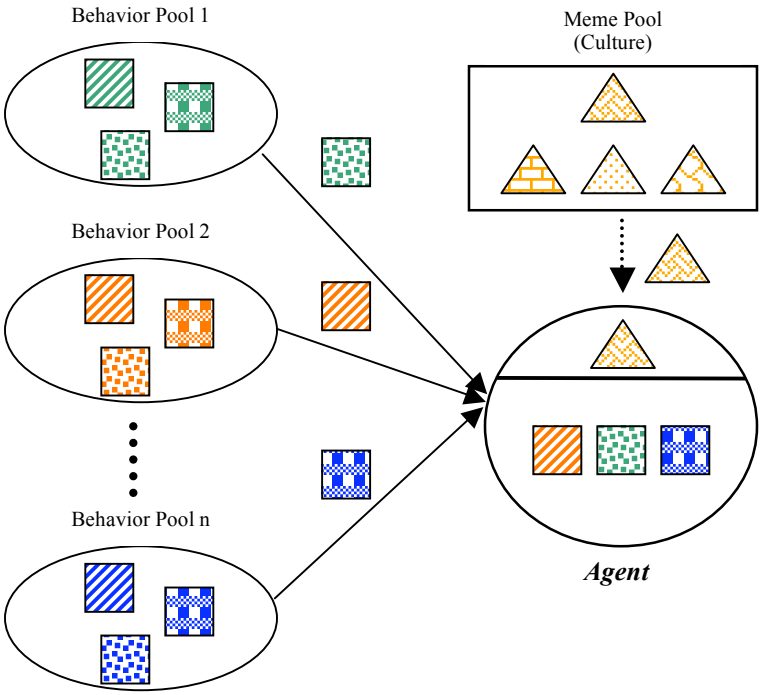

Figure 1. Building an agent from different behavior pools.

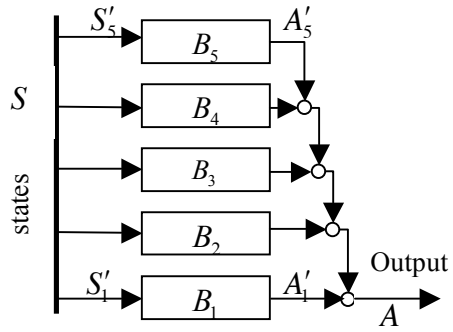

Figure 2. A typical structure of a Purely Parallel Subsumption Architecture.

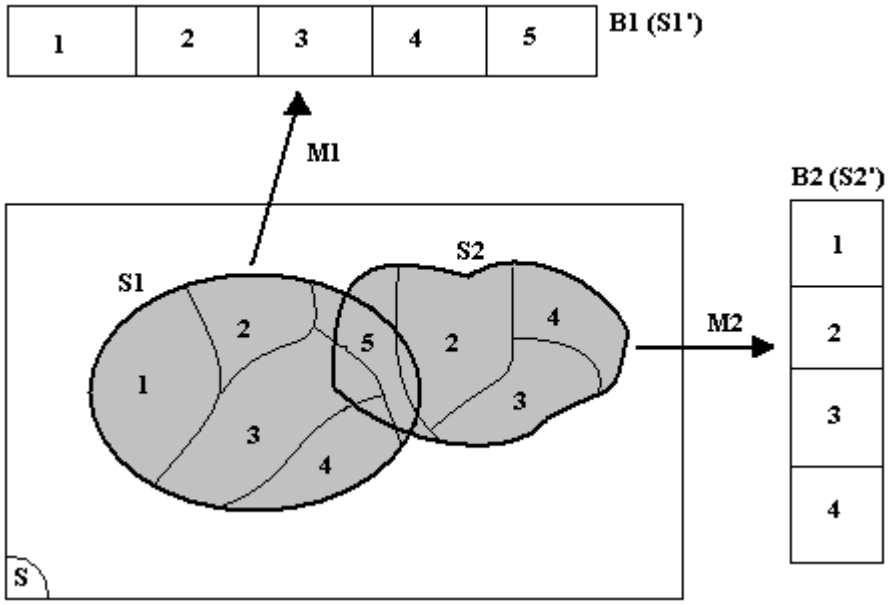

Figure 3. Excitation subspaces of B_1 and B_2 in S and corresponding mapping to S_1' and S_2'. Note that excitation spaces may overlap.

- Initialize n different behavior pools $\{\mathrm{B}_i\}$ for each behavior type B_i
- Initialize an empty culture (meme pool) $\mathbf{M}$
- While stopping condition are not met
    - Selects n different behavior modules B_i from each behavior pool to make a set of randomly chosen behavior modules $\{B_i\}$
    - If there is any meme in the meme pool $\mathbf{M}$, select a meme $T^0 \in \mathbf{M}$ according to the fitness of each meme
    - Pass $\{B_i\}$ and T^0 to the agent
        - Set initial structure as T^0
        - Initialize value (e.g. $\tilde{V}_{ij} = 0$) and learning parameters
        - For a lifetime do
            - Update learning parameters (e.g. decay $\alpha_{k,ij}$)
            - Select an architecture T^* that maximizes (9) (Zero-Order representation) (If the architecture is fixed, skip this step).
            - Using architecture T^*, let the agent interact with the environment for a while
            - Receive reinforcement signal from the critic (external or internal)
            - Update the estimation of necessary values ($\{\tilde{V}_{ij}\}$ for Zero Order structure learning representation (13))
        - Return fitness (14) and the final structure T^* to the evolutionary process
    - Share fitness to behaviors according to the sharing policy (uniform (15) or value-based (17))
    - Update meme pool using T^* and its fitness (22)
    - For each behavior pool
        - Apply conventional genetic operators to behavior modules in order to generate a new population, i.e. Selection, Crossover, and Mutation.

Figure 4. Proposed framework for development of behavior-based systems

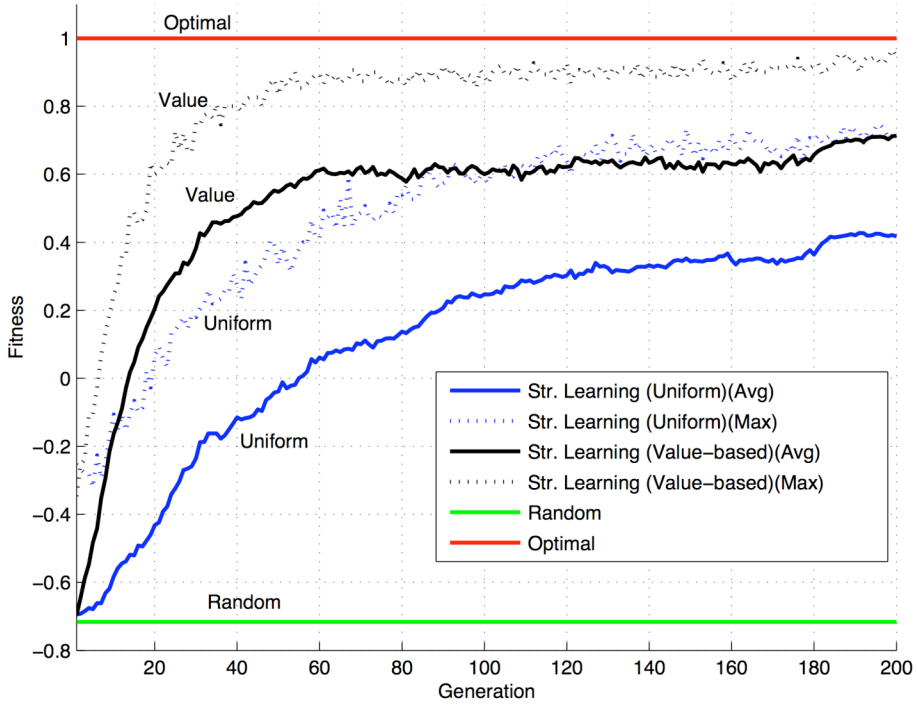

Figure 5. (Abstract Problem) Average and maximum fitness comparison for different fitness sharing methods with behavior co-evolution and structure learning: 1) uniform fitness sharing (blue) and 2) value-based fitness sharing (black). Solid lines indicate the average fitness of the population and dotted lines show maximum fitness. The bottommost line (green) shows the expected fitness of a random behavior and structure (with the same probability of selecting action or NA for each behavior) and the uppermost line (red) is the maximum achievable fitness.

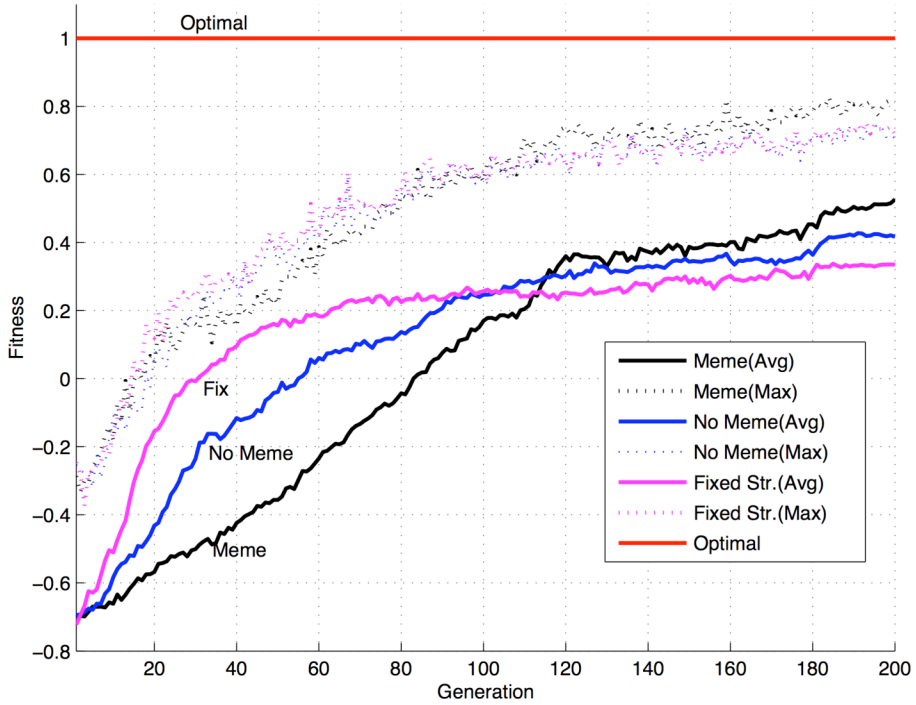

Figure 6. (Abstract Problem) Average and maximum fitness comparison for different design methodologies that use ***uniform*** fitness sharing: 1) co-evolution of behavior modules and learning structure (blue), 2) co-evolution of behavior modules and learning structure benefiting from the meme-induced initial knowledge (black), and 3) co-evolution of behavior modules and fixed structure (magenta). Solid lines indicate the average fitness of the population and dotted lines show the maximum fitness. Maximum achievable fitness is shown by the uppermost line (red).

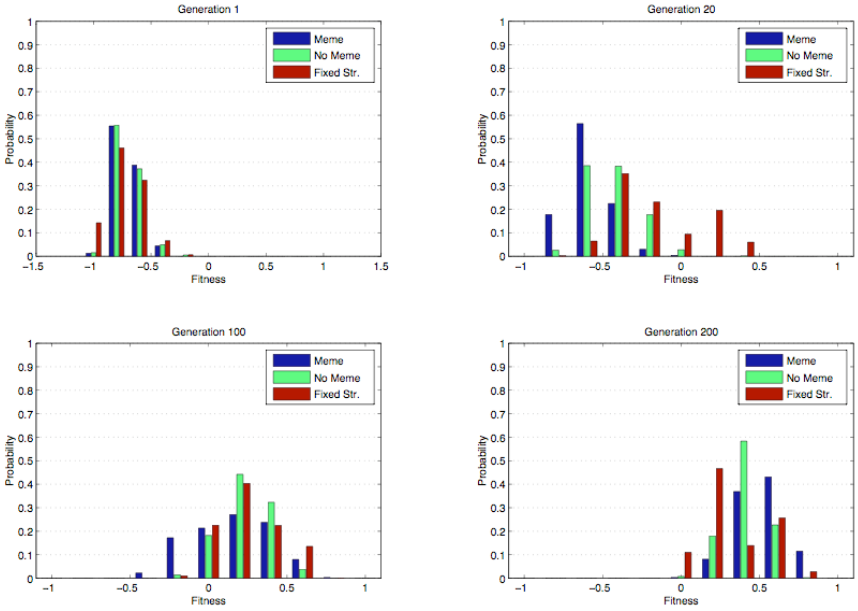

Figure 7. (Abstract Problem) Fitness probability density comparison for ***uniform*** fitness sharing. Comparison is made among agents using meme pool as their initial knowledge for their structure learning (left/dark blue), agents that learn structure from an initial random setting (middle/green), and agents with fixed structure (right/red).

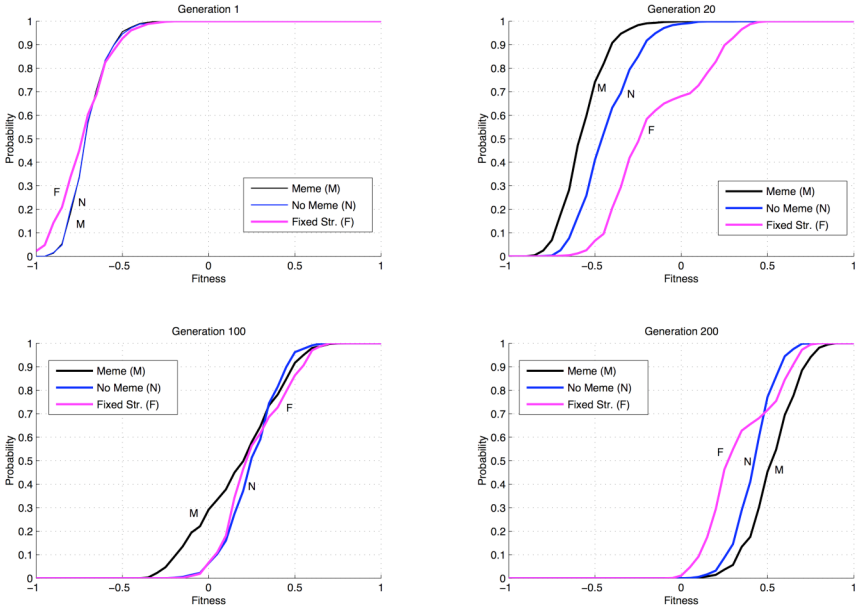

Figure 8. (Abstract Problem) Probability distribution comparison for ***uniform*** fitness sharing ($P\{Fitness \leq \varphi\}$). Comparison is made among agents using meme pool as their initial knowledge for their structure learning (black), agents that learn structure from a random initial setting (blue), and agents with hand-designed structure (magenta). Right-side tendency of distributions indicates higher chance of generating very good agents.

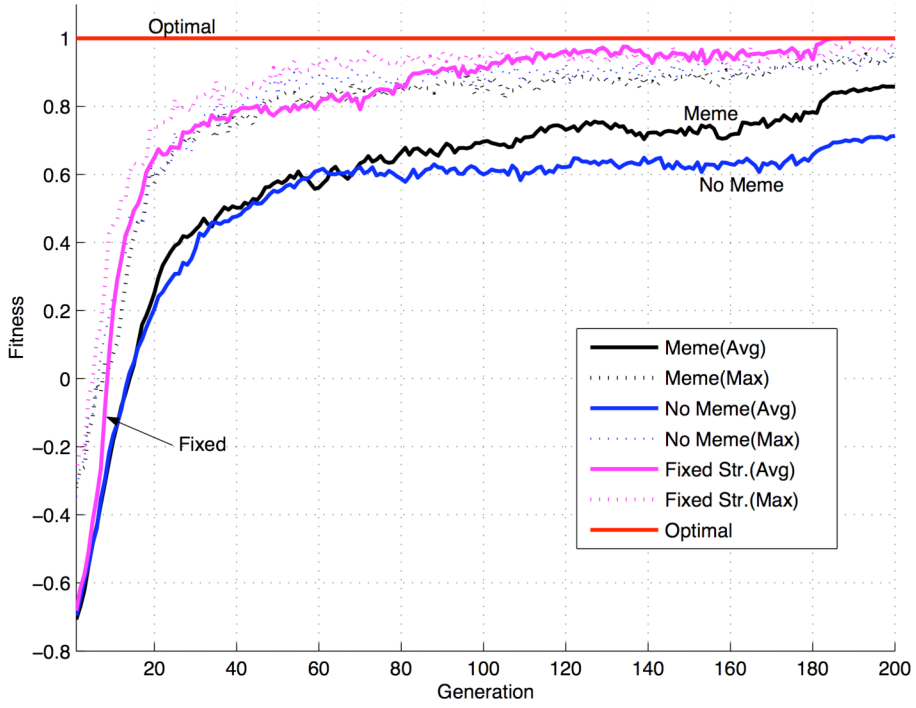

Figure 9. (Abstract Problem) Average and maximum fitness comparison for different design methodologies that use ***value-based*** fitness sharing: 1) co-evolution of behavior modules and learning structure (blue), 2) co-evolution of behavior modules and learning structure benefiting from the meme pool initial knowledge (black), and 3) co-evolution of behavior modules and fixed structure (magenta). Solid lines indicate the average fitness of the population and dotted lines show the maximum fitness. Maximum achievable fitness is shown by the uppermost red line.

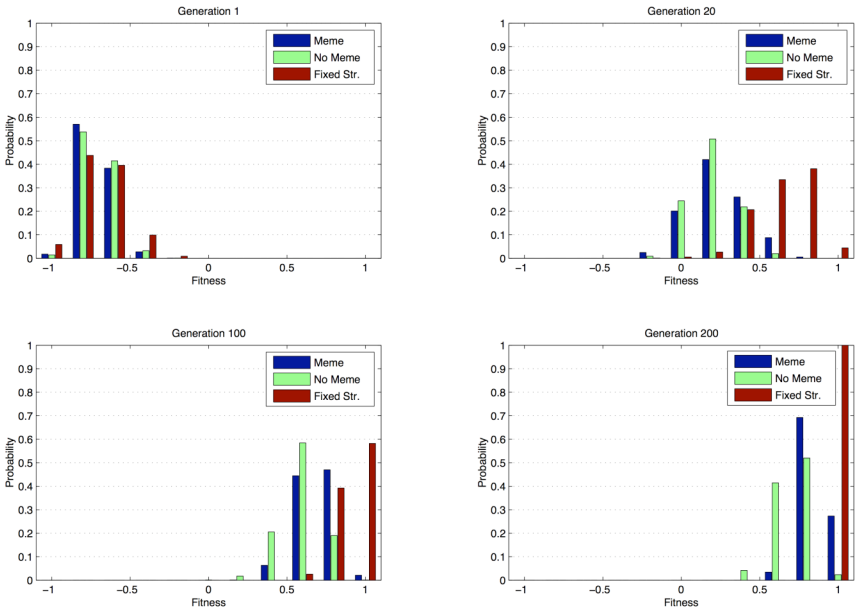

Figure 10. (Abstract Problem) Fitness probability density comparison for ***value-based*** fitness sharing. Comparison is made among agents using meme pool as their initial knowledge for their structure learning (left/dark blue), agents that learn structure from an initial random setting (middle/green), and agents with fixed structure (right/red).

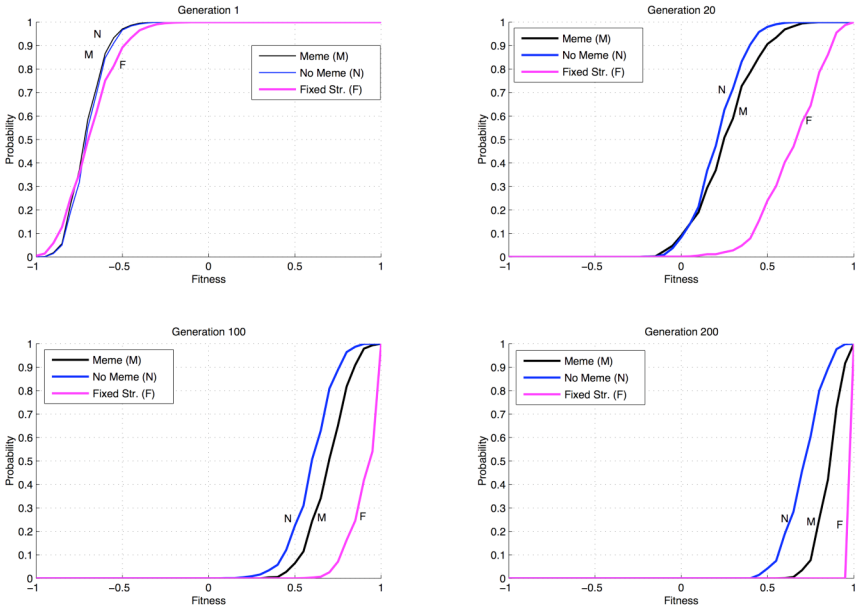

Figure 11. (Abstract Problem) Probability distribution comparison for value-based fitness sharing ($P\{Fitness \le \varphi\}$). Comparison is made among agents using meme pool as their initial knowledge for their structure learning (black), agents that learn structure from a random initial setting (blue), and agents with hand-designed structure (magenta). Right-side tendency of distributions indicates higher chance of generating very good agents.

Figure 12. A group of robots lifting a bulky object.

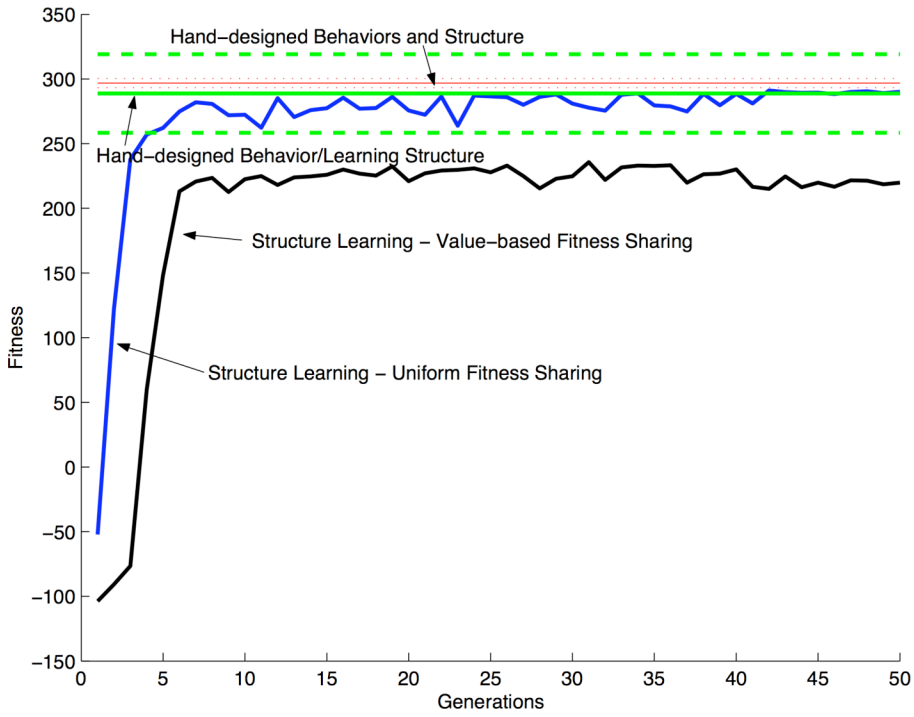

Figure 13. (Object Lifting) Averaged last five episodes fitness comparison for different design methods: 1) co-evolution of behavior modules (uniform fitness sharing) and learning structure (blue), 2) co-evolution of behavior modules (valued-based fitness sharing) and learning structure (black), 3) hand-designed behavior modules with learning structure (the second uppermost solid line/green), and 4) hand-designed behavior modules and structure (uppermost solid line/red). Dashed lines across the hand-designed cases (3 and 4) show one standard deviation region across the mean performance.

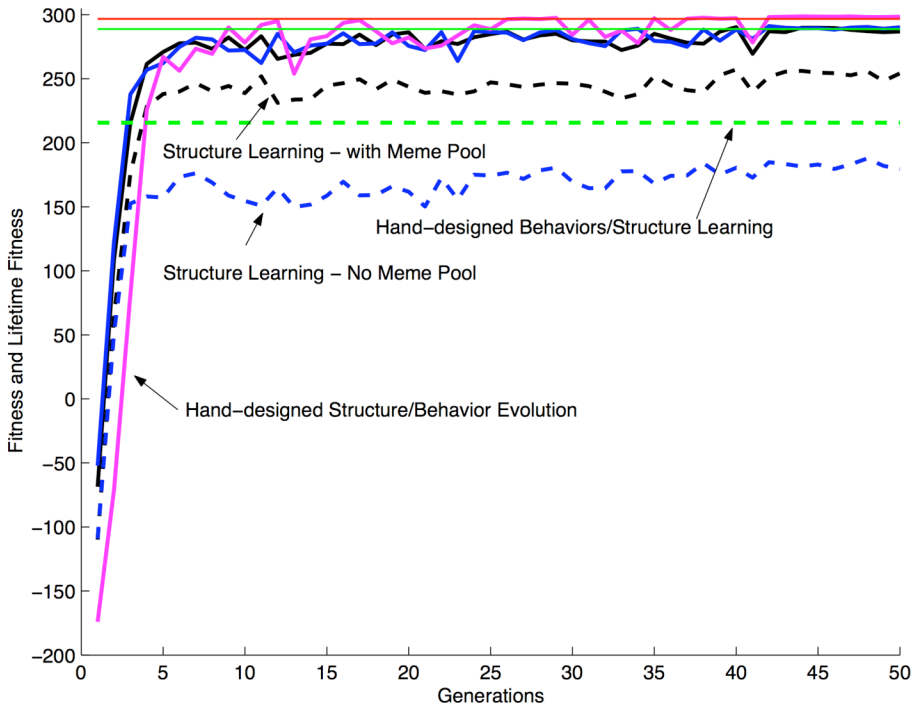

Figure 14. (Object Lifting) Averaged last five episodes and lifetime fitness comparison for ***uniform*** fitness sharing co-evolutionary mechanism: 1) co-evolution of behavior modules and learning structure (blue), 2) co-evolution of behavior modules and learning structure benefiting from meme pool initial knowledge (black), 3) co-evolution of behavior modules and hand-designed structure (magenta), 4) hand-designed behavior modules and learning structure (the second uppermost solid line/green), and 5) hand-designed behavior modules and structure (the uppermost solid line/red). Solid lines indicate the last five episodes of the agent's lifetime and dashed lines indicate the agent's lifetime fitness. Although the final time performance of all cases is close to each other, the lifetime fitness of memetic-based designs is much higher.

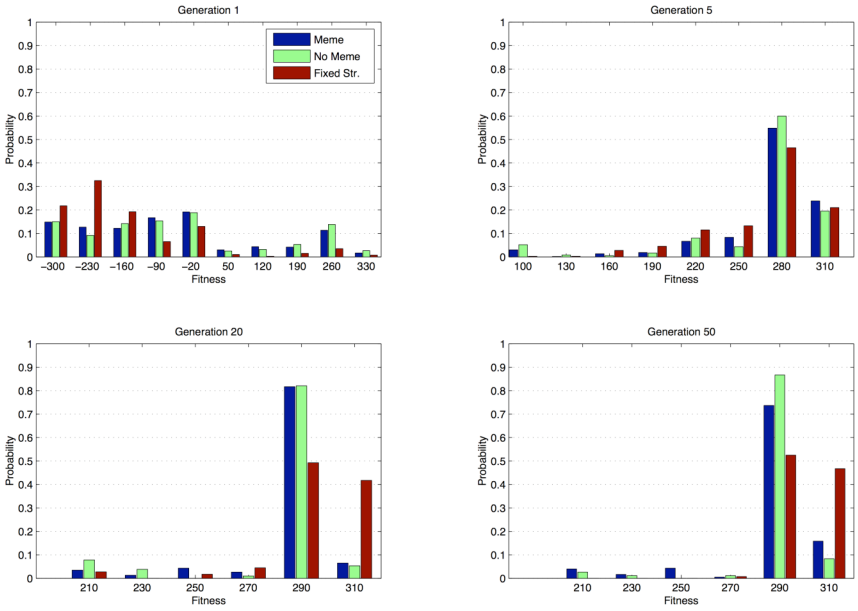

Figure 15. (Object Lifting) Fitness probability density comparison for ***uniform*** fitness sharing. Comparison is made among agents using meme pool as their initial knowledge for their structure learning (left/dark blue), agents that learn structure from a random setting (middle/green), and agents with hand-designed structure (right/red).

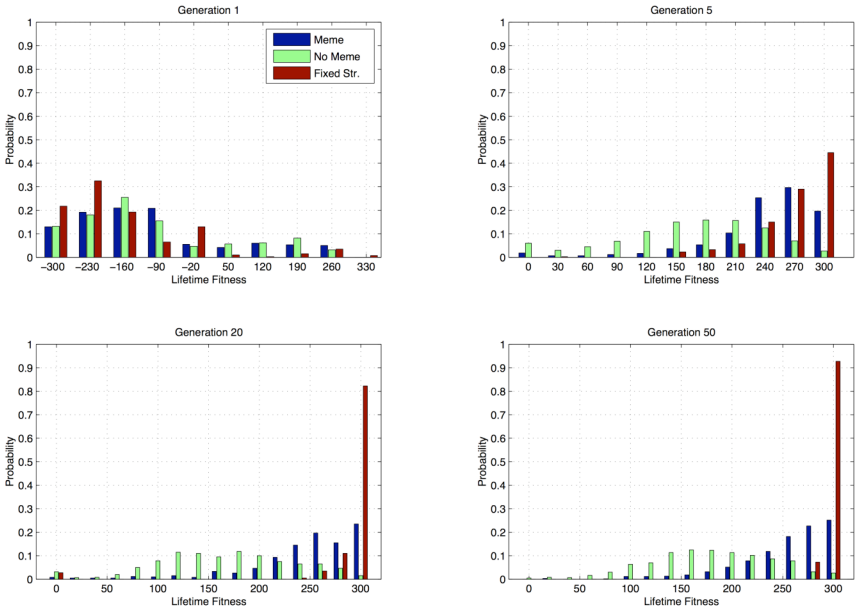

Figure 16. (Object Lifting) Lifetime fitness probability density comparison for ***uniform*** fitness sharing. Comparison is made among agents using meme pool as their initial knowledge for their structure learning (left/dark blue), agents that learn structure from a random setting (middle/green), and agents with hand-designed structure (right/red).

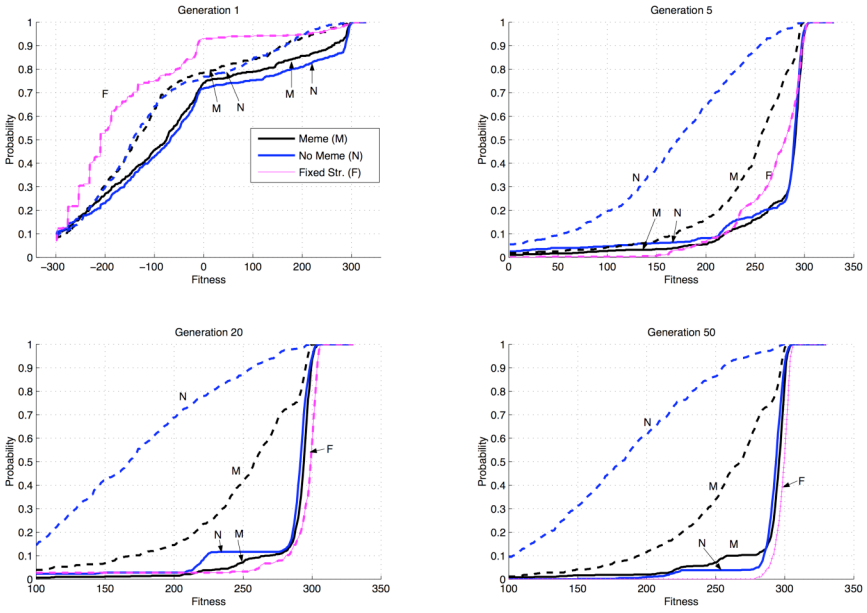

Figure 17. (Object Lifting) Probability distribution comparison for ***uniform*** fitness sharing ($P\{Fitness \le \varphi\}$). Comparison is made among agents using meme pool as their initial knowledge for their structure learning (black), agents that learn structure from a random initial setting (blue), and agents with hand-designed structure (magenta). Dashed lines are for distribution for *lifetime* fitness. Right-side tendency of distributions indicates higher chance of generating very good agents.

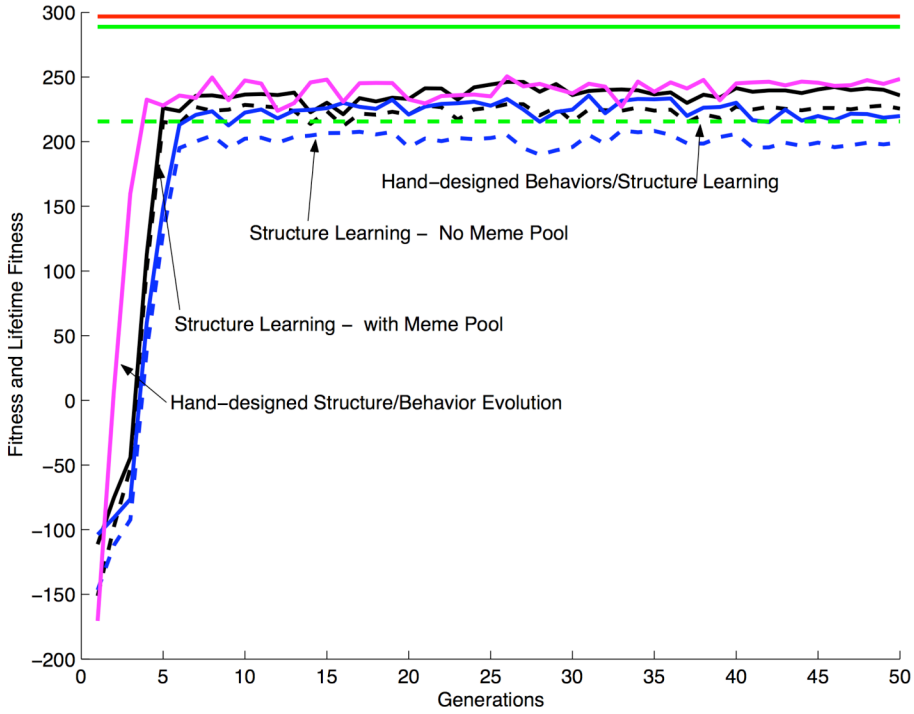

Figure 18. (Object Lifting) Averaged last five episodes and lifetime fitness comparison for ***value-based*** fitness sharing co-evolutionary mechanism: 1) co-evolution of behavior modules and learning structure (blue), 2) co-evolution of behavior modules and learning structure benefiting from meme-induced initial knowledge (black), 3) co-evolution of behavior modules and hand-designed structure (magenta), 4) hand-designed behavior modules and learning structure (the second uppermost solid line/green), and 5) hand-designed behavior modules and structure (the uppermost solid line/red). Solid lines indicate the last five episodes of the agent's lifetime and the dashed lines indicate the agent's lifetime fitness. The lifetime fitness of hand-designed behavior modules and learning structure is the flat dashed line. The lifetime fitness of structure learning without meme pool is the lower and that with meme is the upper dashed curve. Although the final time performance of all cases is rather the same, the lifetime fitness of memetic-based design is higher.

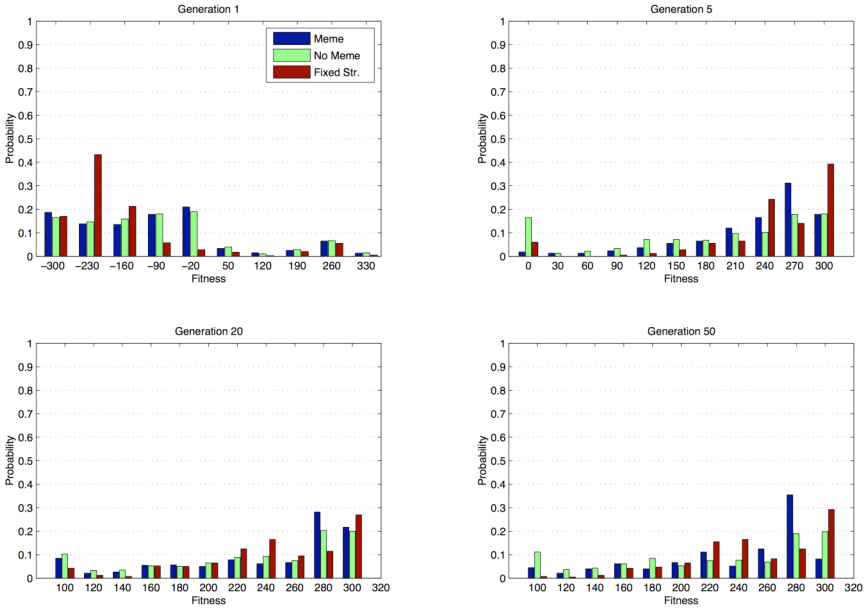

Figure 19. (Object Lifting) Fitness probability density comparison for ***value-based*** fitness sharing. Comparison is made between agents using meme pool as their initial knowledge for their structure learning (left/dark blue), agents that learn structure from a random initial setting (middle/green), and agents with hand-designed structure (right/red).

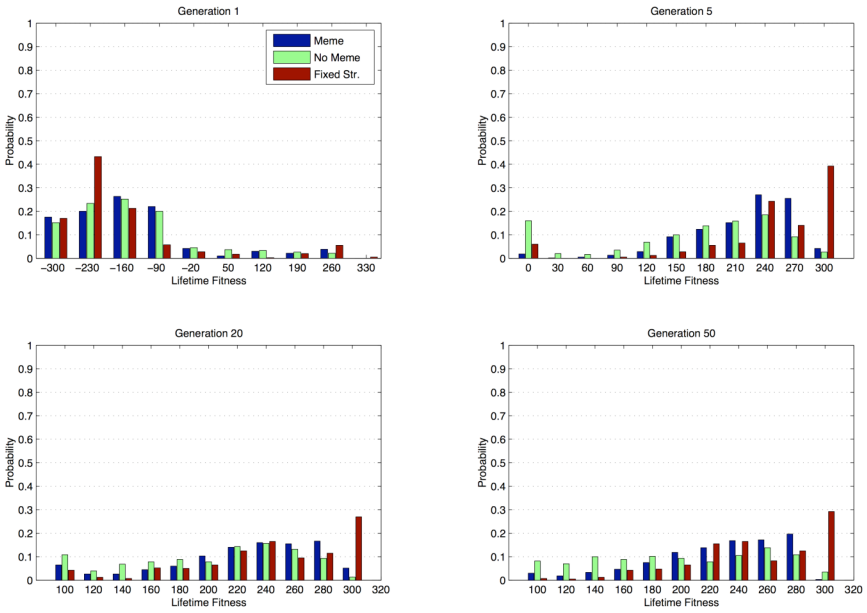

Figure 20. (Object Lifting) Lifetime fitness probability density comparison for ***value-based*** fitness sharing. Comparison is made among gents using meme pool as their initial knowledge for their structure learning (left/dark blue), agents that learn structure from a random initial setting (middle/green), and agents with hand-designed structure (right/red).

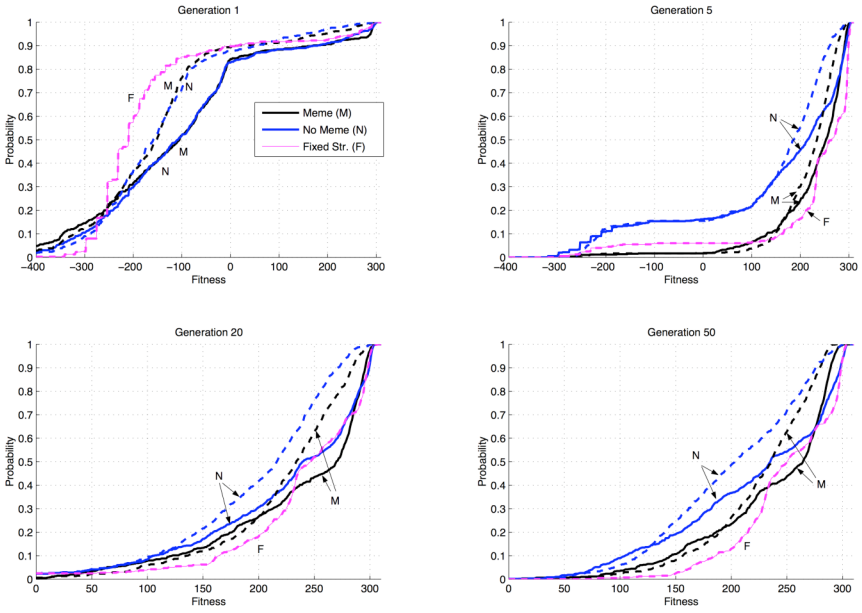

Figure 21. (Object Lifting) Probability distribution comparison for ***value-based*** fitness sharing ($P\{Fitness \le \varphi\}$). Comparison is made among agents using meme pool as their initial knowledge for their structure learning (black), agents that learn structure from a random initial setting (blue), and agents with hand-designed structure (magenta). Dotted lines are for distribution for *lifetime* fitness. Right-side tendency of distributions indicates higher chance of generating very good agents.

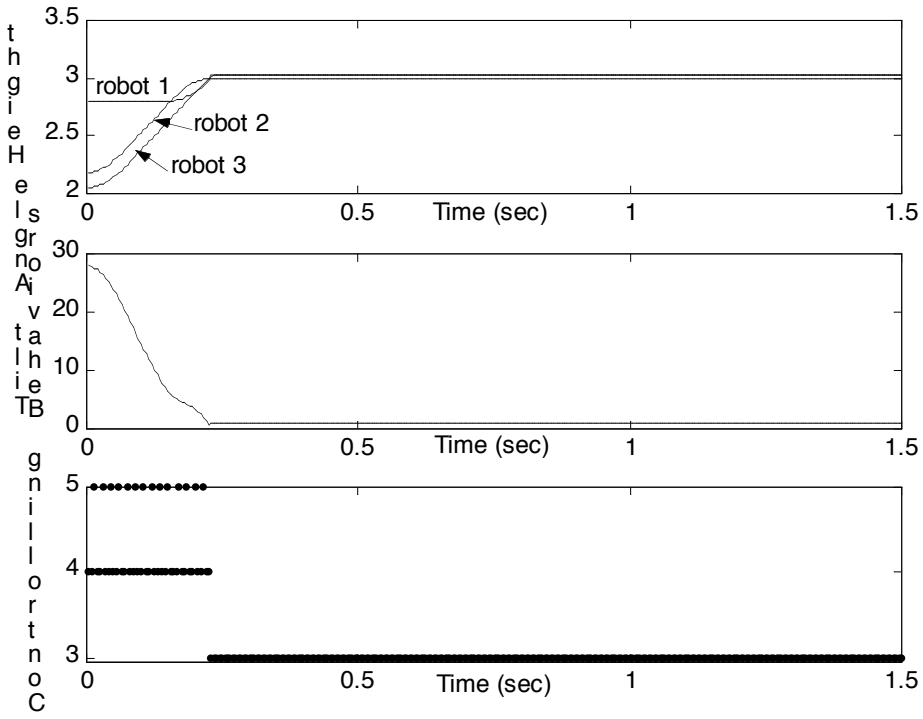

Figure 22. A sample trajectory showing the position of robot-object contact points, the tilt angle of the object during object lifting, and controlling behavior module of robots at each time step after 50 generations of behavior modules cooperative co-evolution (with uniform fitness sharing) and 20 episodes of structure learning in each trial (meme is used). Behavior modules correspondence with numbers in the lowest diagram is as follows: 3 (Stop), 4 (Hurry up), 5 (Slow down). Other behavior modules are not used in this sample co-evolved/learned architecture.

Table 1. Problem Specification

	Abstract Problem	**Object Lifting**
Problem Space	5x5 problem space and 7 actions	Multi-robot object lifting
Solution Space	Seven behavior modules each can produce one type of action (behavior module B_i has $A_i = \{a_i, NA\}$)	Five behaviors modules' state and action space (*Push more*, *Don't go fast*, *Stop at goal*, *Hurry up*, and *Slow down*.)
Co-evolution Parameters	Seven Populations, Generations = 200, Population size = 30, No. of individual evaluations in each gen. = 300, Tournament selection (competition between 3 individuals), the best individual of each genetic pool goes directly to the next generation, $p_c = 0.5$ $p_m^{hard} = \begin{cases} \frac{0.01}{\log_2(gen+1)} & \text{gen} \le 179 \\ 0 & 180 \le \text{gen} \le 200 \end{cases}$ $p_m^{soft} = 2p_m^{hard}$	Five Populations, Generations = 50, Population size = 20, No. of individual evaluations in each gene.= 200, Tournament selection (competition between 3 individuals), the best individual of each genetic pool goes directly to the next generation, $p_c = 0.5$ $p_m^{hard} = \begin{cases} \frac{0.01}{\log_2(gen+1)} & \text{gen} \le 40 \\ 0 & 41 \le \text{gen} \le 50 \end{cases}$ $p_m^{soft} = 2p_m^{hard}$
Learning Parameters	Structure learning with ZO representation $\alpha_{episode} = 1/episodes$	Structure learning with ZO representation $\alpha_{episode}^{str} = 0.1(0.99)^{episode}$
Memetic Parameters	Culture size = 5 $\alpha_{T_i} = 0.3$ (22) Meme selection: scaled roulette wheel (p=0.9)/random (p=0.1)	Culture size = 5 $\alpha_{T_i} = 0.3$ (22) Meme selection: scaled roulette wheel (p=0.9)/random (p=0.1)
Experimentation Conditions	100 episodes trial 3 runs Fitness is the mean value of reinforcement signal during the agent's lifetime	300 time step lift-up, $\Delta T = 0.005$ 25 (5) episodes trial for with structure learning (fixed structure) case 3 runs for learning cases – 2 runs for fix structure case - 1 run for pre-defined behaviors (no evolution) $z^{1,2,3}(1) \in U(2,3)$ Fitness (Lift time fitness) is calculated by averaging the last 5 episodes (all episodes) - Fitness (and not the Lifetime fitness) is used for evolutionary mechanism
Performance Measures	Received reinforcement signal (MA filtered with window size 10)	1-Fitness (average of last 5 episodes) 2-Liftime fitness (average of all episodes)

Table 2. Hand-Designed Behaviors.

Push more: $v(k+1) = v(k) + \Delta v$

Do not go fast: if $v(k) > v_{\max}$ then $v(k+1) = v_{\max}$ else do nothing

Stop at goal: if $z(k) \geq z_{goal}$ then stop ($v(k+1) = 0$)

Hurry up: if $\tau(k) > \tau_0$ and the robot is the lowest one then

$$v(k+1) = \min(v(k) + \Delta v, v_{\max})$$

Slow down: if $\tau(k) > \tau_0$ and the robot is the highest one then

$$v(k+1) = \max(v(k) - \Delta v, 0)$$

$$T^{hand-designed} = [\text{Stop} \quad \text{SlowDown} \quad \text{HurryUp} \quad \text{DontGoFast} \quad \text{PushMore}]$$

Table 3. Reinforcement Signal Definition

$r_k = r_k^{\tau_1} + r_k^{\tau_2} + r_k^{z_1} + r_k^{z_2} + r_k^{v}$ (29)	$r_k^{\tau_1} = \begin{cases} 1 & \tau(k) - t(k-1) < -0.5 \\ -0.1 & \text{otherwise} \end{cases}$ (30)
$r_k^{\tau_2} = \begin{cases} \dfrac{1}{\sqrt{k}} & \tau(k) < \tau_0 \\ -0.1\sqrt{k} & \text{otherwise} \end{cases}$ (31)	$r_k^{z_1} = \begin{cases} 1 & \lvert z(k) - z_{goal} \rvert < 0.5 \\ -0.1 & \text{otherwise} \end{cases}$ (32)
$r_k^{z_2} = -1 \quad z(k) > z_{goal} + 0.2$ (33)	$r_k^{v} = -0.1 \quad v(k) > v_{max}$ (34)

Table 4. State and Action Definitions for to be Evolved Behavior Modules

Push more	$S'_{\text{Push more}} = \{\varnothing\}$	$A'_{\text{Push more}} = \{v(k+1) = v(k) + \Delta v, NA\}$
Do not go fast	$S'_{\text{Don't go fast}} = \{v(k) \mid v(k) \in \{\leq 0,1,2,3,4,\geq 5\}\}$	$A'_{\text{Don't go fast}} = \begin{Bmatrix} v(k+1) = \min(v(k), v_{\max}), \\ NA \end{Bmatrix}$
Stop at goal	$S'_{\text{Stop}} = \{"z(k) - z_{goal} < 0", "z(k) - z_{goal} \geq 0"\}$	$A'_{\text{Stop}} = \{v(k+1) = 0, NA\}$
Hurry up	$S'_{\text{Hurry}} = \begin{Bmatrix} \text{lowest robot, middle robot,} \\ \text{highest robot} \end{Bmatrix} \times \{"\tau < \tau_0", "\tau > \tau_0"\}$	$A'_{\text{Hurry}} = \{\min(v(k) + \Delta v, v_{\max}), NA\}$
Slow down	$S'_{\text{Slow}} = \begin{Bmatrix} \text{lowest robot, middle robot,} \\ \text{highest robot} \end{Bmatrix} \times \{"\tau < \tau_0", "\tau > \tau_0"\}$	$A'_{\text{Slow}} = \{\max(v(k) - \Delta v, 0), NA\}$