Behavior Hierarchy Learning in a Behavior-based System using Reinforcement Learning

Amir massoud Farahmand   Majid Nili Ahmadabadi   Babak Najar Araabi
Control and Intelligent Processing Center of Excellence, Dept. of Elect and Comp. Eng. University of Tehran
School of Cognitive Sciences, IPM, Tehran, Iran
Farahmand@ipm.ir, Mnili@ut.ac.ir, Araabi@ut.ac.ir

Abstract— **Hand-design of an intelligent agent's behaviors and their hierarchy is a very hard task. One of the most important steps toward creating intelligent agents is providing them with capability to learn the required behaviors and their architecture. Architecture learning in a behavior-based agent with Subsumption architecture is considered in this paper. Overall value function is decomposed into easily calculate-able parts in order to learn the behavior hierarchy. Using probabilistic formulations, two different decomposition methods are discussed: storing the estimated value of each behavior in each layer, and storing the ordering of behaviors in the architecture. Using defined decompositions, two appropriate credit assignment methods are designed. Finally, the proposed methods are tested in a multi-robot object-lifting task that results in satisfactory performance.**

Keywords- reinforcement learning, architecture learning, credit assignment, value decomposition, subsumption architecture.

I. Introduction

Our research long-term goal is developing general automatic methods for designing distributed multi-agent systems (which may be used in a robot's mind considering each behaviors as an agent or a team of robots) using reinforcement learning methodology [1]. The main challenges of this problem are learning behaviors of a robot and learning the correct organization of these behaviors. To solve these problems, we must devise a method to solve the credit assignment problem and also be able to make a balance between behavior and architecture learning. Our previous studies reveal that it is not easy to solve credit assignment problem in multi-agent systems unless we assume the task or the team structure [2]. Therefore, we have chosen to work on Subsumption Architecture (SSA), which can give us some credit assignment clues.

SSA has been used in many successful applications and is one of the most well known behavior-based architectures ([3], [4], and [5]). However, designing behaviors and arranging them in order to emerge the desired overall behavior is not a trivial task. There has been some works trying to remedy this problem by adding learning to the architecture (e.g. [6], [7], [8], and [9]) but they have not solved the problem completely and there are many questions remained. Here, we have started working on designing a set of methods that enable a behavior-based agent to learn from its environment. We have divided this task into two different parts: learning architecture of SSA and learning its behaviors. Tangling these two problems, it is possible to build up a complete agent.

Architecture learning means the way behaviors are arranged in the architecture in order to maximize some performance index that is defined by a designer. By behavior learning, we mean learning how a behavior should react to its input. In this paper, we focus on the first problem: architecture learning.

In this paper we present two different methods to learn the architecture of the agent assuming that we have suitable behaviors to put in it. The proposed methods will be presented in Section II. After that in Section III, we will test our ideas in a multi-agent object lifting task and show the effectiveness of our methods. Conclusions will be made in Section IV.

II. Proposed Methods

A. Problem Formulation

Suppose that we have a set of n behaviors B_i, defined as follow

$$\begin{aligned} &B_i : S_i \to A \qquad \mathrm{i} = 1,\ldots,\mathrm{n} \\ &S_i \subset S \end{aligned} \tag{1}$$

where S_i is a subset of the state space observable by B_i and A is its outputted action. Each S_i may have some intersection with each other. State space S can be regarded as a space containing sensory information and also internal states of the agent. If $s_i \in S_i$, the behavior will activate and outputs $a_i = B_i(s_i)$, else it will do nothing. It is apparent that in this formulation, all behaviors have a similar type of output but their inputs are not necessarily the same. For instance, consider a set of behaviors that must control moving direction of a robot. Some behaviors process sonar information, some use vision-based data and some other use tactile sensors, but all of them would suggest a common direction.

Now we want to arrange these behaviors in an architecture in order to maximize the performance of the system. There are many different possible arrangements for behaviors in architecture, but here we will study one special case that we call it *purely parallel Subsumption Architecture* (PPSSA) (see Fig. 1). In a PPSSA, all of the behaviors are parallel and a higher behavior has a priority to suppress the lower one. What we want to do is finding a m-dimensional sequence of behaviors out of n possible ones that maximizes the performance of the agent. More formally, we define T as a sequence of behaviors in the architecture as

$$T = [ind(1)\ ind(2)\ ...\ ind(m)]^{\mathrm{T}} \quad m \le n \quad (2)$$

$$ind(i): j \quad \text{(that indicates } \mathrm{B_j} \text{ is in i}^{\text{th}} \text{ layer)} \quad (3)$$

in which each layer is defined as those places that a behavior can be placed. Assigning $m = n$ means that we want to use all of our behaviors in the structure, but $m < n$ means that we want to use only a fraction of our behavior repertoire. It is plausible, as in a design problem we may have more than necessary behaviors in our toolbox. As a notational convenience, we assume that $T(1)$ is the highest behavior in the architecture and $T(m)$ is the lowest one.

Until now, we have used performance many times without indicating what it means. We use performance here as an objective optimality criterion that should be maximized. Regarding reinforcement learning notion, this criterion is a function of received rewards and punishments through time. Having r_t as a reinforcement signal received at time t, the value of the total system (represented by T) is

$$V_T = E\left[r_t \middle| \text{the agent with structure T}\right]. \quad (4)$$

What must be done is maximizing this value by finding an appropriate T which is consisted of B_i s

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

Now we should find a valid sequence T which satisfies (5). To devise a method for doing so, we can divide the problem to the following sub-problems:

- *Representation:* How should the agent represent knowledge gathered by reinforcement signal?

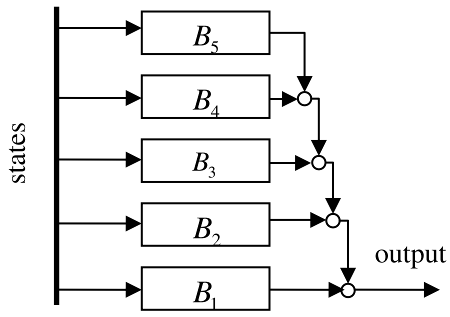

Figure 1. A typical structure of purely parallel Subsumption architecture

- *Hierarchical Credit assignment:* How should the agent assign credit to different behaviors in its architecture?
- *Learning:* How should the agent update its knowledge when it receives reinforcement signals?

First, the agent must have a kind of data structure that stores necessary information gathered during learning. A very simple method is storing the estimated expectation (empirical mean) of reinforcement signal of every possible sequence. Suppose that the behavior repertoire has n behaviors and we want to make an architecture with m behaviors. So we need n^m different storage places to store estimations (in this representation, behaviors may be replicated). It is evident that this is a very big representation space that increases learning time considerably. In addition, there is no relation between two different sequences, so it is not possible to use information gathered by a particular architecture to estimate the value of the others – although they may be similar. Beside that, it is desirable to have representation that is complete and be able to express every possible sequence ordering. Summarizing, a representation must be complete, has small representation space and use good amount of information gathered during learning.

Suppose having a good capable representation with previously mentioned features, it is possible to make a sequence $T_1 = [B_1\ B_2\ ...\ B_m]^T$ (assuming that there is some knowledge stored in it which let us make an architecture with it) and using it, defining a SSA and run the agent with it. While being in the environment, the agent receives a reinforcement signal r_t. Indicating which behavioral organization is responsible for it is the problem of *hierarchical credit assignment.*

The way we should update that representative data structure knowing how the architecture should be rewarded (or punished) is our last problem.

We propose two different representation structures and their relevant credit assignment and learning methods.

B. Zero Order Representation

In this representation, we store the expected value of each behavior in each layer. Having the previous assumption, we can write

$$V_{ZO}(i,j) = V_{ij} = E\left[r_t \left| \begin{matrix} B_j \text{ is controlling} \\ \text{behavior in the i}^{\text{th}}\text{layer} \end{matrix} \right. \right]. \quad (6)$$

What we want to do so is decomposing V_T into some parts that can be stored and updated effectively. This is the main idea of our representations. To do so using this representation and assuming that at least one layer becomes active, we can write

$$\begin{aligned} V_T &= E[r_t] \\ &= E[r_t \mid L_1 \text{ is cont.}] \cdot P(L_1 \text{ is cont.}) \\ &+ E[r_t \mid L_2 \text{ is cont.}] \cdot P(L_2 \text{ is cont.}) + ... \\ &+ E[r_t \mid L_m \text{ is cont.}] \cdot P(L_m \text{ is cont.}) \end{aligned} \quad (7)$$

in which $E[r_t \mid L_i \text{ is cont.}]$ is expected reward of the system when the i^{th} layer takes control and $P(L_i \text{ is cont.})$ is its probability of being controlling layer. We can write the value part of the right-hand side terms as

$$\begin{aligned} &E[r_t \mid L_i \text{ is controlling}] = \\ &\sum_{j=1}^{n} P\{B_j \mid L_i\} E\left[r_t \left| \begin{matrix} B_j \text{ is controlling} \\ \text{behavior in L}_\text{i} \end{matrix} \right. \right]. \end{aligned} \quad (8)$$

In order to find an optimum structure, we should select a one that satisfies (5), which means

$$T^* = \arg\max_T \sum_{i=1}^{m} \begin{matrix} P\{B_{ind(T(i))} \mid L_i\} V_{i,ind(T(i))} \\ \times P(L_i \text{ is cont.}) \end{matrix}. \quad (9)$$

In the case of greedy-like architecture selection that we do not select any other behavior in a layer except the most promising one, we have

$$P\{B_j \mid L_i\} = \begin{cases} 1 & B_j \text{ is controlling behavior in } L_i \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

and we can write

$$T^* = \arg\max_T \sum_{i=1}^{m} V_{i,ind(T(i))} P(L_i \text{ is cont.}). \quad (11)$$

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

$$\text{cardinality(ZO)} = n \cdot m. \quad (12)$$

Credit assignment is not difficult in this case. The updating way of V_{ij} is evident from its definition: if layer i is the controlling layer and B_j has been activated in it, we must update V_{ij} similar to what is common in stochastic parameter estimation, i.e.

$$V_{ij_{n+1}} = (1 - \alpha_{n,ij}) V_{ij_n} + \alpha_{n,ij} r_n \quad (13)$$

with following conditions ([10])

$$\begin{aligned} &\text{(I)} \ \lim_{n\to\infty} a_{n,ij} = 0 \\ &\text{(II)} \ \sum_{n=1}^{\infty} \alpha_{n,ij} = \infty \\ &\text{(III)} \ \sum_{n=1}^{\infty} \alpha_{n,ij}^2 < \infty \\ &\text{(IV)} \ \text{var}\{r_n\} \text{ is finite} \end{aligned} \quad (14)$$

C. First Order Representation

Here, we propose another method that uses relative position of behaviors in architecture instead of their absolute position as in the previous one. This representation is complete and has much smaller representation space than a one that stores all of the possible sequences. In this representation, we want to store order of behaviors. We define '$>$' inequality operator as

$$B_i \overset{T}{>} B_j : \text{B}_\text{i} \text{ is upper than B}_\text{j} \Big|_{\text{B}_\text{i}, B_j \in T}. \quad (15)$$

As in the previous case, we want to find the basic elements to decompose the value of the system into it. Defining the value of an ordering as

$$\begin{aligned} &V_{FO}(B_i > B_j) = V_{i>j} \\ &= E\left[r_t \left| \begin{matrix} B_i \text{ is controlling,} \\ B_j \text{ is the next active behavior} \end{matrix} \right. \right]. \end{aligned} \quad (16)$$

it is easy to write V_T (4) as

$$V_T = E[r_t] = \sum_{i=1}^{m} E\left[r_t \mid B_{ind(i)} \text{ is cont.}\right] P(L_i \text{ is cont.}) \tag{17}$$

which each value part of the right-hand side terms can be expanded into

$$\begin{aligned} &E\left[r_t \mid B_i \text{ is controlling}\right] = \\ &E\left[r_t \middle| B_i \text{ is controlling \& nobody else is active}\right] \\ &+ \sum_j E\left[r_t \middle| \begin{matrix} B_i \text{ is controlling \&} \\ B_j \text{ is the next active behavior} \end{matrix} \right] \\ &= V_{i0} + \sum_j V_{i>j} \end{aligned} \tag{18}$$

where

$$V_{i0} = E\left[r_t \middle| \begin{matrix} B_i \text{ is controlling \&} \\ \text{nobody else is active} \end{matrix} \right]. \tag{19}$$

It can be seen that these events are mutually exclusive as

$$P\left\{ \begin{matrix} (B_j \text{ is the next active behavior}) \,\& \\ (B_k \text{ is the next active behavior}) \end{matrix} \right\} = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases} \tag{20}$$

$$P\left\{ \begin{matrix} (B_j \text{ is the next active behavior}) \,\& \\ (\text{no other behavior is activated}) \end{matrix} \right\} = 0 \,. \tag{21}$$

Assuming that at least one of the behaviors becomes active, we can write the value of the total system as

$$V_T = \sum_{i=1}^{m} \left\{ V_{i0} + \sum_{j=ind(T(i+1))}^{ind(T(m))} V_{i>j} \right\} P(L_i \text{ is cont.}) \,. \tag{22}$$

To find the best structure, we must find the one that maximizes (22).

In order to assign credit in this representation, we should observe activation pattern of behaviors in the architecture. If only one behavior becomes activated, we must update V_{i0}. If two or more behaviors become active, we must update $V_{i>j}$ for i which is a controlling behavior and j which is the next active behavior. Updating rules are

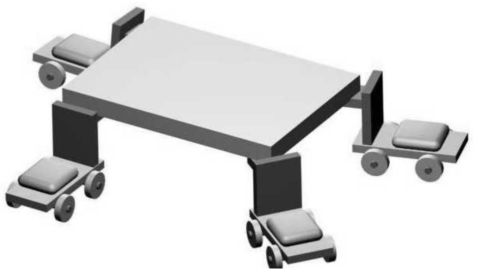
Figure 2. A group of robots lift a bulky object (note that there are four robots in this figure but we have used three robots in our simulations

$$V_{i0_{n+1}} = \left(1 - \alpha_{n,i0}\right) V_{i0_n} + \alpha_{n,i0} r_n \tag{23}$$

$$\begin{aligned} &V_{i>j_{n+1}} = \left(1 - \alpha_{n,i>j}\right) V_{i>j_n} + \alpha_{n,i>j} r_n \;; \\ &B_i > B_j \text{ and } j \text{ is the next active behavior} \end{aligned} \tag{24}$$

with conditions similar to (14). At last, cardinality of this representation space is rather small and is

$$cardinality(FO) = n^2 \,. \tag{25}$$

III. EXPERIMENTS

Here we consider the object lifting task as our test bed [11]. Imagine a situation in which a group of robots 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. Then, a fork lifting mechanism is probably the most suitable for handling the object. When lifting the object, based on the relative position of each robot to the object's center of gravity (c.g.), the required force to lift the object may vary from one robot to another, which introduces heterogeneity in the robot team. Consequently, each individual robot must have the ability to do its job under a range of external loads. In addition, when the object is tilting, each robot must move to prevent sliding at its contact point with the object. If some compliance is provided at the end effector (in the plane parallel to the object's lower surface) and the tilt angle of the object is kept small enough, there is no need for the robot to move while lifting the object. Keeping the inclination angle of the object within a specified range, will also prevent collision between the object and the lifting robots.

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

In [11], a SSA system that can lift an object with unknown mass and center of gravity cooperatively with no central control or communication between agents is hand-designed. Here we try to learn the architecture they developed.

Without going into details, we state that the problem is lifting an unknown object to a set point while keeping its tilt angle small. Here, we make a behavior repertoire similar to those that they used and let our methods learn the correct structure. Defining $z(k)$ as height of robot-object contact point, $v(k)$ as its elevation velocity, and $\tau(k)$ as the object's tilt angle at time step k (all of these quantities can be measured locally), the behaviors are defined and programmed as follow

Push more: $v(k+1) = v(k) + \Delta v$ (26)

Do not go fast: if $v(k) > v_{\max}$ then $v(k) = v_{\max}$ else do nothing (27)

Stop at goal: if $z(k) \ge z_{goal}$ then stop ($v(k+1) = 0$) (28)

Hurry up: if $\tau > \tau_0$ and the robot is the lowest one then $v(k) = \max(v(k) + \Delta v, v_{\max})$ (29)

Slow down: if $\tau > \tau_0$ and the robot is the highest one then $v(k) = \max(v(k) - \Delta v, 0)$ (30)

The results in this paper is obtained using $\Delta v = 1$, $v_{\max} = 5$, $z_{goal} = 3$, and $\tau_0 = 5^\circ$.

We have used our methods in two different problems. One of them is a synthetic abstract one which is defined specially to check the validity of our method. We will not go into its details and results obtained from it, and postpone it to another paper, instead we test our methods for this object lifting task to see whether they are practical in a real situation or not. The main difference between these two problems is that reinforcement signal cannot be easily defined in the latter one. For real situations, one must design a suitable reinforcement function that reflects all the necessary properties of the learnt system. We have designed the following reinforcement function, which seems to be a good candidate for this problem. We have not optimized them, so it is possible to find a better one.

$$r(k) = \begin{cases} 1 & \tau(k+1) - t(k) < -0.5 \\ -0.1 & \text{otherwise} \end{cases} \quad (31)$$

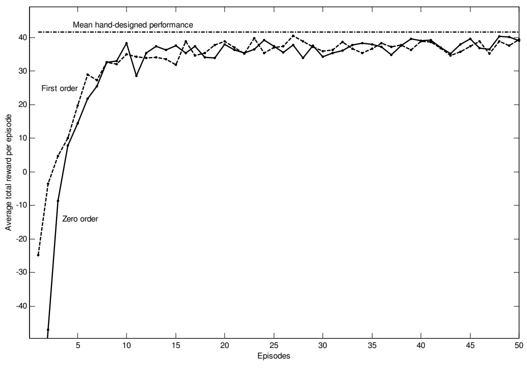

Figure 3. Comparison of zero order and first order methods with a hand-designed one.

$$r(k) = r(k) + \begin{cases} \dfrac{1}{\sqrt{k}} & \tau(k) < \tau_0 \\ -0.1\sqrt{k} & \text{otherwise} \end{cases} \quad (32)$$

$$r(k) = r(k) + \begin{cases} 1 & |z(k) - z_{goal}| < 0.5 \\ -0.1 & \text{otherwise} \end{cases} \quad (33)$$

$$r(k) = r(k) - 1 \qquad \mathrm{z(k)} > \mathrm{z_{goal}} + \delta_z \quad (34)$$

$$r(k) = r(k) - 0.1 \qquad \mathrm{v(k)} > \mathrm{v_{max}} \quad (35)$$

In order to clarify this function, let us discuss it briefly. (31) rewards reducing tilt angle and punish a movement that increases it, (32) 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. (33) rewards being near the goal and punishes being far from it and (34) punishes passing the goal. δ_z has been selected 0.2 for simulations of *zero order method* and 0.05 for *first order method*. And at last, (35) punishes a behavior that make a system move too fast.

For our simulations, we have averaged 100 runs of learning system each of which is trying 50 different episodes that are consisted of setting a random initial position for the object and make decision using the proposed architecture and updating value tables regarding reinforcement signal that is received. Learning rate was set $\alpha_0 = 0.1$ and was reduced using

$$a_{episode} = \alpha_0 (0.99)^{episode} \quad (36)$$

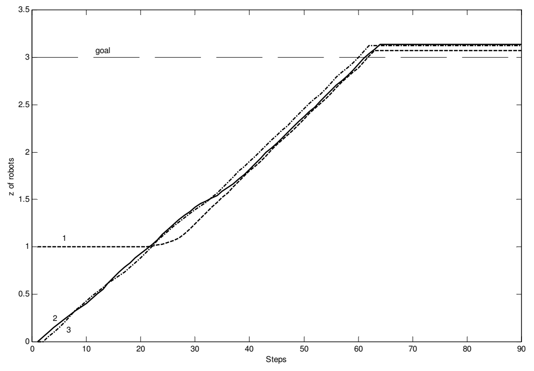

Figure 4. A sample simulation result showing the position of three robots during object lifting after sufficient learning (the architecture of the learned agent is the same as a hand-designed one).

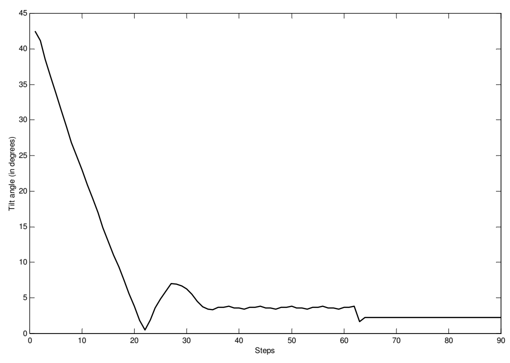

Figure 5. A sample simulation result showing the tilt angle of three robots during object lifting after sufficient learning (the architecture of the learned agent is the same as a hand-designed one).

We have added exploration capability for architecture making process. We have done it by doing a random change in optimal architecture. The time step in our simulations was $\Delta T = 0.01^s$. In Fig. 3 average performance of two different methods and a hand-designed one (which is the same as what has been reported in [11]) is compared. As can be seen, the performance is somehow lower than a hand-designed one but is rather satisfactory. Note that large amount of this difference is due to exploration we have made in architecture building process that may not select optimum architecture in order to find other possible good solutions. Beside that, our methods have found an architecture similar to the hand-designed one most of the times. The graphs show that both methods converge to the same results but first order method acts better in the first stages of learning.

In Fig. 4 and Fig. 5, elevations of three robots and object tilt angle when lifting the object are depicted. In this simulation, the robots implement the architecture learnt using zero order method. The figures show that the team has learned lifting the object without tilting it beyond the defined tilt angle in a distributed manner.

IV. CONCLUSIONS

We have proposed two different methods for behavior hierarchy learning in a purely parallel subsumption system using constructive methods. In these methods, representation and credit assignment are obtained together using a mathematical formulation. Simulations on distributed object lifting task by three robots show that these two methods act satisfactory and it seems that architecture learning is possible if a proper credit assignment function is designed.

Important directions for our future research include learning behaviors and architecture together and also architecture or behavior learning in other hierarchical architectures (Feudal Q-learning [12], Options [13], MaxQ [14] and Hierarchies of Abstract Machines [15]).

REFERENCES

[1] L. P. Kaelbling and M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," Journal of Artificial Intelligence Research, 4:237-285, 1996.

[2] A. Harati and M. Nili Ahmadabadi, "Experimental analysis for knowledge based multiagent credit assignment," Neural Information Processing: Research and Development, Rajapakse, Jagath C.; Wang, Lipo (Eds.), Springer-Verlog, May 2004.

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

[4] R. A. Brooks, "New approaches to robotics," Science 253, pp. 1227-1232, 1991.

[5] R. Brooks, "Intelligent without reason," in Proc. Int. Joint Conf. AI, pp. 569-595, 1991.

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

[7] S. Mahadevan and J. Connell, "Scaling reinforcement learning to robotics by exploiting the subsumption architecture," in 8th Int. Workshop on Machine Learning, Morgan Kaufmann, pp. 328-337, 1991.

[8] M. J. Mataric, "Reward function for accelerated learning," in W. W. Cohen & H. Hirsh, eds., Proc. 8th Int. Conf. Machine Learning, Morgan Kaufmann, pp. 181-189, 1994.

[9] M. J. Mataric, "Learning in behavior-based multi-robot systems: policies, models, and other agents," Cognitive System Research, special issue on Multi-disciplinary studies of multi-agent learning, Ron Sun, ed., 2(1), pp. 81-93, 2001.

[10] T. Jaakkola, M. I. Jordan, and S. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," Neural Computation, 6(6): 1185-1201, 1994.

[11] M. Nili Ahmadabadi and Eiji Nakano, "A Constrain and move approach to distributed object Manipulation," IEEE Tran. On Robotics and Automation, vol. 17, no. 2, pp. 157-172, 2001.

[12] P. Dayan and G. Hinton, "Feudal reinforcement learning," in Advances in Neural Information Processing Systems, 5, pp. 271-278, Morgan Kaufmann, 1993.

[13] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning," Artificial Intelligence, 112:181-221, 1999.

[14] T. G. Dietterich, "Hierarchical reinforcement learning with the MaxQ value function decomposition," Journal of Artificial Intelligence Research, 13: 227-303, 2000.

[15] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," In Advances in Neural Information Processing Systems: Proceedings of the 1997 Conference, 1998.