# Hybrid Behavior Co-evolution and Structure Learning in Behavior-based Systems

Amir massoud Farahmand, Majid Nili Ahmadabadi, Caro Lucas, and Babak N. Araabi

*Abstract*— **Designing an intelligent situated agent is a difficult task as the designer must see the problem from the agent's standpoint considering all its sensors and actuators. We have devised a co-evolutionary/reinforcement learning hybrid method to automate the design of hierarchical behavior-based systems. In our approach, the design problem is decomposed into two separate parts: developing a repertoire of behaviors and organizing those behaviors in a structure. Mathematical formulation shows how to decompose the value of the structure to simpler components. These components can be estimated and used to find the optimal organization of behaviors during the agent's lifetime. Moreover, a novel co-evolutionary mechanism is suggested that evolves each type of behavior separately in their own genetic pool. Our method is applied to the decentralized multi-robot object lifting task which results in human-competitive performance.**

## I. Introduction

OUR long-term research goal is developing general automatic methods for designing distributed multi-agent systems. In this paper, we present a hybrid co-evolutionary/reinforcement learning method for designing behavior-based systems. Our method uses co-evolutionary mechanism for behavior design and reinforcement learning for organizing those behaviors in a hierarchy. The motivation behind our work is benefiting the global searching capability of the evolutionary mechanism and fast adaptation of learning. Moreover, the special decomposition of the problem into behavior evolution and hierarchy learning increases the flexibility and the modularity of the design process.

Behavior-based approach is chosen as the physical agent's low-level controller ([1] and [2]). Behavior-based systems are not only robust to different uncertainties that might be faced by a robot in the real world, but also are biologically plausible. There are many different successful applications of behavior-based systems (e.g. [3], [4], [5], and [6]) most of them are hand-designed. There is little work on automating the design process. In practice, the designer generally uses bottom-up trial and error approach to devise a behavior-based system that meets the required performance objectives. In most cases, this is a difficult task and the design burden is considered one of the major drawbacks of behavior-based systems.

Some researches have tried to solve this problem by using learning or evolution. In one research track, learning is used to partially solve a subset of existing problems in behavior-based system design (e.g. [7], [8], and [9]). Most learning algorithms, which are based on local search, may not find a good solution in a very big and bumpy parameter space and become trapped in a local optimum. Despite this disadvantage, they are relatively fast in finding some solutions even in non-stationary environments. The other prominent approach to automate the design process is based on artificial evolution (e.g. [10], [11], and [12]). Although evolutionary robotics seems a promising alternative for automatic robotic system design, it is usually too slow and cannot handle non-stationary environment easily. Moreover, the current approach to evolutionary robotics produces non-modular, non-reusable controllers (See [11] as an alternative approach that considers the modularity of the controller).

We propose a hierarchical behavior-based system design methodology in which not only the agent learns to arrange behaviors in the architecture (like [13]), but also develops behaviors too. We decompose the problem into two separate sub-problems: behavior development and arranging behaviors in the hierarchical structure. A co-evolutionary mechanism develops new behaviors in separate *genetic (behavior) pools*. Each genetic pool is devoted to a single kind of behaviors (e.g. a pool for "obstacle avoidance behavior" and a separate pool for "go forward" behavior in a simple mobile robot navigation task). To develop a new agent, we collect a behavior instance from each pool to form a set of behaviors. Thereafter, the agent tries to organize those *inherited* behaviors in its architecture guided by the received reinforcement signal. We use a special form of reinforcement learning paradigm for hierarchy (structure) learning (See [14] for more information about reinforcement learning; see also [15] for a survey of hierarchical reinforcement learning methods which are similar to our structure learning). Based on the performance of the agent, the fitnesses of the corresponding behaviors of each

A. M. Farahmand was with the Department of Electrical and Computer Engineering, University of Tehran, Iran. He is now with the Department of Computing Science, University of Alberta, Canada (e-mail: amir@cs.ualberta.ca).

M. Nili Ahmadabadi is with the Department of Electrical and Computer Engineering, University of Tehran, Iran (e-mail: mnili@ut.ac.ir).

C. Lucas is with the Department of Electrical and Computer Engineering, University of Tehran, Iran (e-mail: lucas@ipm.ir).

B. N. Araabi is with the Department of Electrical and Computer Engineering, University of Tehran, Iran (e-mail: araabi@ut.ac.ir).

behavior pool are updated.

The hybridization of an evolutionary method and a local search (seeing learning as a local search) is not new in the evolutionary computation community. The resulted class of algorithms, which has many variations, is often called memetic algorithm [16]. Nevertheless, our hybridization is different than most other approaches. In our method, learning and evolution are applied to two different spaces: hierarchy space and behavior space. Learning optimizes the objective function by arranging a limited set of behaviors in the structure and evolution optimizes the same objective function by changing behaviors' internal mechanism. Learning acts as the fast adaptation mechanism for the agent and evolution acts as the slow but global optimizer.

Finally, we should mention that we have chosen a specialized version of the Subsumption Architecture (SSA) ([1]) as our hierarchical behavior-based system. The SSA is a successful and competent behavior-based architecture in which different behaviors compete with each other to take control of the agent. The reason for selecting SSA was two-fold: its flexibility and the existence of structure learning method for it [13].

After formulating the problem mathematically in Section I, we propose our structure (hierarchy) learning method in Section II. In this section, we show how to organize a given set of behaviors in the architecture. This organization is based on the received reinforcement signal. In Section III, we introduce our special co-evolutionary mechanism and show how to separate behaviors into different genetic pools, and how to assign fitness to each behavior, and the way to combine our behaviors. Subsequently, in Section IV, we show the application of our method in designing a decentralized multi-robot object lifting task that has been previously solved by a human and considered as a difficult problem.

## I. Problem Formulation

Suppose that we have a set of $n$ behaviors $\{B_i\}; i = 1,...,n$, defined as the following map between perception and action of the agent:

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

where $S_i$ is the subset of the state space observable by behavior $B_i$ ($S_i \cap S_j \neq \varnothing$, in general), $A_i$ is the set of $B_i$'s output actions, i.e. actuators, and $M_i$ is the mapping that projects the agent's total state to its internal perception. Each behavior $B_i$ is augmented with "No Action" (NA). NA is an action that even if selected, it does not change actuators. NA is for allowing behaviors not to activate in some regions of their excitation space. It can be shown that this enables the agent to achieve higher performance. A behavior $B_i$ is **excited** if $s_i \in S_i$ ($s_i' \in S_i'$) and it will become **activated** and output $a_i$ if $a_i = B_i(M_i(s_i)) = B_i(s_i') \neq NA$. As an architecture, we use a special but important case of Purely Parallel Subsumption Architecture (PPSSA) (Fig. 1). In PPSSA, all behaviors are parallel and a higher behavior has the priority to suppress the lower one and to become the **controlling** behavior of the agent. Suppose that we have a set of $n$ behaviors $\{B_i\}$ and a $m$-behavior architecture $T$ consisted of $m$ of those behaviors ($m \le n$). $T(i)$ denotes the behavior in the $i^{th}$ layer of $T$ in that the numbering starts from the lowest layer. We can write

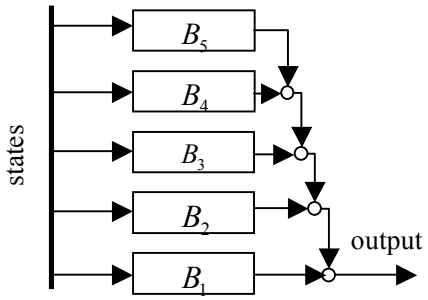


**Fig. 1.** A typical purely parallel Subsumption architecture.

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

Now, we define our structure learning problem. Regarding reinforcement learning notion, the goal of learning is maximizing (or minimizing) a function of received reinforcement signal. Having $r_t$ as the reinforcement signal received at time $t$ (which may be dependent on the system's state, selected action, etc.), defining $R$ as a random variable that indicates the episode's return for the agent, and $N$ as the length of episode, the value of the total system with structure $T$ and set of behaviors $\{B_i\}$ $(i = 1,...,n)$ following policy $\pi$ is

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

In the structure learning, we assume having a suitable behaviors repertoire and our goal is choosing an ordered sequence $T$ with $m$ out of $n$ behaviors that maximizes (3).

$$T^* = \arg\max_T V_T \quad (4)$$

On the other hand, the behavior evolution problem considers adjusting each behavior $B_i$'s mapping from the

sensing space ( $S_i'$) to the augmented action space ( $A_i'$) using evolutionary process. In other words, a behavior must choose an appropriate action $a_i^*$ in each $s_i'$ that maximizes its fitness. More detail is presented in Section III.

## II. STRUCTURE LEARNING

In this section, we propose a method for structure learning. Our aim is finding a $T^*$ that satisfies (4). To solve this problem, we should find a solution to the following sub-problems: 1) Representation: How should the agent represent knowledge gathered during learning? 2) Hierarchical Credit Assignment: How should the agent assign credit to different behaviors and layers in its architecture? 3) Knowledge Updating: How should the agent update its knowledge when it receives reinforcement signal? By defining an appropriate representation, hierarchical credit assignment and knowledge updating can be solved easily as it is shown in this section by introducing Zero Order representation [13].

### A. Zero Order Representation

In this representation, *we store the expected value of each behavior in each layer. In other words, the merit of being in a layer for each behavior is stored*. For a given structure $T$, we have:

$$
\begin{aligned}
V_T &= \underset{\pi}{E}[R] = \underset{\pi}{E}\left[\frac{1}{N}\sum_{t=1}^{N} r_t\right] \\
&= \underset{\pi}{E}\left[\frac{1}{N}\sum_{t=1}^{N}\left\{r_t \wedge \begin{pmatrix} \text{"}L_1 \text{ is controlling"} \vee \text{"}L_2 \text{ is controlling"} \vee \\ \ldots \vee \text{"}L_m \text{ is controlling"} \end{pmatrix}\right\}\right] \\
&= \underset{\pi}{E}\left[\frac{1}{N}\sum_{t=1}^{N}\{r_t \wedge \text{"}L_1 \text{ is controlling"}\}\right] + \ldots + \underset{\pi}{E}\left[\sum_{t=1}^{N}\{r_t \wedge \text{"}L_m \text{ is controlling"}\}\right] \\
&= \underset{\pi}{E}\left[\frac{1}{N}\sum r_t \mid L_1 \text{ is controlling}\right]\cdot P(L_1 \text{ is controlling}) \\
&\quad + \underset{\pi}{E}\left[\frac{1}{N}\sum r_t \mid L_2 \text{ is controlling}\right]\cdot P(L_2 \text{ is controlling}) \\
&\quad + \ldots + \underset{\pi}{E}\left[\frac{1}{N}\sum r_t \mid L_m \text{ is controlling}\right]\cdot P(L_m \text{ is controlling})
\end{aligned}
\tag{5}
$$

in which $L_i$ is the $i^{th}$ layer, $\underset{\pi}{E}[1/N \sum r_t | L_i \text{ is controlling}]$ is the expected reward of the system when the $i^{th}$ layer takes control and $P(L_i \text{ is controlling})$ is its probability of being the controlling layer assuming that at least one of behaviors becomes active. Note that this decomposition is possible because different time instances $\{t | \text{"}L_i \text{ is controlling" in timestep } t\}$ are *mutually exclusive* – only a behavior can be controlling at each moment. Here, we assume that at least of the behaviors is *active* (and not controlling which is just one) at every moment. Note that we do not assume independence of these events.

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

$$
V_{ZO}(i,j) = V_{ij} = \underset{\pi}{E}\left[\frac{1}{N}\sum r_t \,\middle|\, \begin{matrix} B_j \text{ is the controlling} \\ \text{behavior in the } i^{th}\text{layer} \end{matrix}\right] \tag{6}
$$

we have

$$
\begin{aligned}
&\underset{\pi}{E}[\frac{1}{N}\sum r_t \mid L_i \text{ is controlling}] \\
&= \sum_{j=1}^{n} P\{B_j \mid L_i\}\,\underset{\pi}{E}\left[\frac{1}{N}\sum r_t \,|\, B_j \text{ is the controlling behavior in } L_i\right] \\
&= \sum_{j=1}^{n} P\{B_j \mid L_i\} V_{ij} \quad i = 1,\ldots,m
\end{aligned}
\tag{7}
$$

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

$$
V_T = \sum_{i=1}^{m}\sum_{j=1}^{n} P\{B_j \mid L_i\} V_{ij} P(L_i \text{ is controlling}) \tag{8}
$$

It is evident that this representation is complete as it can represent every possible combination of behaviors. The representation space size is much smaller than a complete one that stores every possible combination and is

$$
\text{cardinality(ZO)} = n \cdot m \tag{9}
$$

In order to find the optimal structure, we should select a one that satisfies (4). To do so, it is certain that we must have an estimation of $V_{ij}$, $P(L_i \text{ is controlling})$, and $P\{B_j \mid L_i\}$.

According to the definition of $V_{ij}$ (6), credit assignment is straightforward: *$V_{ij}$ must be updated whenever $B_j$ is the controlling behavior 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 reinforcement signal $r_k$, $V_{ij}$ can be updated in the following stochastic approximation way:

$$
\begin{aligned}
&V_{ij_{k+1}} = (1-\alpha_{k,ij})V_{ij_k} + \alpha_{k,ij} r_k \\
&\begin{pmatrix} \text{whenever } B_j \text{ is the controlling} \\ \text{behavior in the } i^{th} \text{ layer} \end{pmatrix}
\end{aligned}
\tag{10}
$$

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 all of these variables and increase them accordingly. However, there is another elegant method to estimate all these values at once. Instead of updating components of (8) separately, it is possible to

**TABLE 1.** GENERAL LEARNING/CO-EVOLUTION MECHANISM PSEUDOCODE

- Initialize $n$ different behavior pools for each behavior type $\mathbf{B_i}$
- While ~(stopping condition) {
- Selects $n$ different behavior $B_i$ from each behavior pool to make a set of randomly chosen behaviors $\{B_i\}$
- Pass $\{B_i\}$ to the agent{
  - Initialize Learning parameters
  - For a [limited] lifetime do
    - Update learning parameters to ensure convergence
    - Select an architecture $T^*$ that maximizes (8) (Zero Order representation) using an optimization method (If the architecture is fixed, skip this step).
    - Using the provided architecture $T^*$ and behaviors $\{B_i\}$, let the agent interact with the environment for a while
    - Receive reinforcement signal from the critic (that can be external or internal)
    - Update the estimation of necessary values ($\{\tilde{V}_{ij}\}$ (13))
  - Return Fitness (13) to the evolutionary process }
- Share fitness to behaviors according to (14)
- For each behavior pool
  - Apply conventional genetic operators to behaviors in order to generate a new population, i.e. Selection, Crossover, and Mutation.

estimate all of them together by defining $\tilde{V}_{ij}$ as

$$\tilde{V}_{ij} = P\{B_j \mid L_i\} V_{ij} P(L_i \text{ is controlling}) \tag{11}$$

Therefore, we have

$$\tilde{V}_{ij_{n+1}} = (1-\alpha_{n,ij})\tilde{V}_{n,ij} + \alpha_{n,ij}\begin{bmatrix} "B_j \text{ is active at time step } n" \times \\ "L_i \text{ is controlling at time step } n" \times \\ r_n \end{bmatrix} \tag{12}$$

Hence, it is now possible to find the arrangement of behaviors in (8) that satisfies (4). Different methods can solve the combinatorial optimization problem of finding $T^*$ - we have used simple stochastic search in our implementations: calculating values for many different randomly chosen structures and selecting the one with the highest value. This approximate method is suitable for not so big solution spaces.

## III. BEHAVIOR CO-EVOLUTION

In this section, we discuss different aspects of our co-evolutionary behavior development scheme.

### *A. Cooperative Behaviors' Pools*

In contrast with most evolutionary approaches in evolutionary robotics, we do not use a single population of behaviors. We discriminate behaviors by their supposed "behavior" and make them evolve in their own behavior (genetic) pool. Therefore, if n different behaviors are needed to solve the problem, there will be n separate behavior pools without any direct interaction.

In order to evaluate each individual of each pool, we have proposed a special mechanism, which is *inspired* from the Enforced Sub-Population (ESP) [17]: First, a behavior is selected randomly from each behavior pool to form a set of behaviors. Thereafter, they are submitted to a trial in that the agent learns the best arrangement of behaviors using the introduced structure learning method. It is evident that arranged architecture may not work properly unless most of its behaviors are correct. By "correct", we mean that each of them performs such that by arranging it in the architecture, the agent can do something useful. The performance of the agent is correlated with the performance (i.e. fitness) of each behavior in it. However, the exact contribution of each behavior is not known, as the performance of the agent is a complicated function of all contributing behaviors and their arranged structure. Nevertheless, the performance of the agent can be a good measure of the performance of each behavior. Therefore, we estimate the fitness of each behavior as the average the fitness of all agents in which that specific behavior has contribution. If a single instance of a type of behavior involves in the architecture of a few different agents, its fitness is defined as the average fitness of all those agents. More precisely, if we define an average fitness of a set of behaviors in the agent as

$$V_{\{B\}}\Big|_{\text{Last K episodes}} = f_{\{B\}}\Big|_{\text{Last K episodes}} = E\left[\frac{1}{K}\sum_{t\in\text{Last K episode}} r_t \,\middle|\, \begin{matrix}\text{the agent with } \{B\}_i \\ t \in \text{Last K episode}\end{matrix}\right] \tag{13}$$

Then, each behavior's fitness is defined as

$$f(B_i^j) = \frac{1}{N}\sum_{\{B\}_i} V_{\{B\}_i}\Big|_{\text{Last } k \text{ episodes}} \tag{14}$$

in which $B_i^j$ is the $j^{th}$ individual from the $i^{th}$ population and $\{B\}_i$s are $N$ randomly chosen set of behaviors in

which $B_i^j \in \{B\}_i$. $\{B\}_i$ are chosen randomly, so it is possible for a specific behavior to be selected more often than others. However, by performing enough trials, we become sure that each behavior has involved in the sufficient number of experiments and the estimated fitness is a rather good measure for its goodness. Note that like ESP, we share fitness of the agent to its components according to the overall fitness of the agent (in our case, the components are behaviors, and in the ESP, they are neurons).

A real-valued Q-table-like representation with an appropriate dimension is used for each behavior $B_i$ as (note that this table has no value interpretation as Q-table in reinforcement learning.)

$$B_i(s') : S_i' \rightarrow A_i' \tag{15}$$

Action selection is greedy action selection, i.e.

$$B_i^j(s_i) = \arg\max_{a_i} Q_i^j(s_i, a_i) \tag{16}$$

The result of this action selection may be an action for the agent's actuator or NA. In the latter case, that behavior does not become active and lets lower behaviors take control of the agent.

### B. Genetic Operators

We use one crossover and two mutations (named hard and soft) genetic operators. Hard mutation, which is selected with the probability of $p_{m_{hard}}$ replaces a Q-Table with a completely random new one, i.e.

$$Q_i^{j^{new}} = N_i(\mu, \sigma^2) \tag{17}$$

in which $N_i(\mu, \sigma^2)$ is a normal random variable with an appropriate dimension. Soft mutation perturbs each Q-Table in order to search nearby points in the solution space. Therefore, it is a kind of local search commonly used in simulated annealing. Soft mutation, which happens with probability of $p_{m_{soft}}$ is defined as

$$Q_i^{j^{new}} = Q_i^{j^{old}} + N_i(0, \sigma^2),\ \sigma = \eta \left\| Q_i^{j^{old}} \right\|_2 \tag{18}$$

in which we use the individual's matrix Euclidean norm to set the variance of perturbation so that the perturbation be in an appropriate magnitude, i.e. if the matrix's norm is too small, perturbing it too much puts it far from its previous point in the solution space, and vice versa. Crossover operators is chosen as

$$Q_i^{j^{new}} = \alpha Q_i^{j^{old}} + (1-\alpha) Q_i^{k^{old}};\quad \alpha \in U(0,1) \tag{19}$$

in which $U(0,1)$ is a uniform random number between 0 and 1. Doing so, the evolutionary process tries to exploit the solution space by linear combination of parents. This may result in finding a better solution when the fitness landscape between these two points is convex with a maximum between them.

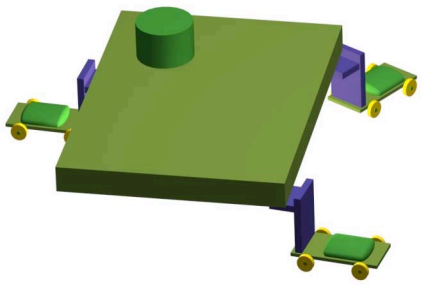

**Fig. 2.** A group of robots lift a bulky object

In order to summarize our approach, the pseudocode of our behavior co-evolution and structure learning method is shown in Table 1.

## IV. Experiments

We tackle a difficult multi-robot object lifting task in order to show the effectiveness of our developed method. This problem had been solved by a traditional approach in [17], i.e. hand-designing the correct behavior and structure in an exhausting trial and error procedure. At the time this problem was solved, it was considered as a challenging problem in multi-robot systems. Results of this paper is based on the simulation of the task.

Imagine a situation in which a group of robots (three in our case) must lift up a bulky and large object (Fig. 2). The object is of such a size and shape that none of the robots can grasp it directly. It is shown in [17] that by keeping the tilt angle of the object in a pre-specified limit, the robots equipped with some compliance are not required to move, the object does not hit the robots, and the system is stable. Moreover, it is easy to design a mechanism for each robot to measure the object's tilt angle in its own coordinate (See [17] for more detailed description). Our goal is finding a suitable architecture that can solve this cooperative multi-robot lifting of an unknown object to a set-point while keeping its tilt angle small with no central control or communication between the robots. For our experiments, we consider four cases: 1) evolution of behaviors and learning of structure, 2) evolution of behaviors with fixed structure (same as hand-designed structure), 3) learning of structure with hand-designed behaviors, and 4) hand-designed behaviors and structure.

Let $z(k)$ be the height of robot-object contact point, $v(k)$ be its elevation velocity, and $\tau(k)$ be the object's tilt angle at time step $k$ (all of these quantities can be measured locally). Hand-designed behaviors and structure are selected similar to [17] (Table 2). We have chosen reinforcement signal by trial and error and benefiting from our intuition about the task ((20) in Table 3). This reinforcement signal acts as the reinforcement signal for the structure learning method and as the fitness evaluator for the behavior

evolution. Although this signal is somehow complex, it is the general belief of reinforcement learning community that the reinforcement function is the most robust and transferable description of the task (see [19]). It is possible that there exists a simpler reinforcement signal that satisfies the designer's subjective evaluation of the agent. Nevertheless, existing a simpler reinforcement signal or not does not do anything with our method as the method is an optimizer of an objective function – be it the most suitable for reinforcing a given task or not.

In order to clarify this function, let us discuss each term of (20). Equation (21) rewards reducing tilt angle and punishes a movement that increases it, (22) rewards being in small tilt angle and punish its largeness. Note that it rewards low tilt angle in early times more than in later time and punishes high angles in the later times more than in the beginning in order to enforce converging to a satisfactory angle sooner. Equation (23) rewards being near the goal and punishes being far from it and (24) punishes passing the goal. At last, (25) punishes a behavior that make a system move too fast.

We make five genetic pools each of them is assigned for a family of behavior by defining an appropriate state and action space for them (Table 4).

The artificial evolution process selects random behaviors from behavior pools and passes them to the agent. The agent learns during a few episodes to organize those behaviors and then returns a measure for its performance. The measure (13) is used to calculate fitness of each behavior (14).

Crossover rate $p_c$ is fixed, but mutation rates ( $p_{m_{hard}}$ and $p_{m_{soft}}$) can be changed in order to increase the diversity while not reducing overall performance. We use following simple heuristic for the changing: denoting $\bar{f}(m)$ as the average fitness of agents (13) in generation $m$, define moving average derivation-like operator as

$$\Delta\bar{f}(m) = \frac{\sum_{k=m-L_1}^{m} \bar{f}(k) - \sum_{k=m-L_2}^{m-L_1+1} \bar{f}(k)}{\bar{f}(m)} \tag{26}$$

in which $L_1$ and $L_2$ define a filtering window size for approximating the derivative. Then, we change mutation rates in the following way:

$$p_{m_{hard}}^{m+1} = \begin{cases} \eta . p_{m_{hard}}^{m} \leftarrow \Delta\bar{f}(m) < -\theta \\ \min(1, \kappa p_{m_{hard}}^{m}) \leftarrow \left|\Delta\bar{f}(m)\right| < \gamma \end{cases} \tag{27}$$

in which $0 < \eta < 1,\ k > 1\, \gamma, \theta > 0$. This rule decreases mutation when the performance degrades noticeably and increases mutation when the performance is almost constant in hope of finding a better solution. A completely similar updating rule is used for $p_{m_{soft}}$. We turn mutation off in the last few generations in order to make average fitness a better estimate of algorithm regardless of the amount of mutation noise. There is nothing special about our mutation rate adaptation and other methods would work similar.

In our simulations, we set initial position of robots as uniformly distributed starting point between 1 and 2 ( $z_i(0) \in U(2,3);\ i = 1,2,3$). Detailed experiment setup is in Table 5. Note that we have not optimized any parameter (e.g. genetic operators' parameters, learning rates, and etc.); therefore, it is possible to get even better results.

The average fitness of agents is depicted in Fig. 3. Evolution generates appropriate behaviors in a few generations. In fact, there exist a combination of behaviors and structures that makes a very good performance from the first generation. However, those behaviors are not dominant in the population at the beginning and the evolution pushes the population toward them gradually. The method that learns structure for the set of hand-designed behaviors of Table 2, which is indeed the constant line as the co-evolution does not change behaviors in this case, performs well but has a partially lower performance comparing with the hand-designed behavior and structure. This is because that in this

TABLE 2. HAND-DESIGNED BEHAVIORS

**Push more:** $v(k+1) = v(k) + \Delta v$
**Do not go fast:** if $v(k) > v_{\max}$ then $v(k) = v_{\max}$ else do nothing
**Stop at goal:** if $z(k) \geq z_{goal}$ then stop ( $v(k+1) = 0$)
**Hurry up:** if $\tau > \tau_0$ and the robot is the lowest one then $v(k) = \min(v(k) + \Delta v, v_{\max})$
**Slow down:** if $\tau > \tau_0$ and the robot is the highest one then $v(k) = \max(v(k) - \Delta v, 0)$
$T^{hand-designed} = [\text{Stop} \quad \text{SlowDown} \quad \text{HurryUp} \quad \text{DontGoFast} \quad \text{PushMore}]$

TABLE 3. REINFORCEMENT SIGNAL DEFINITION

| | |
|---|---|
| $r_k = r_k^{\tau_1} + r_k^{\tau_2} + r_k^{z_1} + r_k^{z_2} + r_k^{v}$ (20) | $r_k^{\tau_1} = \begin{cases} 1 & \tau(k) - \tau(k-1) < -0.5 \\ -0.1 & \text{otherwise} \end{cases}$ (21) |
| $r_k^{\tau_2} = \begin{cases} \frac{1}{\sqrt{k}} & \hat{o}(k) < \hat{o}_0 \\ -0.1\sqrt{k} & \text{otherwise} \end{cases}$ (22) | $r_k^{z_1} = \begin{cases} 1 & \lvert z(k) - z_{goal} \rvert < 0.5 \\ -0.1 & \text{otherwise} \end{cases}$ (23) |
| $r_k^{z_2} = -1 \qquad z(k) > z_{goal} + 0.2$ (24) | $r_k^{v} = -0.1 \qquad v(k) > v_{\max}$ (25) |

TABLE 4. STATE AND ACTION DEFINITION FOR TO BE LEARNT BEHAVIORS

| | | |
|---|---|---|
| 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 | $S'_{\text{Slow}} = \begin{Bmatrix} \text{lowest robot, middle robot,} \\ \text{highest robot} \end{Bmatrix} \times \{"\tau < \tau_0", "\tau > \tau_0"\}$ | $A'_{\text{Slow}} = \{\max(v(k) - \Delta v, 0), NA\}$ |

TABLE 5. EXPERIMENT SETUP

| | |
|---|---|
| *Simulation Parameters* | $v_{\max} = 5$, $\Delta v = 1$, $z_{goal} = 3$, $\tau_0 = 5^\circ$, $\Delta T = 0.005$, $z_i(0) \in U(2,3); i = 1,2,3$, Episode length = 100 steps |
| *Learning Parameters* | Number of episodes = 20, $\alpha_{episode} = 0.1 \times (0.99)^{episode}$, Fitness is calculated in the last 6 episodes. |
| *Evolution Parameters* | Generations = 40, Population size = 20, Roulette wheel selection - 2 best individuals of each pool go directly to the next generation. Initial solutions (for initial population and hard mutation): $N(0,1)$, $N = 100$ (14), $p_c = 0.8$, $p^0_{m_{hard}} = 0.2$, $p^0_{m_{soft}} = 0.5$ (adaptation mechanism) |
| *Mutation rate adaptation* | $\eta = 0.95$, $\theta = 0.2$, $\kappa = 1.06$, $\gamma = 0.01$, $L_1 = 5$, $L_2 = 9$ - Mutation rate is set zero after generation 34. |

case the agent should find an appropriate structure in limited number of episodes (20 episodes). In other words, the agent has less prior knowledge comparing with the hand-designed case (which we consider it as the complete knowledge case).

Both evolving methods are quite good and outperform the hand-designed behaviors/learned structure case and reach close to the hand-designed one after turning the mutation off in generation 35. The one with learning structure performs better in the beginning, but the one with the hand-designed structure performs better ultimately. The difference between these two cases is similar to the difference between hand-designed behavior/structure learning and hand-designed behavior/hand-designed structure. The reason for the superiority of the learning-enabled case in the beginning of the evolution is due to the adaptation of the structure with the current not-so-good behaviors. Note that the hand-designed structure is specifically designed for hand-designed behaviors and not almost random behaviors in the early generations. Also note that inferiority of the cases that do not use prior knowledge (such as hand-designed behaviors or structure) does not mean that they are not suitable because this is our prior knowledge about the problem that enforces the selection of this method or the other.

A sample trajectory of robots and the tilt angle of the object for the agent that its behaviors are evolved and its structure is learnt are shown in Fig. 4. The team behavior is satisfactory as the robots reduce the tilt angle while lift the object to the prescribed goal. These results show that our co-evolutionary scheme can evolve pools of behaviors and the structure learning method can find the correct organization of them that are competitive to what has been designed by an exhaustingly long trial and error in the previous work [17].

## V. CONCLUSIONS

We have proposed a hybrid co-evolutionary/learning scheme to automate the design of behavior-based systems. Artificial co-evolution is used to develop new behaviors and learning is used to organize those behaviors in the architecture. Instead of evolving a single monolithic controller, we have decomposed it into the behavior parts and evolve each behavior separately. This Baldwinian mixture of learning and evolution performed well in our experiment and resulted in behavior-based architectures that are competitive to the hand-designed one. Having a prior knowledge in the form of the correct structure of the agent increases the performance of the system, as the agent does not need to search for an appropriate structure during its lifetime. However in both cases, the performance was satisfactory. Our hybridization of learning and evolution is different from most other approaches. In our approach, learning and co-evolution are searching two different parts of the problem space: hierarchy space and behavior space. Learning optimizes the objective function by arranging a set of limited number of behaviors in the structure and evolution

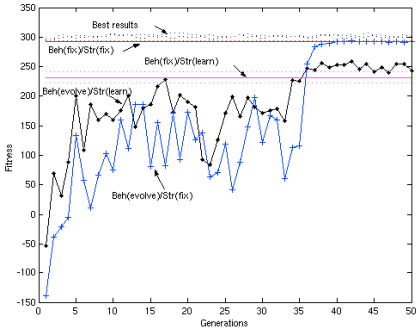


**Fig. 3.** Average fitness comparison for different design methods during generations: 1) beh. evolution and str. learning, 2) beh. evolution with hand-designed str., 3) hand-designed beh. with str. learning, and 4) hand-designed beh. and str. . Best results of the first two evolving cases (1 and 2) are also depicted by dotted lines. Dotted lines across the constant line of the hand-designed beh. and str. learning case (3) show one standard deviation region across the mean performance.

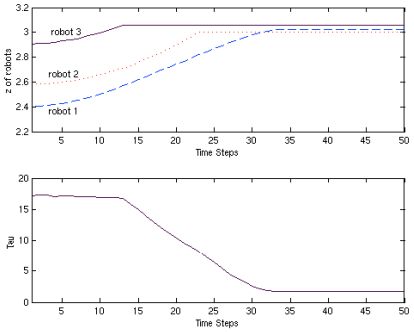


**Fig. 4.** A sample trajectory showing the position of three robots and the tilt angle of the object during object lifting after sufficient evolution and learning.

optimizes the same objective function by changing behaviors' internal mapping. Learning acts as the fast adaptation mechanism for the agent and co-evolution acts as the slow but global optimizer. Generally speaking, our method is not a mere local search for refining the result of the global optimizer.

There are some important directions for our future research. Benefiting from the lifetime experience of the agent to provide the co-evolutionary mechanism a better (and less noisy) estimate of the fitness of each behavior may facilitate the co-evolutionary process. In our current method, all contributing behaviors are assigned the same fitness. However, usually different behaviors contribute in the agent's overall performance differently. One possible way to evaluate each behavior's contribution is looking at the rewards and punishments it received during the agent's life. Learned structures of other agents may be a good prior knowledge for the structure of newborn agents. It can provide a helpful initial bias for newborn agents; the new agents do not need to explore the whole structure space to find a suitable arrangement. Finally, it would be helpful to apply this kind of problem decomposition (in which evolution and learning seek for the solution of the problem in different spaces) to other problem domains. Hierarchical behavior-based systems are good examples of such problems. One may wonder if this kind of decomposition can be applied to other problems too. Problems that adjusting both structure and components are possible might be amenable to this hybridization.

## REFERENCES


[1] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robotics and Automation R.A-2*, pp. 14-23, 1986.

[2] R. A. Brooks, "Intelligent without representation," *Artificial Intelligence*, 47, 1991, pp. 139-159.

[3] R. A. Brooks, "A robot that walks: emergent behavior from a carefully evolved network," *Neural Computation* 1(2), 1989, pp. 252-262.

[4] 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, 1996, pp. 310–317.

[5] 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, 1998, pp. 82-87.

[6] L. Parker, "ALLIANCE: an architecture for fault-tolerant multi-robot cooperation," *IEEE Trans. on Robotics and Automation*, 14 (2), 1998, pp. 220-240.

[7] P. Maes and R. A. Brooks, "Learning to coordinate behaviors," in *Proc. AAAI-90*, 1990, pp. 796-802.

[8] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, 55, 1992, pp. 311-365.

[9] 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), 2001, pp. 81-93.

[10] 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 Systems*, vol. 2, S. J. Hanson, T. Petsche, M. Kearns, and R.L. Rivest, Eds., The MIT Press, 1994, pp. 321-346.

[11] J. Togelius, "Evolution of a subsumption architecture neurocontroller," *J. Intelligent and Fuzzy Systems*,15:1, 2004, pp. 15-20.

[12] S. Chernova and M. Veloso, "An evolutionary approach to gait learning for four-legged robots," In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2004, pp. 2562-2567.

[13] A. M. Farahmand, M. Nili Ahmadabadi, and B. N. Araabi, "Behavior hierarchy learning in a behavior-based system using reinforcement learning," In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2004, pp. 2050-2055.

[14] R. S. Sutton, and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[15] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Systems Journal: Special Issue on Reinforcement Learning*, vol. 13, 2003, pp. 41-77.

[16] P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," In *Handbook of Metaheuristics,* F. Glover and G. Kochenberger, Eds., Kluwer Academic Publishers, Boston MA, 2003, pp. 105-144.

[17] F. Gomez and R. Miikkulainen, "Incremental evolution of complex general behavior," *Adaptive Behavior*, 5, 1997, pp. 317-342.

[18] M. Nili Ahmadabadi and E. Nakano, "A constrain and move approach to distributed object manipulation," *IEEE Trans. Robotics and Automation*, vol. 17, no. 2, 2001, pp. 157-172.

[19] A. Ng, S. Russell, "Algorithms for inverse reinforcement learning," In *Proc. of the Seventeenth International Conference on Machine Learning*, 2000.