Adaptive Combination of Behaviors in an Agent

Olivier Buffet[1] and **Alain Dutech**[1] and **François Charpillet**[1]

Abstract. Agents are of interest mainly when confronted with complex tasks. We propose a methodology for the automated design of such agents (in the framework of Markov Decision Processes) in the case where the global task can be decomposed into simpler -possibly concurrent- sub-tasks. This is accomplished by automatically combining basic behaviors using Reinforcement Learning methods. The main idea is to build a global policy using a weighted combination of basic policies, the weights being learned by the agent (using Simulated Annealing in our case). These basic behaviors can either be learned or reused from previous tasks since they will not need to be tuned to the new task. Furthermore, the agents designed by our methodology are highly scalable as, without further refinement of the global behavior, they can automatically combine several instances of the same basic behavior to take into account concurrent occurences of the same subtask.

1 Introduction

Our researches aim at automatically designing the behavior of *reactive situated* agents limited to only *local perceptions*. Reinforcement Learning (RL)[9] can be applied in that field. Nevertheless good RL algorithms are usually used for simple cases and generally suffer from combinatorial explosion, as discussed in section 2.1.

To overcome these difficulties, we make the hypothesis, as in Brook's subsumption architecture [2], that a complex problem can be efficiently dealt with if considered as a combination of simple problems. An example is given by the tile-world represented on figure 1, where the agent has to manage three simple behaviors (the first one intends to avoid a hole, and the two others to push a tile in the hole).

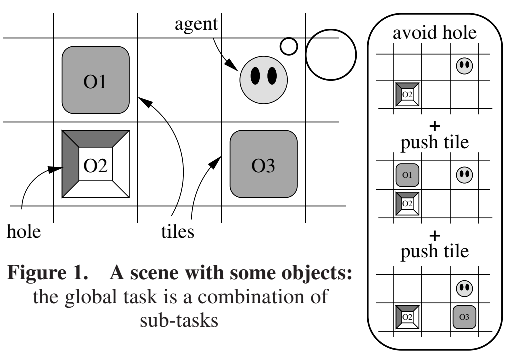

Figure 1. A scene with some objects: the global task is a combination of sub-tasks

Our method is twofold. First we provide our agent with basic behaviors, either through RL or transfers from previous tasks. Then, this basic behaviors are weighted and combined into the agent's global behavior. The main point is that the weights are *learned*. As such, we derive an *adaptive* (through learning relations between behaviors) and *scalable* agent (working with various world sizes).

The following section of this paper develops in more details the context of our work and presents previous works that inspired us. Then, in Section 3, we give the details of our algorithm with a special focus to learning and scalability. Section 4 is devoted to an experimental validation of our work on the well known tile-world problem. A discussion and a conclusion end this paper.

2 Framework

This section introduces the use of Reinforcement Learning in our context. Then an analysis of previous works shows their limitations and help us outline the solution we propose.

2.1 Reinforcement Learning and limitations

Reinforcement Learning (RL) methods are very appealing ways to have agents learn optimal reactive behaviors in uncertain worlds, as only a scalar feedback from the system to the agents is required.

But the convergence of RL algorithms (like *Q-Learning* or *TD(λ)*) has only been proven for Markov Decision Processes (MDP). A MDP is defined as a $< \mathcal{S}, \mathcal{A}, T, r >$ tuple, $\mathcal{S}$ being a finite set of states and $\mathcal{A}$ a finite set of actions. When the system is in given state s, an action a being chosen, the probability for the system to end in state s' is given by $T(s, a, s')$. After each transition, the environment generates a reward $r(s, a)$. The problem is then to find the optimal mapping $\pi(s, a)$ between states and actions so as to maximize the reward received over time, usually expressed as a utility function $Q(s, a) = E\left[\sum_{t=0}^{\infty} \gamma^t * r_t | s_0 = s, a_0 = a\right]$. Such a mapping is called a *policy* and, for a MDP, it is well known that an optimal *deterministic* policy exists [9].

As our agent only has a partial view of its environment, the learning task we are confronted with belongs to the more general class of Partially Observed Markov Decision Processes. Nevertheless, the assumption that the agent faces a Markovian problem is often made. This is truly a weak approximation and the policies learned this way are clearly sub-optimal as explained by [7]. In fact, it is better in that case to look for *stochastic* policies, using gradient descent algorithms for example (like in [1] or [6]).

Even under the Markovian approximation, the problem of combinatorial explosion remains. The number of an agent's possible observations can still be huge, even though the locality of its perceptions helps reducing it. Our algorithm takes advantage of the possibility to decompose the task in subtasks to address this specific problem.

[1] LORIA, BP 239 F-54506 Vandœuvre-lès-Nancy
{buffet,dutech,charp}@loria.fr

2.2 Previous and similar works.

A common idea to overcome the curse of dimensionality is to decompose the Markov Decision Process in some way. A taxonomy of MDP decompositions is proposed for example by Wang and Mahadevan in [10], in which our approach stands in the *action decomposition* class. But to our knowledge, there is up to now no satisfying solution to this class of problem in the learning framework.

2.2.1 Why we need a new algorithm

The starting point of our approach is to give the agent stochastic policies for each of its basic behaviors. The hard point is then to combine these basic policies to derive a global policy. At this point, two major directions can be taken.

A first option, as in the work of Humphrys [5], is to select one particular behavior as the one to be privileged for the immediate perception of the agent. To do this, the actions desired by each behavior and information about the utility of these actions are fed into a controller which determines the "leader" behavior. Many strategies for the controller have been studied by Humphrys, all based on his *W-Learning* framework. The best one requires an adaptation of the reward functions used to teach each basic behavior, as the selection of a behavior is based on minimizing the difference between the expected and received local reward associated to *each* behavior. As such, this adaptation is very specific to the global task.

Another option is to combine the basic policies into a new policy. This option is more appealing than the preceding one for several reasons. In some cases, as shown on figure 2, the best action to take is not given by any sub-policy but could emerge from a combined policy, especially if the sub-policies are deterministic. Besides, this option is less task-specific in the sense that we can re-use the basic behaviors for different complex tasks as we only need to adjust the weights of the behaviors and not the behaviors themselves.

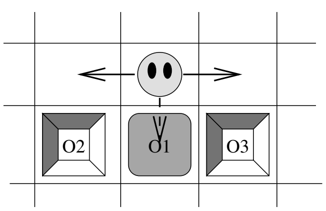

Figure 2. No leading behavior. The action desired by the basic `push` behaviors is either to go left $\{hole\ O_3, tile\ O_1\}$ or right $\{hole\ O_2,\ tile\ O_1\}$, whereas the optimal global action is to go down.

The work from Dixon et al [3] briefly describes a method for combining sub-policies where the probability distribution on the actions for each sub-task are linearly combined. A simple weight is associated to each sub-policy, and the problem is then to choose the right weights. As this method only intended to control their exploration, no particular attention were given on how to choose these weights. This is exactly the problem we will address in our algorithm by *learning these weights*, as explained in Section 3.4.3.

2.2.2 Scalability is desired

We lastly want to point out that existing methods do not offer scalability. When a basic behavior can be applied more than once in a given situation (for example, two holes are to be avoided), this must be specified *a priori* in the algorithms cited before. This specific point is also dealt with in our algorithm. To that end, we learn one weight for each "generic" basic behavior (such as *avoid hole*). Thus, even if many instances of the same generic behavior can be applied in a given situation (several holes to avoid), they will be combined using only one *common* weight. More details are given in Sections 3.3 and 3.4.3.

3 Proposition

The starting point of our approach is the idea that a complex behavior is obtained as an answer to many basic motivations. Moreover, simple basic behaviors can be easily associated to each of these motivations. Given these two points, we make the hypothesis that, in many cases, a good solution of the complete problem can be reconstructed using the basic behaviors. In this section, we will give more details on these basic behaviors, describe how a scene can be analysed through a decomposition in basic behaviors, and finally propose a method to automatically build a complex behavior by learning[2].

3.1 Some notations

An agent perceives a scene as an **observation**, which is composed of a set of percepts. Such an observation can be broken into **configurations** (i.e: subsets of the observation), and a percept can belong to different configurations. As each percept is characterized by a type (for example a hole, a tile or a door in our case), we also define the very important notion of a **type of configuration** which is described by a set of types of percepts (i.e. $\{hole, tile\}$). With these notations, the next section defines a basic behavior.

3.2 Basic behaviors

A **basic behavior** is defined by 1- a type of configuration, 2- a stochastic decision policy learned by reinforcement and 3- the utility of this policy. For a basic behavior b, we will note $\mathcal{C}^T(b)$ its type of configuration. This notion of type of configuration is essential for the scalability of our approach, since it allows generic basic behaviors to be instantiated several times in the same observation. Then, the policy P_b of the behavior is a mapping from configurations c of $\mathcal{C}^T(b)$ to probability distributions over actions : $P_b : \mathcal{C}^T(b) \times \mathcal{A} \longrightarrow [0, 1]$. Note that two different configurations can belong to the same type of configuration, so the agent can deal with them by using the same behavior (i.e. the agent can try to avoid two holes at the same time).

The knowledge of this policy for each basic behavior is not sufficient to take efficient decisions when the agent has to deal with concurrent motivations. To weight them in some way -giving a higher priority to danger avoidance or to important reward in sight- we suggest to evaluate a situation by using Q-values, as they will give us the expectation of discounted reward (i.e. the utility) of each configuration-action pair. This Q-values can be learned while also learning the policy of a basic behavior[3].

To sum it up, for each behavior b are calculated two tables [4] for an upcoming use:

- $P_b(c, a)$: the probability to choose action a while seeing configuration c, and
- $Q_b(c, a)$: the expected discounted reward when choosing action a for the configuration c.

[2] These basic behaviors could be usefully collected into a library which would be reusable in other situations.
[3] Q-values depend on the chosen policy.
[4] Both tables have the same definition set $\mathcal{C}^T(b) \times \mathcal{A}$.

How to use these basic behaviors is discussed in the following subsections. Please note that how to find which basic behaviors to use for a given complex problem is not discussed in this paper.

3.3 Scene decomposition by the agent

When an agent is confronted to a complex situation, it must decompose it into simpler known configurations in a *scalable* way. To this end, the first step is to look for "familiar" and "useful" configurations in the perceived situation. As any subset of percepts of the agent's current observation can be a configuration associated to a behavior, an agent will only consider configurations which belong to at least one type of configuration associated to one existing basic behavior.

For an observation o, let us call $\mathcal{U}(o)$ the set of these "useful" configurations. Then, for each useful configuration c of $\mathcal{U}(o)$ we will note $\mathcal{B}(c)$ the set of basic behaviors associated to this configuration. Similarly, $\mathcal{C}(b, o)$ will be the set of useful configurations in observation o associated to a given basic behavior b. Scalability derives mainly from the fact that one basic behavior (resp. configuration) can be associated to more than one configuration (resp. basic behavior). This is all the more interesting that, due to the locality of perceptions, the number of useful configurations changes.

In the tile-world scene presented on figure 1, the agent's perceptions concern objects O_1, O_2 and O_3. With two possible behaviors: $a$voiding the holes (b_a) associated to $\{hole\}$ and $p$ushing blocs in those holes (b_p) associated to $\{hole,\ tile\}$, the agent has to take into account three different $(behavior, configuration)$ pairs: $(b_a, \{O_2\})$, $(b_p, \{O_2, O_1\})$ and $(b_p, \{O_2, O_3\})$.

Among the hard points of the method we propose, a combinatorial explosion can be feared as far as the search for useful configurations is concerned. In practice, the number of objects seen remains usually small, as only local perception should be used.

3.4 Basic behaviors combination

3.4.1 *General formula*

The next step is to use these useful configurations to choose an action. There are several ways to make this choice (voting, biding, random choice) but we decided to compute a policy $\mathcal{P}(o, a)$ giving a probability distribution over actions a for each possible observation o. As written previously, we chose to define this policy as a recombination of basic behaviors using their P- and Q-tables. To be more precise, we try to define a linear combination of the P policies. The general formula is thus:

$$\mathcal{P}(o, a) = \frac{1}{K} \sum_{c \in \mathcal{U}(o)} \sum_{b \in \mathcal{B}(c)} w(b, c, a).P_b(c, a)$$

where $w(b, c, a)$ are some positive functions Q-values called *weights* and K is a normalizing factor ($\sum_a \mathcal{P}(o, a) = 1$). The action is then chosen according to the $\mathcal{P}$ distribution.

How to calculate $\mathcal{P}$ optimally using the Q- and P-tables is subject to discussion. Only the formula that gave us most satisfaction is presented in this paper.

3.4.2 *Chosen formula*

We chose w to depend on the Q-values. A first remark is that the absolute value of Q-values will be used, in a view to give the same importance to future earnings ($Q > 0$) and immediate danger ($Q < 0$). If these Q-values seem to give good comparisons between state-action pairs of a single behavior, the relative importance of different Q-tables can be efficiently corrected by learning a parameter θ_b for each behavior (appearing as a factor e^{θ_b}).

For a given observation-action pair (o, a), the idea is to consider that each behavior tells that its probability $P_b(c, a)$ is the right one with a force of conviction $|e^{\theta_b} * Q_b(c, a)|$ [5]. This leads to compute the mean of the $P_b(c, a)$ probabilities weighted by $|e^{\theta_b} * Q_b(c, a)|$ (for each current behavior):

$$\mathcal{R}(o, a) = \frac{1}{k_{(o,a)}} \sum_{c \in \mathcal{U}(o)} \sum_{b \in \mathcal{B}(c)} e^{\theta_b}.|Q_b(c, a)|.P_b(c, a)$$

$$(k_{(o,a)} = \sum_{c \in \mathcal{U}(o)} \sum_{b \in \mathcal{B}(c)} e^{\theta_b}.|Q_b(c, a)|)$$

After normalizing and putting in common e^{θ_b} for all instances of a type of behavior, the final version is rewritten as:

$$\mathcal{P}(o, a) = \frac{1}{K} \cdot \frac{1}{k_{(o,a)}} \sum_{b \in \mathcal{B}} \underbrace{e^{\theta_b}}_{to\ learn} \left[\underbrace{\sum_{c \in \mathcal{C}(b,o)} |Q_b(c, a)|.P_b(c, a)}_{already\ known} \right]$$

3.4.3 *Learning and Scalability*

Each set of θ parameters defines a global complex policy for the agent. Tuning the weights of the formula is like learning an optimal parameterized policy in the framework of reinforcement learning. To that end, we have simply used a straightforward simulated annealing algorithm with a geometric decrease of the temperature (only a few parameters have to be learned: one for each behavior).

The scalability of the process derives also from the fact that only one θ_b coefficient needs to be defined for each behavior. Even when many configurations are associated to one behavior, the complex policy can be computed without further learning or refining of the parameters.

In the next section, we present an example showing an application of our methodology on the tile-world problem.

4 Experiments

4.1 The tile-world

4.1.1 *Problem*

The tile-world is a grid domain in which a cell may contain a hole, a tile or an agent. In the complete problem, the agent (if we consider only one agent) has to push tiles in holes as often as possible, while avoiding to go itself in one of those holes.

To give some details about the simulation, the agent can go freely in a hole (and also go out), but will get a negative reward doing so. Moreover, when a tile is pushed in a hole, both the tile and the hole disappear and reappear anywhere on the grid. Finally, to avoid some blocking situations, holes and tiles cannot be on cells of the grid's border.

In this complete complex problem, many tiles and holes must be handled. As it appears in previous examples, a simple decomposition of the problem in basic behaviors can be made: `[avoid hole]` $(b_a, \{hole\})$ and `[push tile in hole]` $(b_p, \{hole,\ tile\})$, as shown on figure 1.

[5] With a single non-zero reward r, the Q-table is proportional to r.

4.1.2 *Agent's skills*

In these experiments, the agent has always all other objects of the environment in sight. The principle of locality is nevertheless present in the fact that perceptions are unprecise. For any object O in the scene, the agent's perception of O gives:

- `near(`O`)`: tells if object O is in the 9-cells square centered on the agent `(true|false)`,
- direction(O): gives the object's direction `(N-NE-E-SE-S-SW-W-NW)`.

The only actions available for an agent are to move one cell North, South, East or West (it cannot ask to stay on a cell). And to conclude, the reward given is $+1$ when a tile falls in a hole, -3 when the agent goes in a hole, and 0 otherwise.

4.2 Conducted experiments

To assess the method described in this paper, we first learn the needed basic behaviors, then also learn complex behaviors by classical approaches (so as to have references), and we finally compare these behaviors with the ones obtained by recombination.[6]

4.2.1 *Basic behaviors' preparation*

The first step was simply to learn the basic behaviors we would have to use. The evolution of both policies' efficiencies is shown on the two sample runs of figure 3, where the x axis gives the number of simulation steps (to be multiplied by 10000), and the y axis gives the agent's reward for the last 10000 simulation steps. In both cases, the reward must be maximized, with a maximum of zero for the *avoid* behavior (since the reward is always negative), and a positive maximum for the *push* behavior.

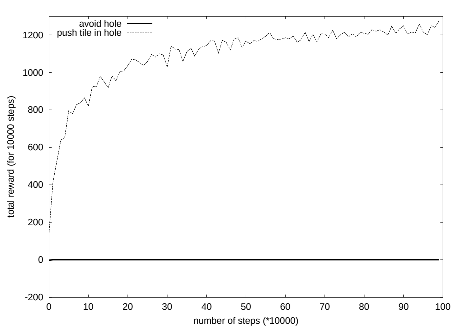

Figure 3. learning of basic behaviors (just notice that the very simple `[avoid hole]` behavior is practically immediately learned)

4.2.2 *Classical approach*

In order to compare our approach with classical Reinforcement Learning, we tried to apply both a gradient ascent [1] and an adapted Bolzmann Q-learning on the complete tile-world problem with different numbers of tiles and holes in the environment. Whereas in the `[1 tile/1 hole]` case learning is quite fast and efficient, there is a really fast increase of the difficulty when adding other objects. The gradient ascent did only succeed in the simplest case, bringing us to present just the results of the Q-learning.

The sizes of the different observation-spaces shown on table 1 explain this phenomenon[7]. It appears that with these large problems the gradient method is prone to fall in local optima: the agent only avoids holes and does not push tiles in holes.

Table 1. Size of the different observation-spaces

objects		size of the	number of
#tiles	#holes	observation-space	configurations
1	1	$16^2/4 = 64$	$1+1=2$
2	1	$16^3/4 = 1024$	$1+2=3$
1	2	$16^3/4 = 1024$	$2+2=4$
2	2	$16^4/4 = 16384$	$2+4=6$

The results shown on figure 4 may be far from optimum solutions, but are quite realistic: the more objects are present, the more the agent suffers from its unprecise perceptions (and is obstructed in its moves).

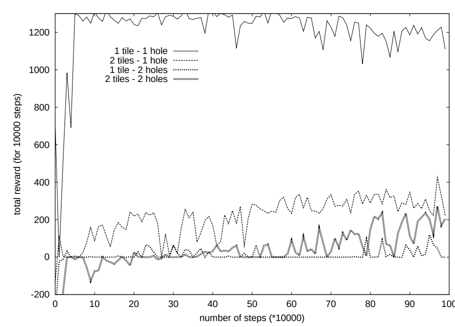

Figure 4. Examples of complex behaviors learned tabula rasa
(the slow decrease of the highest curve is a classical phenoma due to an evolution toward a deterministic behavior)

4.2.3 *Comparisons*

At last an agent -knowing the two basic behaviors- is put in the same situations, and learns the weights to balance both behaviors. The statistical results that follow have been collected after learning the weights. There were only few variations in our samples, so this means are nearly always reached.

Basic behaviors' recombinations are compared to the policies learned tabula rasa in the first four lines of table 2. We used in one case "optimal" basic behaviors (column $c(1)$), and more noisy basic behaviors (column $c(2)$) in a second set of experiments (where each action has at least a 5% probability of being chosen), as we hope to get out of some blocking situations. Even in the case of figure 2, the learned weights give enough importance to the `[avoid hole]` behavior to prevent the agent from falling into the holes.

The quality of the solutions with at most four objects is very satisfying if we consider that the average reward reached is at least as good as the one obtained through the classical approach (except in the simplest case). Moreover, the comparison of columns $c(1)$ and $c(2)$ shows –as we expected– that adding some noise may be useful in certain situations.

The three last situations presented in table 2 (at least five objects) could not be learned tabula rasa. The results presented for them are

[6] The environment considered always has a size of 6×6 cells.

[7] The division by 4 is possible since the problem does not depend on the orientation.

Table 2. Comparative table between policies obtained by recombination and tabula rasa (see text for details)

objects		reward (for 10000 steps)		
#tiles	#holes	tabula rasa	**c(1)**	**c(2)**
1	1	**1300**	1017	842
2	1	300	302	**493**
1	2	~ 0	**405**	259
2	2	200	283	**411**
3	2	-	166	134
2	3	-	262	247
3	3	-	179	129

- c(1): our combination of policies
- c(2): the same combination with noise added in the decisions of the basic behaviors

the efficiencies obtained in an 8×8 grid (to have enough space) **reusing** the θ parameters learned with `[2 tiles/2 holes]`.

5 Discussion

5.1 Optimality ?

It is clear that, with our method, we cannot be sure of reaching optimality. One of the reasons is that, as we use only a combination of basic policies, only a subclass of the possible policies can be explored. Furthermore, because of their limited perception, the agents are confronted to a non-Markovian task. As a consequence, the Markovian approximation we use cannot give an optimal result. Finally, the simulated annealing which learns the weights of the global behavior converges only to *local* optima. The lack of optimality is the price to pay for a tractable algorithm.

5.2 Combination of basic behaviors

One could argue that a finer level of combination could improve the performances of our algorithm. This could be achieved with weights depending not only the behaviors, but also on the actions, observations... The drawbacks of this approach lies in a bigger number of parameters to learn. One would have to pay attention to scalability and genericity issues.

Another possibility for a finer control of the global behavior would be to make use of the sign of the utility of a basic policy. This way, we could for example make a clear distinction between basic "negative" behaviors (forbidden actions) and "positive" behaviors that could just promote actions. To do that would require altering the policy combination function but without bringing too much task-specific knowledge into the process.

5.3 For a greater scalability

The "additive" nature of our present combination of behaviors is well adapted to additive environments (where the global utility is the sum of sub-tasks utilities). When it is not the case, too many instances of the same basic behavior (many tiles) could abusively overweigh other behaviors (avoid hole). One solution could be the use of a "critic" module trying to predict the global utility so as to adapt the way the behaviors are handled (additive, concurrent, preemptive...). This is yet another argument in favor of further work on behaviors' combination.

5.4 Scope of our methodology

The application presented in Section 4 to validate our approach belongs to what Wang and Mahadevan [10] called the *action decomposition* class of problem. As our framework makes no assumption on the transition and reward functions of the system, it should also be well suited to problems of *policy decomposition*. In fact, we have also tested our approach on a modified version of the prey-predator problem (see [8]) where the predators must also avoid obstacles in their environment. Because of the limited perceptions of the agents, the policy of one agent alters the transitions probability of the other agents. Actually, even the basic behaviors are more difficult to learn and a special kind of incremental Reinforcement Learning [4] had to be used. Nevertheless, the tests we conducted are highly satisfying and show good performances. Lack of space prevented us to present this more complex application here.

6 Conclusion

This paper presented an automated process for designing agents solving a complex task. In the general framework of Reinforcement Learning, our algorithm learns how to combine basic low-level behaviors into complex ones. Thus, as many complex problems are a combination of simpler tasks, which may be concurrent, we can take advantage of this decomposition for automatically building our agents. Scalability is another strength of our method as it allows an agent to dynamically combine several instances of the same type of basic behaviors without having to learn or modify its global behavior.

The main idea of our work is to learn how to weight the different basic behaviors needed by the agent, with only *one* weight by *type* of behavior. This was made possible by using *generic* basic behaviors which, through their associated *type of configuration*, can be instantiated several times. The validity of the approach is tested on a classical tile-world problem and can be used on a wide range of complex problems. Future work will focus on improving the combination of basic behaviors in order to build finer complex behaviors.

REFERENCES

[1] J. Baxter and P. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[2] R. Brooks. A robust layered control system for a mobile robot. Technical report, September 1985.

[3] K. Dixon, R. Malak, and P. Khosla. Incorporating prior knowledge and previously learned information into reinforcement learning agents. Technical report, Carnegie Mellon University, 2000.

[4] A. Dutech, O. Buffet, and F. Charpillet. Multi-agent systems by incremental gradient reinforcement learning. In *Proc. of IJCAI'01*, 2001.

[5] M. Humphrys. Action selection methods using reinforcement learning. In *Proc. of SAB'96*, September 1996.

[6] L. Peshkin, K. Kim, N. Meuleau, and L. Kaelbling. Learning to cooperate via policy search. In *Proc. of UAI'00*, 2000.

[7] S. Singh, T. Jaakkola, and M. Jordan. Learning without state estimation in partially observable markovian decision processes. In *Proc. of ICML'94*, 1994.

[8] P. Stone and M. Veloso. Multiagent systems: a survey from a machine learning perspective. *Autonomous Robotics*, 8(3), 2000.

[9] R. Sutton and G. Barto. *Reinforcement Learning*. Bradford Book, MIT Press, 1998.

[10] G. Wang and S. Mahadevan. Hierarchical optimization of policy-coupled semi-Markov decision processes. In *Proc. of ICML'99*, 1999.