Evolutionary Development of Hierarchical Learning Structures

Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I. Christensen

***Abstract*—Hierarchical reinforcement learning (RL) algorithms can learn a policy faster than standard RL algorithms. However, the applicability of hierarchical RL algorithms is limited by the fact that the task decomposition has to be performed in advance by the human designer. We propose a Lamarckian evolutionary approach for automatic development of the learning structure in hierarchical RL. The proposed method combines the MAXQ hierarchical RL method and genetic programming (GP). In the MAXQ framework, a subtask can optimize the policy independently of its parent task's policy, which makes it possible to reuse learned policies of the subtasks. In the proposed method, the MAXQ method learns the policy based on the task hierarchies obtained by GP, while the GP explores the appropriate hierarchies using the result of the MAXQ method. To show the validity of the proposed method, we have performed simulation experiments for a foraging task in three different environmental settings. The results show strong interconnection between the obtained learning structures and the given task environments. The main conclusion of the experiments is that the GP can find a minimal strategy, i.e., a hierarchy that minimizes the number of primitive subtasks that can be executed for each type of situation. The experimental results for the most challenging environment also show that the policies of the subtasks can continue to improve, even after the structure of the hierarchy has been evolutionary stabilized, as an effect of Lamarckian mechanisms.**

***Index Terms*—Autonomous development, genetic programming (GP), hierarchical reinforcement learning (RL), Lamarckian evolution.**

I. Introduction

HIERARCHICAL reinforcement learning (RL) algorithms [2], [14], [19] can learn a policy faster than flat (nonhierarchical) RL algorithms, by capturing a hierarchical structure of the task. However, the hierarchical frameworks put a burden on the human designer, who has to design in advance, each module in the hierarchy, and most problematically the hierarchy itself. The construction of a hierarchy, i.e., the decomposition of an overall task into smaller suitable subtasks is, in general, a difficult problem. The difficulty arises from the fact that the human designer's point of view of a task differs greatly from the agent's point of view. Generally, human designers' construct the hierarchy by combining high-level behaviors, such as avoidance and approach, that seem suitable for achieving the task in general. However, the performance is dependent on the sensorimotor interaction between the agent and the environment, and the interactions between the modules, which are difficult to predict beforehand.

In this paper, we propose a Lamarckian evolutionary approach, by combining genetic programming (GP) [11] and the MAXQ hierarchical RL framework [2], for developing a hierarchy that is adapted to the given task and environment. In the proposed method, the MAXQ method learns the policy based on the hierarchies obtained by the GP, while the GP explores the appropriate hierarchies using the result of the MAXQ method. We have chosen to use Dietterich's MAXQ framework to represent the hierarchical learning structures, because: 1) MAXQ's tree-based representation of the learning structures provides a straightforward approach for mixing hierarchies by the GP crossover operator; 2) each subtask can optimize its own policy independently of higher level subtasks, which makes it possible to reuse learned policies after applying crossover operations; 3) it is very easy to assign different reward functions, state space abstractions, and primitive actions to different subtasks; and 4) the rigorous mathematical foundation of Dietterich's approach.

This research has been inspired by the philosophical ideas presented in Minsky's book "Society of Mind" [12]. Minsky introduced the concept that complex intelligent behaviors emerge as the result of competing primitive agents in the brain. Although we use an evolutionary algorithm, the process is what should happen during a single agent's lifetime. The nervous system should have some sort of architecture search mechanism, which is approximated here by GP. In the proposed method, different learning structures compete by the means of a Lamarckian evolutionary process. We consider that the agents already have the knowledge to execute primitive behaviors, such as avoiding an obstacle and approaching a target. The role of learning and evolution are, therefore, not to improve low-level motor actions, but to combine the primitive behaviors into higher level modules, adapted to the current task and environment.

We are interested in the interaction between learning and evolution. Our general approach is to use RL to learn the task at hand and use the evolutionary search process for optimizing meta-aspects of the learning, such as meta-parameters [7],

Manuscript received June 21, 2005; revised October 3, 2005. This research was conducted as part of "Research on Human Communication" with support from the Telecommunications Advancement Organization of Japan. The work of S. Elfwing was supported in part by a shared grant from the Swedish Foundation for Internationalization of Research and Education (STINT) and in part by the Swedish Foundation for Strategic Research (SSF).

S. Elfwing is with the Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology, Okinawa 904-2234, Japan, and also with the Computer Vision and Active Perception Laboratory, Centre for Autonomous Systems, Kungl Tekniska Högskolan, 100 44 Stockholm, Sweden (e-mail: elfwing@oist.jp).

E. Uchibe and K. Doya are with the Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology, Okinawa 904-2234, Japan (e-mail: uchibe@oist.jp; doya@oist.jp).

H. I. Christensen is with the Computer Vision and Active Perception Laboratory, Centre for Autonomous Systems, Kungl Tekniska Högskolan, 100 44 Stockholm, Sweden (e-mail: hic@kth.se).

Digital Object Identifier 10.1109/TEVC.2006.890270

1089-778X/$25.00 © 2007 IEEE

switching of primitive behaviors [6] and, in this study, hierarchical learning structures. While there is very little biological evidence for the benefits of combining learning and evolution, it has been shown for evolutionary computation that the addition of learning techniques can improve the adaptivity and thereby increase the speed of the evolutionary process [8]. These improvements can originate from the biologically plausible Baldwin effect [20], i.e., indirect genetic assimilation of learned traits, or from biologically disproved Lamarckian mechanisms [9], i.e., inheritance of learned traits. In general, it has been shown for artificial evolution that Lamarckian evolution is more effective in a stationary environment [1], while standard Darwinian evolution is more effective in a nonstationary environment [16].

A secondary objective of this study, in addition to the development of hierarchical learning structures, is to investigate the potential benefits of Lamarckianism for accelerating the learning of the policies in the subtasks. The MAXQ framework enables reusing of learned subtask policies after genetic operations, which minimizes the need for learning the subtask policies from scratch in each generation, and the learning can, therefore, continue over several generations. In our experimental setup, the environmental conditions do not change during the evolution and the learning time for the individuals in one generation is relatively short, which suggests a good case for Lamarckian learning. Positive evidence for a Lamarckian learning effect would be if the learning performance continues to improve after the hierarchical structure has been evolutionary obtained, and if the obtained hierarchies have significantly higher performance compared with hierarchies with identical learning structures, but the learning of the subtask policies reset to scratch.

The studies most related to our work have been performed by Downing [5] and Iba [10], i.e., research that combines tree-based GP and RL. Their results showed that a hybrid approach can benefit both evolution and learning. The RL can accelerate the evolution by means of the Baldwin effect, and/or Lamarckian mechanisms. Additionally, the evolutionary search provided by the GP can construct a proper state abstraction for the current task and environment, which is especially valuable for high-dimensional state spaces. It should be noted that Downing and Iba have only evaluated their methods in grid-world navigation tasks, where the agents can access global position information. The main difference between our study and the earlier studies is that they integrate GP and flat RL to achieve a more efficient state space construction in flat RL, whereas we use the GP as a tool for optimizing the hierarchical structures used for MAXQ.

The remainder of this paper is organized as follows. After a short review of standard RL and a presentation of the MAXQ framework, in Section II, we present our method of combining Lamarckian GP and the MAXQ hierarchical RL method in Section III. In Section IV, we describe our foraging task, the subtasks, and the three different environmental settings used in our simulation experiments. Thereafter, in Section V, we present the results of the evolution experiments in the three environments, showing that an optimal task decomposition strongly depends on the current task environment. The main conclusion is that the evolution can find a minimal strategy, i.e., a hierarchy that minimizes the number of primitive subtasks that can be executed in each type of situation. If a subgoal requires more than one primitive subtask to be accomplished, there can be a Lamarckian learning effect that optimizes the learning in the hierarchy over generations, even if the hierarchy itself has been evolutionary stabilized. We conclude with a discussion of the results and future research directions.

II. Background

We begin with a short review of the basics of flat RL and, thereafter, we present the MAXQ framework, which has been used for representing the hierarchical learning structures in this study.

A. Basics of Reinforcement Learning (RL)

RL [18] is a computational approach to learning from interaction with the environment. An agent learns a policy, state-to-action mapping, based on scalar reward signals received from the environment. At time t, the agent observes the current state $s_t \in S$, the set of states of the environment. Based on the current policy $\pi_t = P(a|s)$, the agents selects an action $a_t \in A$, the set of primitive actions. The environment makes a state transition from s_t to s_{t+1}. According to the action a_t, the agent receives a reward r_t. The goal is to learn a policy, π, that maximizes the cumulative discounted future reward. The value of a state s, the state-value function, under policy π is given by Bellman equation as

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P(s'|s,a) \left[R(s'|s,a) + \gamma V^{\pi}(s')\right] \tag{1}$$

where $R(s'|s,a)$ is the reward for taking action a in state s, resulting in the new state s', and γ, $0 < \gamma < 1$, is the discount factor of future rewards. Similarly, the value of selecting action a in state s, the action-value function is given as

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a) \left[R(s'|s,a) + \gamma \sum_{a'} \pi(s',a') Q^{\pi}(s',a')\right]. \tag{2}$$

The learning in the MAXQ framework is based on Q-learning [21] and SARSA [15]. Q-learning is an *off-policy* RL algorithm, which learns an estimate of the action value function Q, independent of the policy being followed, according to:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \tag{3}$$

where α is the learning rate. SARSA is an *on-policy* RL algorithm, which learns an estimate of the action value function Q, while the agent follows policy π, according to:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a)\right]. \tag{4}$$

In both Q-learning and SARSA, the policy π is derived from the action values. The probably most common action selection method is *ϵ-greedy*, where the agent selects the greedy action, $a^* = \operatorname{argmax}_a Q(s,a)$, most of the time, but with a small probability ϵ, the agent selects a random action. An alternative

is *softmax* action selection, where the actions are ranked and weighted according to their action values. The most common softmax method uses a Boltzmann distribution and selects an action a, with probability

$$P(a) = \frac{e^{\beta Q(s,a)}}{\sum_b e^{\beta Q(s,b)}} \tag{5}$$

where the positive parameter β is the inverse temperature.

B. The MAXQ Framework

Hierarchical RL algorithms are methods for introducing state abstraction to the RL framework, to be able to apply RL to complex large-scale problems. The goal of hierarchical RL is to exploit hierarchical structures in complex Markov decision processes (MDPs). This is realized by decomposing an overall task into smaller suitable subtasks. This gives a hierarchy of the task, where the actions of the subtasks are other subtasks or primitive commands/subtasks. Many of the subtasks represent abstract subgoals that cannot be accomplished in one time step. The actions of the subtasks are, therefore, extended in time and a subtask, i.e., an action of a higher level subtask, is active until a well-defined termination criterion is fulfilled. As the subtasks can be active for variable amount of time steps N, hierarchical RL can be defined as a semi-Markov decision process (*semi-MDP*). The state transition probability function is a joint distribution of the result state s' and the number of time steps N, when action a is performed in state s: $P(s', N|s, a)$, with the expected reward $R(s', N|s, a)$. The Bellman equations are thereby changed to

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s',N} P(s',N|s,a) \left[R(s',N|s,a) + \gamma^N V^\pi(s')\right] \tag{6}$$

$$Q^\pi(s,a) = \sum_{s',N} P(s',N|s,a) \left[R(s',N|s,a) + \gamma^N \sum_{a'} \pi(s',a') Q^\pi(s',a')\right]. \tag{7}$$

The objective of learning in the MAXQ framework, given a *MAXQ graph*, i.e., a task decomposition (see Fig. 1 for a naive task decomposition of the foraging task used in our experiments), is to find a *recursively optimal policy*, defined as the following.

A recursively optimal policy for MDP M with MAXQ decomposition $\{M_0, \ldots, M_n\}$ is a hierarchical policy $\pi = \{\pi_0, \ldots, \pi_n\}$ such that for each subtask M_i, the corresponding policy π_i is optimal for the semi-MDP defined by states S_i, the set of action A_i, the state transition probability function $P^\pi(s', N|s, a)$, the reward function of the original reward function $R(s'|s, a)$, and the pseudo-reward function $\tilde{R}(s')$.

Recursive optimality is a form of local optimality in which the policy at each node is optimal given the policy of its children. A subtask tries to find the optimal policy without reference to the parent node's policy.

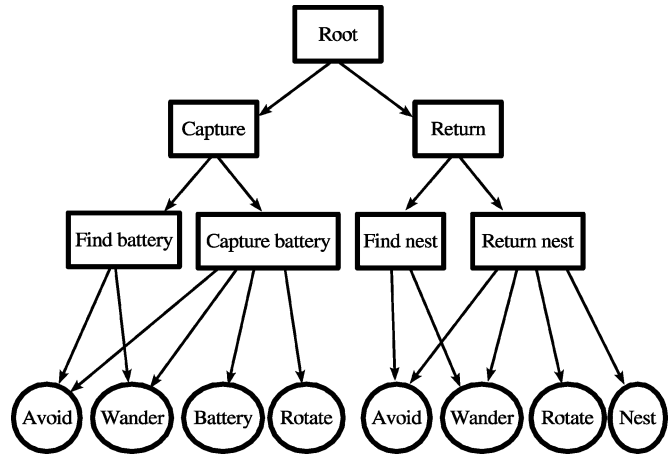

Fig. 1. Naive task decomposition of the foraging task used in our experiments, including all composite subtasks (inner nodes) and primitive subtasks (leaf nodes) provided to the agent. The overall task (Root) is decomposed into two main tasks: find and capture a battery (Capture), and return the battery to the nest (Return). The main capturing subtasks is decomposed into finding the battery (Find battery), and capturing a visible battery (Capture battery). The main returning subtasks is decomposed into finding the nest (Find nest), and returning the battery to the visible nest (Return nest). The composite subtasks in the lowest abstraction layer then use the available primitive subtasks (Avoid, Wander, Battery, Nest, and Rotate) to achieve their goals. Note that the primitive subtasks are shared by all hierarchies and composite subtasks in our framework. The duplication of primitive subtasks in the figure is only for illustrative purposes.

Formally, the MAXQ task decomposition requires a given MDP M to be decomposed into a set of subtasks $\{M_0, M_1, \ldots, M_n\}$, where M_0 is the root subtask. A subtask is a three-tuple $\langle T_i, A_i, \tilde{R}_i \rangle$ defined as follows.

- T_i is a termination predicate that partitions states S into a set of *active states* S_i, and a set of *terminal states* T_i. The policy for a subtask M_i can only be executed if the current state $s \in S_i$.
- A_i is a set of actions that can be performed to achieve subtask M_i. These actions can either be primitive actions from A or they can be other subtasks, denoted by their indexes i.
- $\tilde{R}_i(s')$ is the *pseudo-reward* function, which specifies a pseudo-reward for each transition from a state $s \in S_i$ to a terminal state $s' \in T_i$. The pseudo-reward tells how desirable each of the terminal states is for the subtask. Typically, goal terminal states have a pseudo-reward of 0 and all nongoal terminal states have negative pseudo-rewards.

Each primitive action a from A is a primitive subtask in the MAXQ decomposition, which is always executable, terminates immediately after execution, and its pseudo-reward function is uniformly zero.

The value function $V^\pi(s)$ for executing a hierarchical policy π is called the *projected value function*. The projected value function is the value for executing π, starting in state s and at the root of the hierarchy. The projected value function $V^\pi(i, s)$ for subtask i in state s is decomposed into two parts, defined as

$$V^\pi(i,s) = \begin{cases} Q^\pi(i,s,\pi_i(s)), & \text{if } i \text{ is composite} \\ \sum_{s'} P(s'|s,i) R(s'|s,i), & \text{if } i \text{ is primitive} \end{cases} \tag{8}$$

where R is the reward function for the primitive subtasks and $Q^\pi(i,s,a)$ is recursively defined as

$$Q^\pi(i,s,a) = V^\pi(a,s) + C^\pi(i,s,a). \tag{9}$$

$C^\pi(i,s,a)$ is called the *completion function* and is defined as the discounted cumulative reward for completing subtask M_i after invoking the subroutine for subtask M_a in state s

$$C^\pi(i,s,a) = \sum_{s',N} P_i^\pi(s',N|s,a)\gamma^N Q^\pi(i,s',\pi(s')). \tag{10}$$

(8)–(10), the *decomposition equations*, tell how to decompose the projected value for the root $V^\pi(0,s)$ into projected value functions for the individual subtasks $\{M_0, M_1, \ldots, M_n\}$, and the individual completion functions $C^\pi(j,s,a)$ for $j = 1, \ldots, n$. The projected value function is stored explicitly as V values for all primitive actions and implicitly as C values for all composite subtasks.

In general, the decomposition of the projected value function can be expressed as

$$V^\pi(0,s) = V^\pi(a_m,s) + C^\pi(a_{m-1},s,a_m) + \cdots + C^\pi(a_1,s,a_2) + C^\pi(0,s,a_1) \tag{11}$$

where $a_0, a_1, \ldots, a_m$ is the path of nodes from the root node to a primitive leaf node, given by the hierarchical policy π and the current state s.

To be able to learn recursively optimal policies, the MAXQ method uses two completion functions, C and $\tilde{C}$. C is the normal completion described above [(10)]. C is used by the parent task to compute $V(i,s)$, the expected reward for performing action i starting in state s. The second completion function $\tilde{C}$ is only used inside node i in order to discover the local optimal policy for M_i. $\tilde{C}$ uses both the original reward function $R(s'|s,a)$, and the pseudo-reward function $\tilde{R}_i(s')$. The learning rules for updating $\tilde{C}$, similar to Q-learning [(3)] for semi-MDP, and C, similar to SARSA [(4)] for semi-MDP are defined as

$$\tilde{C}(i,s,a) \leftarrow \tilde{C}(i,s,a) + \alpha(i)\Big\{\gamma^N\big[\tilde{C}(i,s',a^*)\tilde{R}_i(s') + V(a^*,s)\big] - \tilde{C}(i,s,a)\Big\} \tag{12}$$

$$C(i,s,a) \leftarrow C(i,s,a) + \alpha(i)\Big\{\gamma^N\big[C(i,s',a^*) + V(a^*,s')\big] - C(i,s,a)\Big\} \tag{13}$$

where $a^* = \mathrm{argmax}_{a'}\big[\tilde{C}(i,s',a') + V(a',s')\big]$. The updating of the value function in primitive subtasks are accomplished by SARSA or Q-learning, where the discount factor γ is set to zero

$$V(i,s) \leftarrow V(i,s) + \alpha[r - V(i,s)]. \tag{14}$$

III. Exploration of MAXQ Graphs by GP

By introducing hierarchical learning structures, the MAXQ method can learn a good policy faster than flat RL methods. However, the MAXQ framework provides no way of learning the structure of the hierarchy, which was duly noted in the introduction of Dietterich's original work. This section presents our method of using GP to develop an appropriate hierarchy.

A. MAXQ Modifications

One aim of this study is that the agent shall behave as an autonomous agent, using local and continuous state information. We suspect that one reason for the limited research activities in methods for automatic construction of hierarchies is that most MAXQ studies only consider toy examples in a discrete grid world, where the interaction between agent and environment plays a limited role in the construction of a good hierarchy. The original MAXQ algorithm is also formulated for the case where each primitive subtask consists of a single action. In this paper, we consider a foraging task for a mobile robot, where the primitive subtasks consist of primitive behaviors using prelearned policies and terminal conditions. Therefore, instead of storing the state-value function directly, the primitive subtasks store the action-value function $Q^\pi_{\text{prim}}(i,s,a)$, which modifies (8) to

$$V^\pi(i,s) = \begin{cases} Q^\pi(i,s,\pi_i(s)), & \text{if } i \text{ is composite} \\ \max_a Q^\pi_{\text{prim}}(i,s,a), & \text{if } i \text{ is primitive} \end{cases} \tag{15}$$

where $Q^\pi_{\text{prim}}(i,s,a)$ is defined as

$$Q^\pi_{\text{prim}}(i,s,a) = \sum_{s'} P(s'|s,a)\left[R(s'|s,a) + \gamma\sum_{a'}\pi(s',a')Q^\pi_{\text{prim}}(i,s',a')\right] \tag{16}$$

which is identical to the standard RL formulation [(2)]. To handle the continuous state input, the value functions and completion functions are approximated by linear normalized Gaussian RBF networks, see the Appendix for details.

We have used softmax action selection with a Boltzmann distribution [(5)], where the inverse temperature β is exponentially increased, i.e., multiplied with the parameter c_β, after the subtask has terminated as

$$\beta \leftarrow c_\beta \beta. \tag{17}$$

Experience has shown that it is important to keep an amount of stochasticity in the action selection through the learning, to prevent the agent from getting trapped in suboptimal behaviors. Therefore, we have set a maximum probability (0.98 in our experiments) for selecting an action, to ensure that the agent will continue to select exploratory actions. In such cases, ϵ-*greedy* ($\epsilon = 0.02$) action selection is used. Note that this is also the case for the primitive subtasks, so although the primitive subtasks do not update their value functions during the evolution, their action selection is still stochastic.

B. Evolutionary Scheme

Learning and evolution are two forms of adaptation that occur on different time scales. Learning operates on a single individual within one generation, while evolution operates on the whole population over many generations. We, therefore, find it natural to use RL to learn the task at hand and the evolutionary search

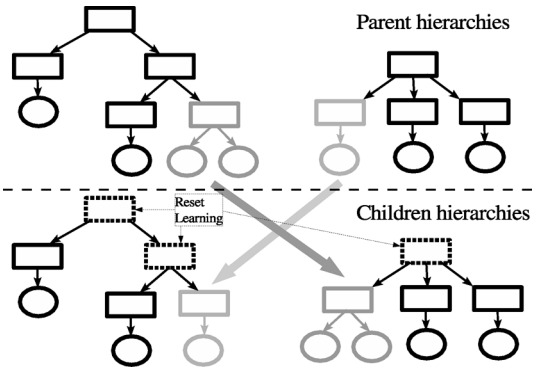

Fig. 2. Crossover operation between two MAXQ graphs. The parent hierarchies switch subtrees at randomly selected crossover points. Note that the new parent has to exist in the parent list of the root node of the switched subtree. After the crossover, the new hierarchies reset the learning of the subtasks in the path from the root to the parent of new subtree, dotted rectangles in the figure.

process to optimize meta-aspects of the learning, such as meta-parameters, switching of primitive behaviors and, in this study, hierarchical learning structures.

The two main motivations for using the MAXQ framework for representing the hierarchical learning structures are that: 1) the tree-based MAXQ graphs provide a straightforward approach for mixing hierarchies by the GP crossover operator and 2) the fact that a subtask learns to optimize its own policy without reference to the parent node's policy, which makes it possible to preserve learned structures under genetic crossover manipulations. Our combination of the MAXQ framework and GP is, therefore, seamless algorithmically and also has the potential of benefiting from computationally interesting Lamarckian mechanisms.

Our basic assumption for this study is that the artificial evolution is seen as a process for developing the strategy or hierarchy for a given task and environment during the lifetime of one agent. The agent already knows how to perform primitive behaviors, i.e., the primitive subtasks in the MAXQ graphs. The role of the evolutionary process is, therefore, to combine the already learned primitive behaviors into more complex and abstract learning structures by mixing competing learning strategies according to how well they perform the learning task. It should be noted that this assumption requires that the implementer designs the available subtasks in the same manner as for standard MAXQ implementations. We have thereby limited this study to optimization of the hierarchy and an investigation of the benefits of combining learning and evolution in this framework.

In our method, the genome is represented by the MAXQ graph, where the function and terminal sets are the composite and primitive subtasks, respectively. As the different subtasks have different state spaces, goals and termination criteria, it is natural that all subtasks are not allowed to be parent nodes to all other subtasks. We have, therefore, used a form of *strongly typed GP* [13]. In general, strongly typed GP allows the designer to assign a type to the arguments and the return value of each function. In our case, this means that the subtasks store a list of the subtasks that are allowed to be parent nodes.

To realize the mixing of the competing learning structures, we use the GP crossover operator, as illustrated in Fig. 2. The parent hierarchies switch subtrees at randomly selected applicable crossover points, i.e., the new parent node exists in the list of valid parent nodes of the root node of the switched subtree. After crossover, it is necessary to reset the learning in the subtasks, where the policy is affected by the operation, i.e., in the path from the root to the parent of new subtree as indicated by the dotted rectangles in the figure. By resetting the learning, we mean that the values for the function approximation of $\tilde{C}$, C, and Q_{prim}, and the inverse temperature β are set to their initial values.

Alg. 1. Pseudocode for our evolutionary scheme.

1) **Pretrain** the primitive subtasks within the MAXQ framework;

2) **Initialize** population;

3) **repeat**

4) **Evaluate** the fitness of each hierarchy, according to the performance of the last half of the trials;

5) **Reproduce** a fixed number of individuals;

6) **Create** the remaining hierarchies by crossover, where the parents is given by tournament selection;

7) **until** *fittest half of the population has identical structure;*

Our evolutionary scheme is shown in Alg. 1. The primitive subtasks are first trained by a hand-coded MAXQ hierarchy that contains all primitive subtasks. After convergence, the action-values are saved to evolutionary process, where all hierarchies share the obtained primitive behaviors. The population is initialized with a variety of hierarchies: from simple solutions containing no composite subtasks except for the root task, to complex solutions containing all primitive and composite subtasks. Each hierarchy is evaluated according to the number of time steps it takes to successfully complete a fixed number of trials. The fitness is computed as the number of time steps to complete the last half of the learning trials, where a short time represents a high fitness value. A new generation is created by: 1) elitism: a fixed number of the fittest hierarchies is reproduced and 2) crossover: the remaining part of the population is created by crossover, where the parents are given by tournament selection.

To be able to compute the difference in structure between hierarchies, we have constructed a simple difference measurement. The structural difference is the number of subtasks that differs between two hierarchies, where subtasks in different abstraction layers are assigned different values. A difference in highest abstraction level (subtasks Capture and Deliver) gives a value of 4, a difference in the next abstraction layer (the remaining composite subtasks) gives a value of 2, and a difference at the lowest level (all primitive subtasks) gives a value of 1.

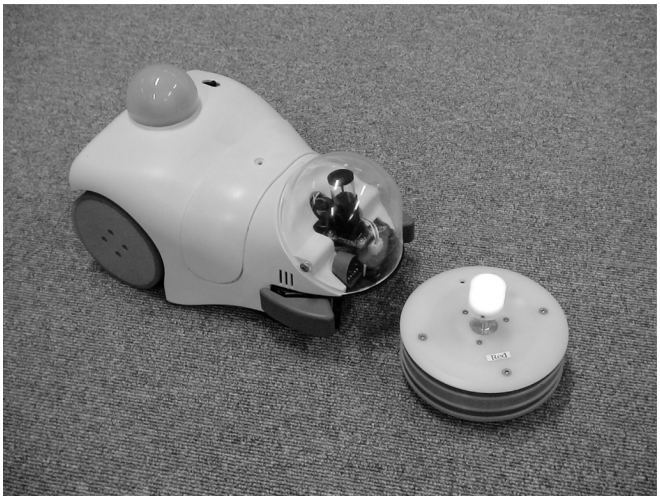

Fig. 3. The CR robot with battery pack.

IV. Experimental Setup

A. Cyber Rodent (CR) Robot

This study has been performed within the Cyber Rodent (CR) project [4]. The main goal of the CR project is to study the adaptive mechanisms of artificial agents under the same fundamental constraints as biological agents, namely, self-preservation and self-reproduction. The CR, shown in Fig. 3, has two main features: the ability to capture and recharge from battery packs in the environment for self-preservation, and to exchange data and programs via IR-communications for self-reproduction.

The CR is a two-wheel mobile robot equipped with an omnidirectional vision system, eight distance sensors, color LEDs for visual signaling, an audio speaker, and microphones.

The experiments presented in this paper have been performed in a MATLAB simulator developed for the CR project. To make the simulations more realistic, we added uniformly distributed noise to the sensor inputs and the action outputs in the ranges (relative to the full range of $[-1, 1]$): $[-0.1, 0.1]$ for the distance sensors and the distance information from the vision system, $[-0.05, 0.05]$ for the angle information from the vision system, and $[-0.2, 0.2]$ for the two wheel speeds of all actions. Our recent study [6] showed that primitive behaviors learned by RL in simulation could be directly transferred to the hardware with reasonable performance.

B. RL Task

The learning task used in this paper is foraging of battery packs: the CR should find a battery pack, capture the battery pack, find the nest, and then return the battery pack to the nest. To solve the task, the robot relies only on local state information received by the robot's sensory system: distances from the five front proximity sensors, distance and angle to the closest battery pack and the nest, if they are visible. The angular range of the vision system is $[-60°, 60°]$ and within the field of view both the battery packs and the nest are visible from all distances, if they are not obscured by obstacles. In short, we have only used the state information that is available for the real CR robot.

TABLE I
Four Sequential Subgoals for the Foraging Task

Subgoal	Relevant state info to accomplish goal
Find a battery	Proximity sensors
Capture a battery	Proximity sensors Angle & distance to battery
Find the nest	Proximity sensors
Deliver battery to nest	Proximity sensors Angle & distance to nest

TABLE II
Provided Composite Subtasks and Their Goal and Nongoal Terminal States

Composite Subtask	Goal terminal state	Non-goal terminal states
Root	Battery delivered to nest	-
Capture	Battery captured	-
Deliver	Battery delivered to nest	Battery not captured
Find battery	Battery visible	Battery captured
Capture battery	Battery captured	Battery not visible
Find nest	Nest visible	Battery not captured
Return nest	Battery delivered to nest	Battery not captured, Nest not visible

We designed the foraging task to be simple, but with enough variability to serve our purpose of: 1) showing that our method is able to find hierarchies that are suitable for both the task and the current environmental setting; 2) demonstrating the importance of taking the environment, and not only the task itself, in consideration when performing task decomposition; and 3) investigation of the potential benefits from Lamarckianism.

During the execution of task, the agent has to accomplish four sequential subgoals, used as goal terminal states for the composite subtasks, as shown in Table I. The table also shows the relevant local state information available to the CR to accomplish each subgoal.

Fig. 1 shows the hierarchy that was used to train the primitive subtasks. It contains all available subtasks, composite and primitive, that is provided to the CR. This naive task decomposition is based on the intuition that the composite subtasks should be related to the subgoals. The primitive subtasks should be connected to the lowest abstraction layer, and use all available primitive subtasks for each composite subtask.

Table II shows the available composite subtasks and the goal and nongoal terminal states for each subtask. Each of the four subtasks in the lowest abstraction layer, Find battery, Capture battery, Find nest, and Return nest, are designed to accomplish one of the subgoals, and are therefore only active in the states related to that subgoal. The subtasks in the next abstraction layer, Capture and Deliver, are active in the states related to the two main tasks, i.e., capture a battery and deliver the battery to the nest, respectively. The top abstraction layer, Root, is of course active for all states.

TABLE III
PRELEARNED PRIMITIVE SUBTASKS FOR THE FORAGING TASK

Subtask	Behavior	Active states	State input
Avoid	Avoid hitting obstacles	All	Proximity sensors
Wander	Explore the environment	All	Proximity sensors
Battery	Approach and capture a battery	Battery visible	Angle to battery
Nest	Return to the nest	Nest visible	Angle to nest
Rotate	Rotate to a battery or the nest	Battery or nest visible	Angle to battery or the nest

For all composite subtasks, the pseudo-reward $\tilde{R}_i$ is set to 0 for the goal terminal states and -1 for the nongoal terminal states. A composite subtask terminates if it reaches a terminal state and it can only be selected if it is active, i.e., the current state is not a terminal state. Additionally, a composite subtask will also terminate if it experiences a state for which it cannot select any action, i.e., the current state is a terminal state for all its children nodes. If this situation occurs for the root node, i.e., that the root cannot select any action, the hierarchy is labeled as nonfunctional and is not allowed to be reproduced.

The prelearned primitive subtasks are shown in Table III. The primitive subtasks are simple reactive behaviors using one type of state input and four or five discrete actions to accomplish their tasks, where the actions are pairs of right and left wheel velocities.

To explore the environment and avoid obstacles the CR can execute two types of exploration behaviors: Wander and Avoid. Wander executes more forward directed actions with higher speeds, intended to be used for exploration in open areas. Avoid executes actions that are intended primarily for avoiding obstacles and preventing the CR from being trapped at corners. Avoid has therefore, one backward action and two, one in each direction, rotating actions, but lacks a straight forward action.

The reward functions R_i for the primitive subtasks are designed to promote the CR to move a long distance forward and keep a small angle to a target, i.e., the closest battery pack or the nest in each time step. The reward given in each time step is in the interval $[-1, 0]$. The reward functions are functions of either movement, for Avoid and Wander, or the angle to a target, for Battery, Nest, and Rotate. The subtasks receive maximum negative reward -1 if the subtasks fail, i.e., if the smallest proximity sensor reading is below the minimum effective range, 70 mm, for Avoid and Wander, or if the target is not visible anymore, for Battery, Nest, and Rotate.

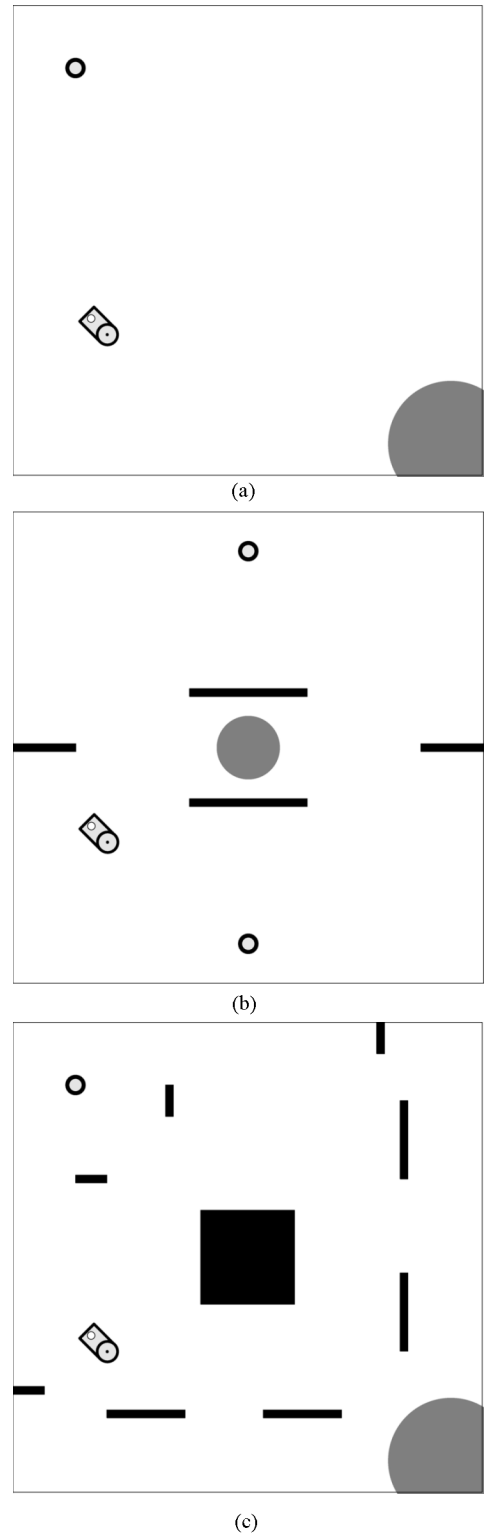

Fig. 4. The three different environmental settings used for evaluating our method. The small filled light gray circles with black boundaries represent the battery packs, and the big darker gray circles represent the nests. (a) Environment 1. (b) Environment 2. (c) Environment 3.

C. Environmental Settings

To evaluate the performance of the proposed method, it has been tested in three different environmental settings, shown as snapshots from the MATLAB simulator in Fig. 4. The small filled light gray circles with black boundaries represent the battery packs, and the big darker gray circles represent the nests.

At the beginning of each trial, the CR starts in the nest area, and captures the battery if the center of the head circle reaches a battery circle. The trial ends if the center of the head circle reaches the nest area after a successful battery capturing. A trial is also terminated if the CR has used more than a fixed number

of time steps to complete the task (2000 in our experiments). The next hierarchy in the population starts from the final position of the previous hierarchy, except when the previous hierarchy could not complete the task in time, in which case the CR is moved back to nest area for the new trial. The CR can only receive visual information from the nest and battery if the centers of the targets are visible, i.e., if the target is within the vision range and not blocked by obstacles. The hierarchies (totaling 16 in our experiments) perform 16 learning trials in random order, in each generation.

Environment 1 [Fig. 4(a)] constitutes a very simple task setting, as there are no obstacles in the environment. The prediction is that an optimal solution should be a simple strategy, where the CR needs to execute only one primitive subtask for accomplishing each subgoal and, therefore, no learning is required in the composite subtasks. Such a strategy could be that the CR uses the Battery behavior to capture the battery and Nest behavior for returning to the nest. As there are no obstacles, the need for exploration (Avoid and Wander) and correction of the trajectory (Rotate) seems to be very limited. Avoid can be used for performing a 180° turn at the nest corner and at the battery corner, after battery capturing.

Environment 2 [Fig. 4(b)] was included to evaluate a task setting where the two main tasks (battery capturing and returning to the nest) have different degrees of difficulty. Capturing a battery is relatively simple, as the CR will quickly get visible contact with a battery after it has moved out of the nest area, surrounded by the two walls. Returning to nest constitutes a much more challenging subtask, as the walls surrounding the nest obscure the nest from most positions.

Environment 3 [Fig. 4(c)] is the most challenging task setting, where all subtasks seem to constitute an approximately equal challenge. The prediction is therefore that an optimal hierarchy has to learn composite subtasks with two or more primitive subtasks for accomplishing each subgoal.

V. Experimental Results

Here, we present the hierarchies obtained in the three environments. Although the GP did not find exactly the same hierarchy in all simulations in an environment, the obtained hierarchies in an environment represented almost identical functionality in all simulations.

A. Artificial Evolution

The main results from the experiments are shown in Figs. 5 and 6. Fig. 5 shows, for the three environments, the evolution of the average learning time (the average number of time steps to complete the last half of the learning trials) and the average difference in learning structure (the hierarchies ranked 2–8 are compared with the fittest hierarchy), for the best half of the population (eight individuals).

Fig. 6(a), (c), and (e) shows the hierarchies obtained by the GP in the three environments. Fig. 6(b), (d), and (f) gives an illustration of the behaviors used by the CR for the hierarchies. A typical trajectory used by the best individual in the last generation is shown in the figures. The different types of markers represent the different primitive subtasks executed by the CR in each time step.

In environment 1, the decrease in learning time is correlated with the decrease in structural difference [Fig. 5(a) and (b)]. Both curves reach a stable low level after about 20 generations, where the average trial time is approximately 230 time steps and the average structural difference has a value below 1, suggesting that the best hierarchies differ only for one primitive subtask. The obtained hierarchical structure is very simple, using only two composite subtasks [Fig. 6(a)]. In fact, as predicted earlier, the CR executes only one prelearned primitive subtask for accomplishing each subgoal. While searching for the battery and also while returning the battery to the nest, the CR can select both Avoid and Nest. However, behavioral analysis, as illustrated in Fig. 6(b), verifies that the obtained hierarchy simply combines pretrained primitive subtasks without any learning in the middle nodes. This solution differs in one aspect compared with the naive strategy suggested in the description of the environments (Section IV-C). While searching for the battery, the CR always approaches the nest if it is visible, as seen for the first 7 time steps in Fig. 6(b). Thereby, the CR uses the available vision information received from the nest to navigate quickly to the nest and uses the corner to perform a 180° turn by executing the Avoid behavior.

We consider this result to give strong support for our evolutionary approach, since even for this very simple task setting, the GP is able to find a solution (by exploring the constraints of the task and environment) that is very easily overlooked by a human designer.

The evolutionary process in environment 2 [Fig. 5(c) and (d)] is different from the other two environments. There is only a small improvement in learning time over generations, from approximately 350 time steps to 300 time steps, and the variance remains very large through the whole process. This suggests that there is a large variability in the performance of a single hierarchy between trials. In environment 2, the obtained hierarchy [Fig. 6(c)] has, as predicted, different types of structures for the two main tasks, capturing the battery and returning to the nest. For capturing the battery, the strategy is completely decided by the hierarchy: when the battery is not visible the CR executes the Avoid behavior and after the battery becomes visible the CR executes the Battery behavior. That the GP search does not include the Wander behavior when searching for a battery is probably explained by the fact that Avoid does not have a straightforward action and thereby the CR will get visible contact with a battery as soon as it passes the two walls that surround the nest, as seen in Fig. 6(d).

An interesting type of behavior is displayed when the CR is returning to the visible nest, i.e., the last part of the trajectory. The CR executes the Rotate behavior fairly often, which maybe seems surprising as this behavior does not move the CR forward at all. The explanation is that the CR can only get visual information from the center point of the nest and has to keep a very precise trajectory towards the goal. Otherwise, it will lose sight of the nest or get trapped at the wall. However, since the CR keeps doing this after it has passed the wall, i.e., the very last part of the trajectory, suboptimal learning is suggested.

The obtained hierarchy in environment 3 displays, predictably, the most complex structure [Fig. 6(e)]. The CR executes two primitive subtasks for accomplishing each sub-

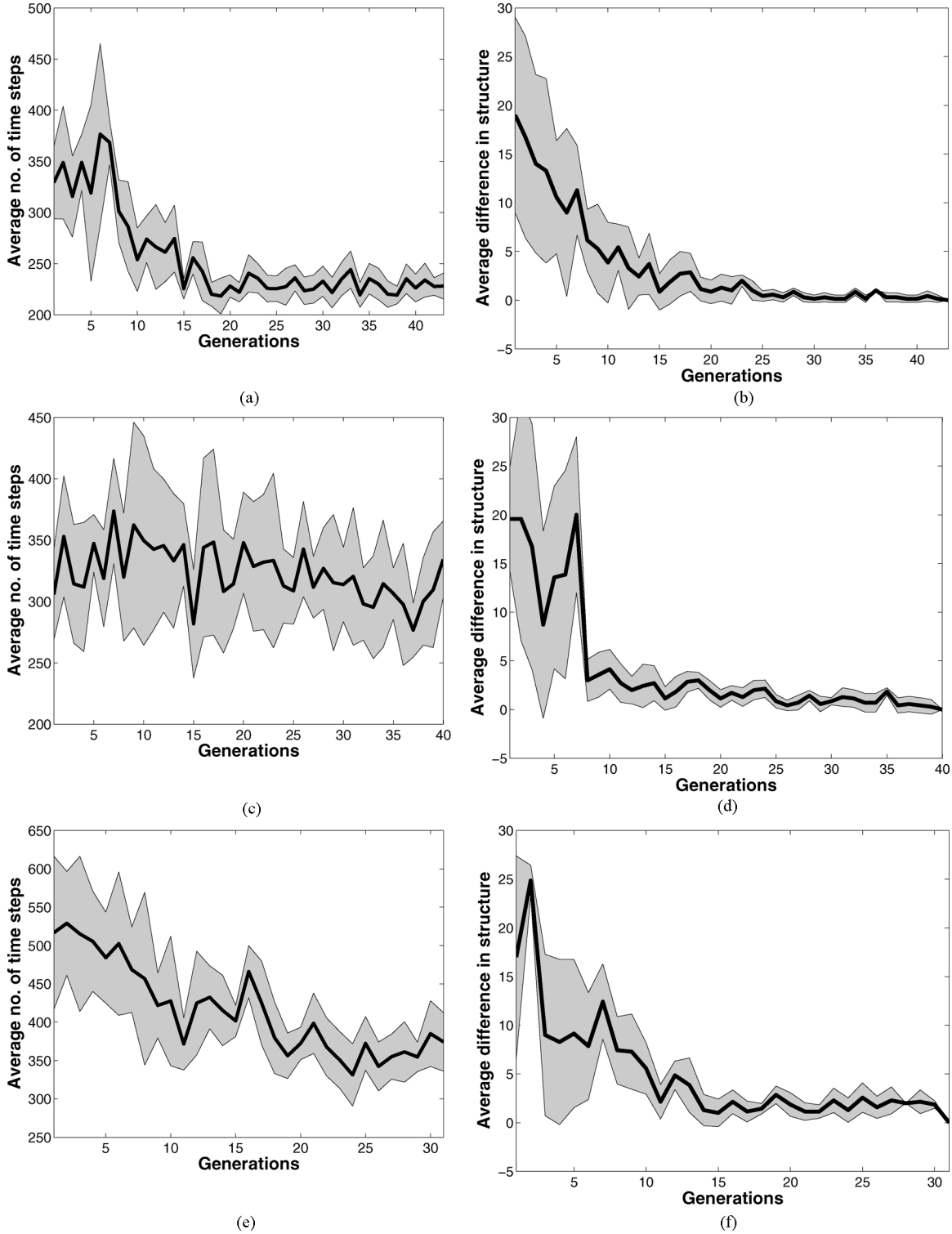

Fig. 5. Panel (a), (c), and (e) show the average learning time for the last half (8) trials, with standard deviation for the eight fittest hierarchies in each generation, in the three environments. Panel (b), (d), and (f) shows average difference in learning structure, with standard deviation for the eight fittest hierarchies in each generation. For each generation, the hierarchies ranked 2–8 have been compared with the fittest hierarchy. Note that the termination criterion for GP is that the eight fittest individuals have identical learning structures, which means that the difference has to be 0 for the last generation. (a) Average learning time in environment 1. (b) Average difference in structure in environment 1. (c) Average learning time in environment 2. (d) Average difference in structure in environment 2. (e) Average learning time in environment 3. (f) Average difference in structure in environment 3.

goal, except for returning to the visible nest [Fig. 6(f)]. While searching, the CR executes the Wander behavior in more open areas and the Avoid behavior to prevent it from getting trapped at walls and corners. When the CR approaches the battery it uses the Rotate behavior to correct the path at three occasions, to prevent losing sight of the battery in similar fashion as in environment 2.

The curves for environment 3 [Fig. 5(e) and (f)] follow the same pattern as for environment 1, with the important difference that the convergence of the structural difference precedes the

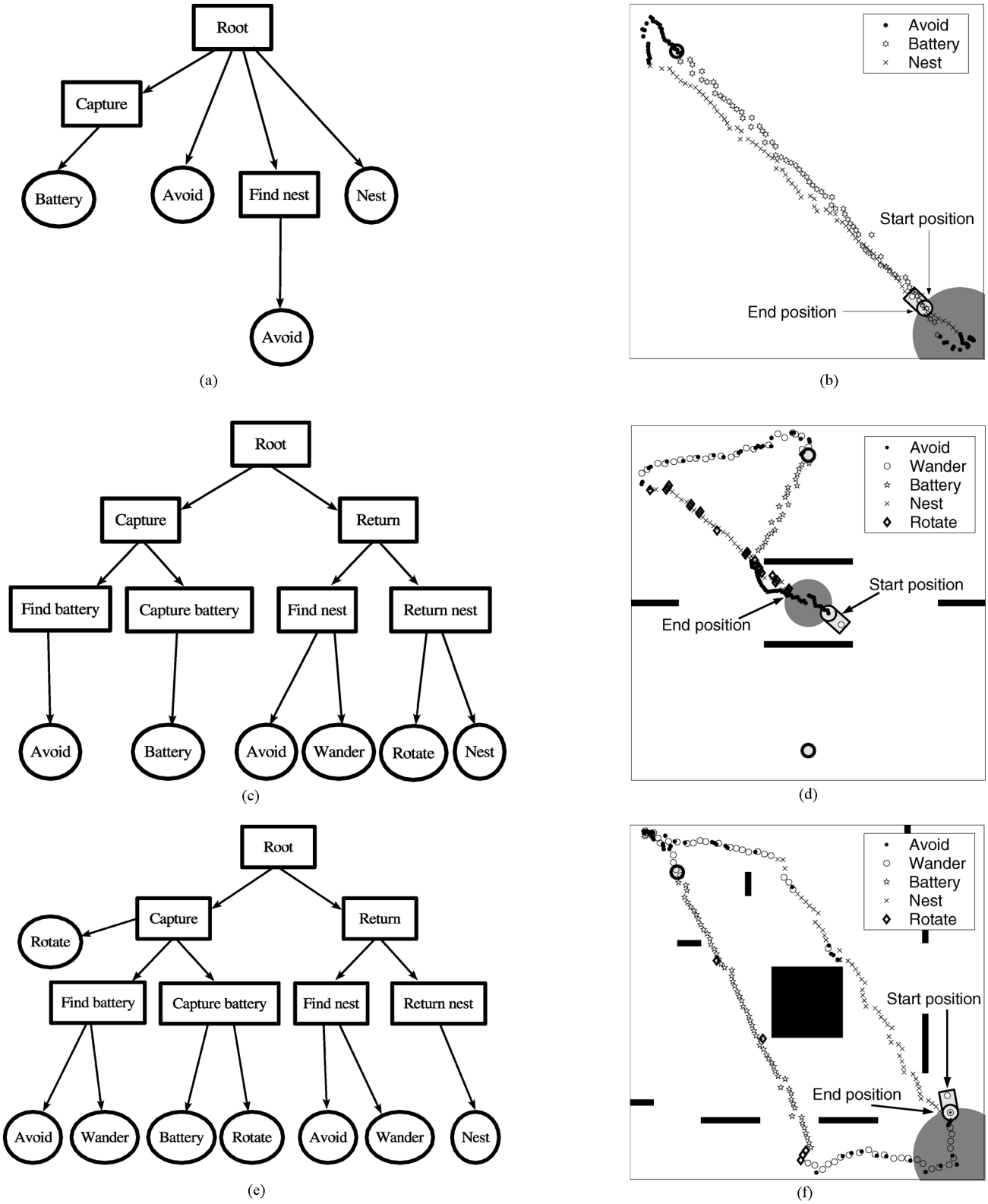

Fig. 6. The obtained hierarchies [(a), (c), and (e)] and typical trajectories [(b), (d), and (f)] used by the best individuals in the last generation in the three environments. The markers indicate which primitive subtask the CR used in each time step. The position of the markers is equal to the position of the center of the head circle of the simulated robot. (a) Obtained hierarchy in environment 1. (b) Typical trajectory used by the best individual in environment 1. (c) Obtained hierarchy in environment 2. (d) Typical trajectory used by the best individual in environment 2. (e) Obtained hierarchy in environment 3. (f) Typical trajectory used by the best individual in environment 3.

convergence of the learning time with approximately ten generations. This suggests a Lamarckian learning effect because the hierarchies continue to improve their behaviors, i.e., the policies of the composite subtasks, over generations, even when the learning structure is almost fixed. The benefit of Lamarckianism in environment 3 was confirmed in an additional experiment, in

which the eight best hierarchies from the last generation were compared with eight hierarchies with identical structure, but with the learning reset to scratch in all composite subtasks, i.e., the function approximation of $\tilde{C}$, C, and Q_{prim}, and the inverse temperature β were set to their initial values. The experiment was performed in the same manner as the evaluation part of the evolution. Each hierarchy performed 16 learning trials and the fitness was calculated as the average number of time steps for the last 8 trials. This was repeated 10 times for each hierarchy, giving a total of 80 evaluations each for Lamarckian learning and learning from scratch. The performances were compared by a student *t* test and the difference was significant ($p = 0.0265$). Similar experiments were also performed for the hierarchies obtained in environment 1 and environment 2, but there was no significant difference in performance between Lamarckian learning and learning from scratch. This is not surprising because large parts of these behaviors are completely decided by the hierarchies, i.e., the CR can only execute one primitive subtask. Therefore, the necessity for learning is limited (environment 2) or very limited (environment 1).

Fig. 7 shows the development of composite and primitive subtasks for the evolution in environment 1. The figures show how the number of composite [Fig. 7(a)] and primitive [Fig. 7(b)] subtasks included in the best half of the hierarchies changes over generations. We have chosen to only present the results for one environment because the process was very similar for all our simulations in all environments. As indicated earlier by the difference in structure, the evolution very quickly, i.e., 15–20 generations, finds the preferable composite subtasks, as seen in Fig. 7(a). During the first half of the evolution, the majority of the primitive subtasks are also decided, except for one or two subtasks. Typically, the evolution takes a longer time deciding whether to include or exclude the Avoid and/or Wander behaviors while exploring, as seen in Fig. 7(b).

In general, the evolutionary process tries to find a minimal strategy for the task. With a minimal strategy, we do not mean a hierarchy that contains as few subtasks as possible. Instead, a minimal strategy minimizes the number of primitive subtasks that can be executed to achieve the subtasks' goals, and the composite subtasks are used for limiting the scope of the different primitive subtasks. Especially, the evolution tries to find strategies for which only one primitive subtask is required to solve a subgoal, as seen for the overall task in environment 1 [Fig. 6(a)], finding and capturing of a battery in environment 2 [Fig. 6(c)], and returning to the visible nest in environment 3 [Fig. 6(e)]. A minimal strategy minimizes the need for relearning of the behaviors, and thereby promotes more stable behaviors with higher fitness.

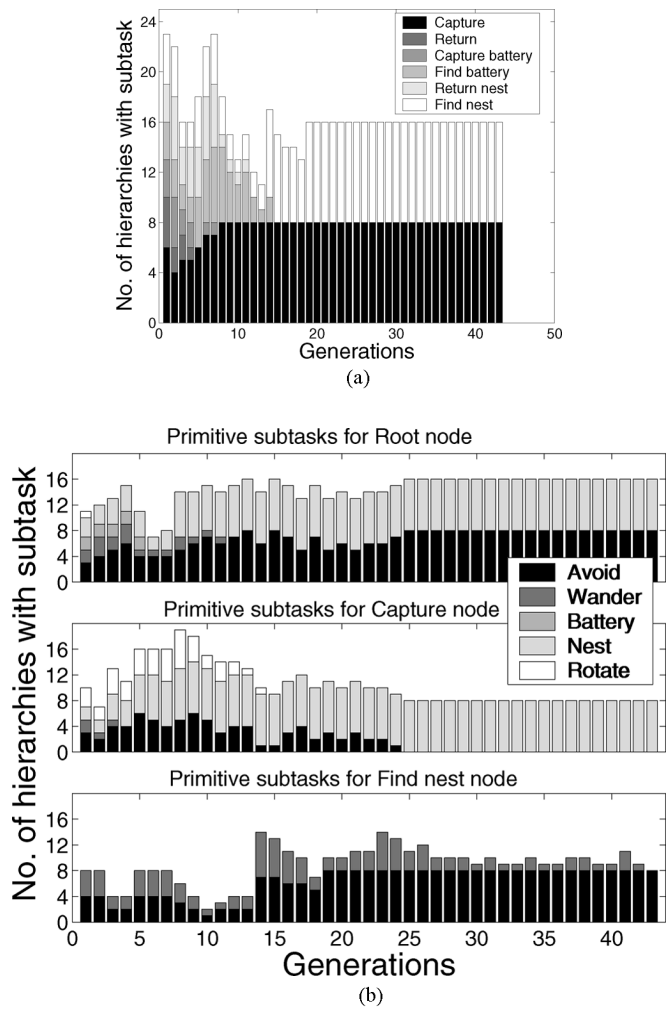

Fig. 7. The bar graphs show the development of composite and primitive subtasks in environment 1. The top figure shows the number of composite subtasks included in the best half the hierarchies (8) in each generation. The bottom figure shows the number of primitive subtasks included in best half of the hierarchies, for each composite subtask. The figure includes only the relevant composite subtasks, i.e., composite subtasks that are included in the final obtained solution. (a) The development of composite subtasks over generations in environment 1. (b) The development of primitive subtasks for the relevant composite subtasks in environment 1.

B. Cross Evaluation

We have performed cross-testing experiments to test how well the evolution adapts the learning structures for the different environments, and the ability to perform the task in environments different from the training environment.

We compared the eight best hierarchies from the last generation in each environment and tested their performances in the three environments. The experiments were performed in the same manner as the evaluation part of the evolution. Each hierarchy performed 16 learning trials and the fitness was calculated as the average number of time steps for the last 8 trials. This was repeated 10 times for each environment, giving a total of 80 evaluation tests for each type of hierarchy and environment. Fig. 8 shows the average of the 80 tests for each tested hierarchy and environment, as bar graphs including standard deviation. An asterisk indicates that the difference in average performance between the hierarchy trained in the environment and another hierarchy was highly significant ($p = 0.001$), according to a student *t* test. The trajectories in Fig. 9 give an illustration of the performance of the hierarchies, when tested in an environment different from the training environment.

The results show that the GP is, indeed, able to obtain hierarchies that are especially adapted to the given task and environment. In each environment, the hierarchy obtained in that environment has significantly higher performance compared with

Fig. 8. Cross-testing results in the three environments. The performances of the eight best hierarchies from the last generation in each environment were tested in environment 3. The figures show the average learning time with standard deviation for 80 evaluation tests for each type of hierarchy and environment. An asterisk indicates that the average difference in performance between the hierarchy trained in the environment and another hierarchy was highly significant ($p = 0.001$). (a) Cross-testing results in environment 1. (b) Cross-testing results in environment 2. (c) Cross-testing results in environment 3.

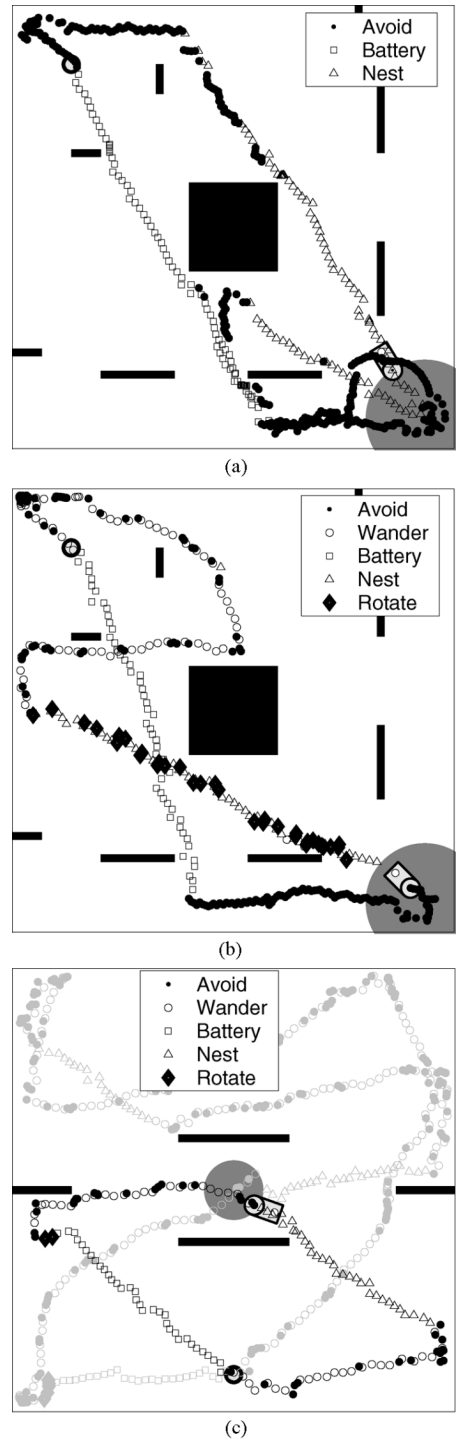

Fig. 9. Illustration of the behaviors of the obtained hierarchies when tested in an environment that is different from the training environment. (a) Typical trajectory used by the best hierarchy obtained for environment 1, when tested in environment 3. (b) Typical trajectory used by the best hierarchy obtained for environment 2, when tested in environment 3. (c) Good (black) and bad (gray) trajectories used by the best hierarchy obtained for environment 3, when tested in environment 2.

the other two hierarchies. The only exception is that in environment 2, the solution obtained in environment 3 performs slightly better on average than the obtained solution [the third bar in Fig. 8(b)]. However, this is not a robust solution because the variance is more than twice as large compared with the hierarchy trained in environment 2. The large difference in performance seems to have two main explanations that are visualized in Fig. 9(c), where the trajectories for a good (black) and bad (gray) learning trial are shown. First, the hierarchy trained in environment 2 [Fig. 6(c)] uses only Avoid, which lacks a straight

forward action, to move out of the nest. This ensures that the CR will get visual contact with the battery very quickly. In contrast, the hierarchy [Fig. 6(e)] trained in environment 3 use a mix of Avoid and Wander. The CR is, therefore, moving in a relatively straight trajectory out of the nest and has to turn around by encountering a wall (the black trajectory) or a corner (the gray trajectory). Second, the hierarchy trained in environment 3 lacks the Rotate behavior when the CR is returning to the nest. The CR, therefore, cannot perform precise correction of the trajectory towards the nest, and will easily lose the visual contact, as seen for the gray trajectory in the upper left part of the figure.

Fig. 9(a) and (b) illustrates well why the hierarchies trained in environment 1 and 2 are suboptimal when tested in environment 3. Although, the trajectories are not so different from the trajectories executed by hierarchy trained in the environment [Fig. 6(f)], the suboptimal hierarchies execute on average 75–100 more primitive subtasks to complete a trial. Four clearly displayed suboptimal features are that: 1) both hierarchies use the slow Avoid behavior to find the battery; 2) the hierarchy trained in environment 1 uses only the slow Avoid behavior to find the nest; 3) this solution also returns to the nest when searching for the battery; and 4) the hierarchy trained for environment 2 uses the Rotate behavior frequently when returning to the nest, as this action does not move the CR forward, it prolongs the trial.

The results show that it is difficult to construct a general solution, i.e., a hierarchy that performs really well in all environmental settings. However, the Lamarckian approach can offer assistance in achieving this goal, providing that the hierarchy is developed in the most challenging task setting.

If the task is very simple (environment 1) or parts of the task are very simple (the battery capturing part of environment 2), the evolution will try to find a strategy that uses only one primitive subtask for completing the simple parts of the task. This solution is, of course, very effective for the given simple task, but generalizes very badly to other types of environments, as illustrated by the performance in environments 2 and 3, for the hierarchy trained in environment 1 [the first bar in Fig. 8(b) and (c)]. Also, as the hierarchy itself is deciding the behavior, or part of, there is no possibility or minimal possibility for Lamarckian optimization of the behavior over generations.

In contrast, hierarchies that are developed in a difficult task setting have to use a mix of different behaviors to accomplish each subgoal. Thereby, the hierarchies can benefit from the Lamarckian learning effect and keep on optimizing the learning in the different composite subtasks through the evolutionary process. When the agent is placed in a new environment, it can compensate for a suboptimal learning structure with more optimized policies for the composite subtasks. This is illustrated for the hierarchy obtained in environment 3 [the third bar in the graphs in Fig. 8], the most challenging environmental setting in our experiments. It has the best performance on average in both environments 2 and 3, and second best in environment 1.

A natural objection to our conclusions is that a minimal strategy learns the task faster, but it is not certain that it will outperform a hierarchy containing more primitive subtasks for accomplishing each subgoal in the long run. A hierarchy containing many primitive subtasks will naturally require

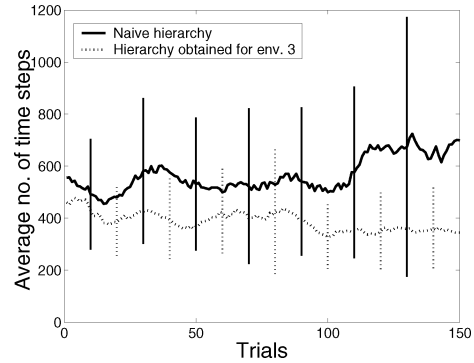

Fig. 10. Performance for longer evaluation period in environment 3. The figure shows the average performance for 150 learning trials for the eight fittest hierarchies obtained for environment 3 (dotted line), and eight naives hierarchies with the structure used to pretrain the primitive subtasks (full line). The vertical bars indicate the standard deviation.

a longer time for the learning in the composite subtasks to converge. The evaluation time (16 learning trials) may have been too short for complex learning structures, and our method could therefore prevent more complex hierarchies from being obtained. To test this hypothesis, we compared, in environment 3, the average performance for the eight fittest hierarchies obtained in environment 3 with eight general hierarchies used to pretrain the primitive subtasks [Fig. 1]. In the experiment, we extended the evaluation time from 16 to 150 learning trials and the result is shown in Fig. 10.

The result shows convincingly that our method, i.e., obtaining minimal strategies by GP, not only accelerates the learning process, but also provides solutions that have good long-time performance with low variance. The results also illustrate the difficulty to perform the task decomposition in advance without considering the current environmental setting.

The dotted curve for the hierarchy obtained in environment 3 shows a relatively rapid increase in performance from approximately 450 to 380 time steps around trial 16. The performance is thereafter relatively stable until trial 100. At that point, the performance is increased again to 350 time steps, and remains very stable for the rest of the learning time. The increase in performance around trial 100 is explained by our exploration/exploitation scheme, i.e., exponential increase of the inverse temperature β (see the end of Section III-A). At this point in time, the learning in the composite subtasks is considered converged and the action selection becomes almost greedy, with the exception that we limit the maximum probability for selecting a child subtask (0.98 in our experiments). The CR's behavior will be almost deterministic from this point and a hierarchy with proper structure and well learned policies in the composite subtasks will achieve better and more stable performance.

Generally, the issue of assigning meta-parameters, such as the inverse temperature β is an open question in RL, see, e.g., Doya's work [3]. In contrast to discrete grid-world tasks, which can be completely described as an MDP, more realistic robotic tasks are more accurately described as a partially observable MDP (*POMDP*). POMDP problems often lack theoretically optimal solutions and are, in general, very difficult from a RL point of view. Although the effect of meta-parameters in the RL framework is outside the scope of this study, we can conclude

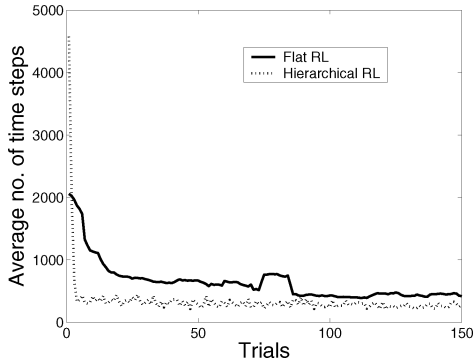

Fig. 11. Comparison of the performance, when learning from scratch, between flat RL, implemented by Q-learning, and the obtained hierarchical MAXQ solution for environment 3.

for our POMDP task, that the exploration/exploitation scheme plays an important role for the performance of the hierarchies. It is especially important that the composite subtasks have learned a good policy at the point of time when the action selection is changed from being more stochastic to almost deterministic.

The curve for the naive hierarchy shows an almost linear increase in performance from 550 to 455 time steps during the first 16 trials. The performance is then decreased, probably due to less stochasticity in the action selection, and reaches a relatively stable level around 520 time steps between trials 50 to 100. Around trial 100, when the behavior is almost deterministic, the performance drastically decreases and reaches a stable high-level of about 650 time steps with very large variance. As described in the paragraph above, the explanation is that because of the POMDP nature of the task, it is very difficult for the composite subtasks to obtain optimal policies. The naive hierarchy, therefore, needs a lot of exploratory random actions to accomplish a fairly good performance, i.e., after about 16 trials. When the behavior becomes almost deterministic, the performance decreases significantly because of suboptimal learning in the composite subtasks.

Another possible objection to our results is that our task is too simple and a hierarchical structure is not needed. We therefore compared the learning performance between flat RL implemented by Q-learning, to the obtained hierarchical solution in environment 3. To be able to compare the learning speed, we started the learning from scratch in both the primitive subtasks, which were given *a priori* in the earlier experiments, and the composite subtasks. The result for ten simulations is shown in Fig. 11 as the average time per trial. The figure clearly shows that a good hierarchical task decomposition can accelerate the learning speed.

VI. Discussion

This paper has proposed a method for combining GP and the MAXQ framework for automatic development of hierarchical learning structures. In the proposed method, the MAXQ method learns the policy based on the hierarchies obtained by the GP, while the GP explores the appropriate hierarchies using the result of the MAXQ method. In the MAXQ framework, each subtask learns the policy without reference to the parent task's policy. Learning structures can, therefore, be preserved in a Lamarckian fashion, when applying the GP crossover operation, which was used for mixing the competing task hierarchies.

The method was evaluated for a robotic foraging task in three different environmental settings in simulation. In earlier hierarchical RL studies, the effect of different environmental conditions on the optimal task decomposition has been mostly neglected. Our results show that the environment plays an important role for good task decomposition. For example, the obtained hierarchy for the simple environment without obstacles represents a simple and specialized strategy, while the obtained hierarchy for the most challenging environment represents a more general solution that also performs well in other environments than the training environment.

The main conclusion of the experiments is that the GP can find a minimal strategy to the problem at hand, i.e., a hierarchy that uses as few primitive subtasks as possible for each type of situation the agent can experience while performing the task. Especially, the evolutionary search tries to find strategies for which only one primitive subtask is required to solve each subgoal. In this case, the behavior is completely decided by the hierarchy itself and no learning is required.

The experimental results also show that the Lamarckian approach, i.e., continuous learning in the subtasks over generations, can have an advantage in our framework, given that the environmental setting is difficult enough to require that the agent has to use a mix of primitive subtasks to accomplish the subgoals. The benefit of Lamarckianism was displayed for the most difficult environmental setting in our experiment, for which there was significant difference in average performance between the hierarchies obtained by Lamarckian evolution and hierarchies with identical learning structure, but the learning reset to scratch.

Our main goal is to perform hardware experiments on the CR robot. A robotic platform developed for the study of artificial agents under the same fundamental constraints as real biological agents. The real test of the proposed method will, therefore, be to perform the experiments in the hardware setting. Our recent embodied evolution study [6] showed that it is possible to transfer primitive behaviors learned by RL in simulation directly to the hardware with reasonable performance. We therefore believe it is possible to move the implementation from the simulation to the hardware setting in a relatively straightforward manner. The main obstacles are related to computational capacity of the CR robot, such as the high-computational cost for updating and the large memory requirement for storing the RBF approximation of the value functions of the subtasks.

Our results show the need for hierarchical RL methods to take both the task and environment in consideration when performing task decomposition. We see this as a first promising step toward a truly automatic hierarchical RL method. Besides the task decomposition, the MAXQ framework still requires a lot of human involvement, such as deciding the number of subtasks and the important design of each of the provided subtasks. In the future, we plan to develop a more integrated method for combining GP and the MAXQ framework. The evolutionary search should not only be used to optimize the hierarchy, but also for construction of, e.g., the state space abstractions in the subtasks. In the current method, we only use the GP crossover operator to construct new types of hierarchies, and we therefore need to

construct genetic operators for adding and deleting subtasks in a structural manner.

One possible inspiration for future work comes from the related field of evolution neural network (NN) topologies. Stanley and Miikkuklainen have shown that neuroevolution, evolution of neural networks by genetic algorithms, can be used for fast and efficient adaptation of the NN topology. They evaluated their NEAT system [17] for the most difficult version of the pole balancing problem, the standard benchmark test for RL algorithms. The NEAT system, which utilizes incrementally growing NNs and a principled method for genetic operations, was significantly faster and outperformed the best fixed-topology methods.

APPENDIX
RBF FUNCTION APPROXIMATION

A linear normalized Gaussian RBF network is generally defined for approximator vector $\boldsymbol{\theta}$ with features $\boldsymbol{\phi}_s$ as

$$V(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}_s \tag{18}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\big(r + \gamma V(s') - V(s)\big)\boldsymbol{\phi}_s \tag{19}$$

$$\phi_s(i) = \frac{e^{-\|s-c_i\|^2/2\sigma_i^2}}{\sum_{j=1}^{n} e^{-\|s-c_j\|^2/2\sigma_j^2}} \tag{20}$$

where c_i is the center of feature i and σ_i^2 is the variance of feature i.

The approximation of the completion functions, $\tilde{C}$ and C, and the approximation of the action-value function for the primitive subtasks Q_{prim}, are represented with the parameter vectors $\boldsymbol{\theta}_{i,a}^{\tilde{C}}$, $\boldsymbol{\theta}_{i,a}^{C}, \boldsymbol{\theta}_{i,a}^{Q_{\text{prim}}}$, with feature vectors $\boldsymbol{\phi}_{i,s}^{\tilde{C}}, \boldsymbol{\phi}_{i,s}^{C}, \boldsymbol{\phi}_{i,s}^{Q_{\text{prim}}}$. The learning rules in (12)–(14) are modified to

$$\boldsymbol{\theta}_{i,a}^{\tilde{C}} \leftarrow \boldsymbol{\theta}_{i,a}^{\tilde{C}} + \alpha(i)\Big\{\gamma^N\big[\tilde{R}_i(s') + \tilde{C}(i,s',a^*) + V(a^*,s)\big] - \tilde{C}(i,s,a)\Big\}\boldsymbol{\phi}_{i,s}^{\tilde{C}} \tag{21}$$

$$\boldsymbol{\theta}_{i,a}^{C} \leftarrow \boldsymbol{\theta}_{i,a}^{C} + \alpha(i)\Big\{\gamma^N\big[C(i,s',a^*) + V(a^*,s')\big] - C(i,s,a)\Big\}\boldsymbol{\phi}_{i,s}^{C} \tag{22}$$

$$\boldsymbol{\theta}_{i,a}^{Q_{\text{prim}}} \leftarrow \boldsymbol{\theta}_{i,a}^{Q_{\text{prim}}} + \alpha(i)\left[r - Q_{\text{prim}}(i,s,a)\right]\boldsymbol{\phi}_{i,s}^{Q_{\text{prim}}} \tag{23}$$

where

$$\tilde{C}(i,s,a) = \boldsymbol{\theta}_{i,a}^{\tilde{C}^T}\boldsymbol{\phi}_{i,s}^{\tilde{C}} \tag{24}$$

$$C(i,s,a) = \boldsymbol{\theta}_{i,a}^{C^T}\boldsymbol{\phi}_{i,s}^{C} \tag{25}$$

$$Q_{\text{prim}}(i,s,a) = \boldsymbol{\theta}_{i,a}^{Q_{\text{prim}}^T}\boldsymbol{\phi}_{i,s}^{Q_{\text{prim}}}. \tag{26}$$

REFERENCES

[1] D. H. Ackley and M. L. Littman, "Interaction between learning and evolution," in *Proc. 2nd Conf. Artif. Life*, C. G. Langton *et al.*, Ed. *et al.*, 1991, p. 152.

[2] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, 2000.

[3] K. Doya, "Metalearning and neuromodulation," *Neural Netw.*, vol. 15, no. 4, 2002.

[4] K. Doya and E. Uchibe, "The cyber rodent project: Exploration of adaptive mechanisms for self-preservation and self-reproduction," *Adaptive Behav.*, vol. 13, no. 2, pp. 149–160, 2005.

[5] K. L. Downing, "Adaptive genetic programs via reinforcement learning," in *Proc. Genetic Evol. Comput. Conf.*, 2001, pp. 19–26.

[6] S. Elfwing, U. Uchibe, K. Doya, and H. I. Christensen, "Biologically inspired embodied evolution of survival," in *Proc. IEEE Congr. Evol. Comput.*, 2005, pp. 2210–2216.

[7] A. Eriksson, G. Capi, and K. Doya, "Evolution of meta-parameters in reinforcement learning algorithm," in *Proc. IEEE Conf. Intell. Robots Syst.*, 2003, pp. 412–417.

[8] G. E. Hinton and S. J. Nowlan, "How learning can guide evolution," *Complex Syst.*, vol. 1, pp. 495–502, 1987.

[9] C. Houck, J. Joines, M. Kay, and J. Wilson, "Empirical investigation of the benefits of partial Lamarckianism," *Evol. Comput.*, vol. 5, no. 1, pp. 31–60, 1997.

[10] H. Iba, "Multi-agent reinforcement learning with genetic programming," in *Proc. 3rd Annu. Conf. Genetic Programming* , J. R. Koza, Ed. *et al.*, Madison, WI, 1998, pp. 167–172.

[11] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[12] M. Minsky, *The Society of Mind*. New York: Simon and Schuster, 1986.

[13] D. J. Montana, "Strongly typed genetic programming," *Evol. Comput.*, vol. 3, no. 2, pp. 199–230, 1995.

[14] R. Parr and S. Russel, "Reinforcement learning with hierarchies of machines," *Advances in Neural Inf. Process. Syst.*, vol. 10, pp. 1043–1049, 1998.

[15] G. A. Rummery and M. Niranjan, Online Q-Learning using connectionist systems, Cambridge Univ., Cambridge, U.K., Tech. Rep. CUED/FINFENG /TR 166, 1994.

[16] T. Sasaki and M. Tokoro, "Adaptation toward changing environments: Why Darwinian nature?," in *Proc. 4th Eur. Conf. Artif. Life*, P. Husbands and I. Harvey, Eds., 1997, pp. 145–153.

[17] K. O. Stanley and R. Miikkulainen, "Efficient reinforcement learning through evolving neural network topologies," in *Proc. Genetic Evol. Comput. Conf.*, 2002, pp. 569–577.

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

[19] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, pp. 181–211, 1999.

[20] P. Turney, L. D. Whitley, and R. W. Anderson, "Introduction to the special issue: Evolution, learning, and instinct: 100 years of the Baldwin effect," *Evol. Comput.*, vol. 4, no. 3, pp. iv–viii, 1997.

[21] C. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, 1992.

Stefan Elfwing received the M.Sc. degree from the Royal Institute of Technology, Stockholm, Sweden, in 2003. He is currently working towards the Ph.D. degree in computer science at the Centre for Autonomous Systems, Royal Institute of Technology, and the Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology, Japan.

He does research on robotics with focus on reinforcement learning and embodied evolutionary systems.

Eiji Uchibe received the Ph.D. degree in mechanical engineering from Osaka University, Osaka, Japan, in 1999.

He was a Research Associate of the Japan Society for the Promotion of Science, in Research for the Future Program titled Cooperative Distributed Vision for Dynamic Three-Dimensional Scene Understanding. Then, he joined ATR as a Researcher in 2001. Since 2004, he has been a Researcher at the Neural Computation Unit, Initial Research Project, Okinawa Institute of Science and Technology. His research interests are in learning robots, reinforcement learning, evolutionary computation, and computational neuroscience, and their applications.

Kenji Doya received the B.S., M.S., and Ph.D. degrees from the University of Tokyo, Tokyo, Japan, in 1984, 1986, and 1991, respectively.

He became a Research Associate at the University of Tokyo in 1986, the University of California at San Diego in 1991, and Salk Institute in 1993. He joined ATR in 1994 and is currently the Head of Computational Neurobiology Department, ATR Computational Neuroscience Laboratories. In 2004, he was appointed as a principal investigator of Initial Research Project, Okinawa Institute of Science and Technology. He is interested in understanding the functions of basal ganglia and neuromodulators based on the theory of reinforcement learning.

Henrik I. Christensen received the M.Sc. and Ph.D. degrees in electrical engineering from Aalborg University, Aalborg, Denmark, in 1987 and 1989, respectively.

He is a Professor of Computer Science and Director of the Centre for Autonomous Systems, Royal Institute of Technology, Sweden. He has published more than 190 contributions. He does research on robotics, vision, and AI with a main emphasis on system perspectives.

Dr. Christensen serves on the Editorial Board of the *International Journal of Robotics Research* (IJRR), the *International Journal of Pattern Recognition and Artificial Intelligence* (IJPRAI), AIM, and *Autonomous Robotics*. He is also the founding Chairman of the European Robotics Network (EURON) and serves as an IEEE RAS Distinguished Lecturer in Robotics.