Modular Reinforcement Learning: A Case Study in a Robot Domain

Zsolt Kalmár * Csaba Szepesvári † András Lőrincz ‡

Abstract

The behaviour of reinforcement learning (RL) algorithms is best understood in completely observable, finite state- and action-space, discrete-time controlled Markov-chains. Robot-learning domains, on the other hand, are inherently infinite both in time and space, and moreover they are only partially observable. In this article we suggest a systematic design method whose motivation comes from the desire to transform the task-to-be-solved into a finite-state, discrete-time, "approximately" Markovian task, which is completely observable, too. The key idea is to break up the problem into subtasks and design controllers for each of the subtasks. Then operating conditions are attached to the controllers (together the controllers and their operating conditions which are called modules) and possible additional features are designed to facilitate observability. A new discrete time-counter is introduced at the "module-level" that clicks only when a change in the value of one of the features is observed. The approach was tried out on a real-life robot. Several RL algorithms were compared and it was found that a model-based approach worked best. The learnt switching strategy performed equally well as a handcrafted version. Moreover, the learnt strategy seemed to exploit certain properties of the environment which could not have been seen in advance, which predicted the promising possibility that a learnt controller might overperform a handcrafted switching strategy in the future.

1 Introduction

Reinforcement learning (RL) is the process of learning the coordination of concurrent behaviours and their timing. A few years ago Markovian Decision Problems (MDPs) were proposed as the model for the analysis of RL [17] and since then a mathematically well-founded theory has been constructed for a large class

*Department of Informatics JATE, Szeged, Aradi vértanúk tere 1, Hungary, H-6720; Present adress: Mindmaker Ltd. Budapest 1121, Konkoly Thege M. út 29-33; email: kalmar@mindmaker.hu

†Research Group on Art. Int., JATE, Szeged, Aradi vértanúk tere 1, Hungary, H-6720; Present adress: Mindmaker Ltd. Budapest 1121, Konkoly Thege M. út 29-33; email: szepes@mindmaker.hu

‡Eötvös Loránd University, Budapest, Múzeum krt. 8, Hungary H-1068; email: lorincz@iserv.iki.kfki.hu

of RL algorithms. These algorithms are based on modifications of the two basic dynamic-programming algorithms used to solve MDPs, namely the value- and policy-iteration algorithms [25, 5, 10, 23, 18]. The RL algorithms learn via experience, gradually building an estimate of the optimal value-function, which is known to encompass all the knowledge needed to behave in an optimal way according to a fixed criterion, usually the expected total discounted-cost criterion. The basic limitations of all of the early theoretical results of these algorithms was that they assumed finite state- and action-spaces, and discrete-time models in which the state information too was assumed to be available for measurement. In a real-life problem however, the state- and action-spaces are infinite, usually non-discrete, time is continuous and the system's state is not measurable (i.e., with the latter property the process is only partially observable as opposed to being completely observable). Recognizing the serious drawbacks of the simple theoretical case, researchers have begun looking at the more interesting yet theoretically more difficult cases (see e.g. [11, 16]). To date, however, no complete and theoretically sound solution has been found to deal with such involved problems. In fact the above-mentioned learning problem is indeed, intractable owing to partial-observability. This result follows from a theorem of Littman [9].

In this paper an attempt is made to show that RL can be applied to learn real-life tasks when *a priori* knowledge is combined in some suitable way. The key to our proposed method lies in the use of high-level modules along with a specification of the operating conditions for the modules and other "features", to transform the task into a finite-state and action, completely-observable task. Of course, the design of the modules and features requires a fair amount of *a priori* knowledge, but this knowledge is usually readily available. In addition to this, there may be several possible ways of breaking up the task into smaller subtasks but it may be far from trivial to identify the best decomposition scheme. If all the possible decompositions are simultaneously available then RL can be used to find the best combination. Here we propose design principles and theoretical tools for the analysis of learning and demonstrate the success of this approach via *real-life* examples. A detailed comparison of several RL methods, such as Adaptive Dynamic Programming (ADP), Adaptive Real-Time Dynamic Programming (ARTDP) and Q-learning is provided, having been combined with different exploration strategies.

The article is organized in the following way. In Section 2 we introduce our proposed method and discuss the motivations behind it. The notion of "approximately" stationary MDPs is also introduced as a useful tool for the analysis of "module-level" learning. Then, in Section 3 the outcome of certain experiments using a mobile robot are presented. The relationship of our work to that of others is contrasted in Section 4. Finally our conclusions and possible directions for further research are given in Section 5. Some details were left out from this article, but these can be found in [8].

2 Module-based Reinforcement Learning

First of all, we will briefly run through Markovian Decision Problems (MDPs), a value-function approximation-based RL algorithm to learn solutions for MDPs and their associated theory. Next, the concept of recursive-features and time discretization based on these features are elaborated upon. This is then followed by a sensible definition and principles of module-design together with a brief explanation of why the modular approach can prove successful in practice.

2.1 Markovian Decision Problems

RL is the process by which an agent improves its behaviour from observing its own interactions with the environment. One particularly well-studied RL scenario is that of a single agent minimizing the expected-discounted total cost in a discrete-time finite-state, finite-action environment, when the theory of MDPs can be used as the underlying mathematical model. A finite MDP is defined by the 4-tuple (S, A, p, c), where S is a finite set of states, A is a finite set of actions, $p : S \times A \times S \rightarrow [0,1]$ is a transition probability function satisfying $\sum_{s' \in S} p(s,a,s') = 1$ for all $(s,a) \in S \times A$ pairs and $c : S \times A \rightarrow \Re$ is the so-called immediate cost-function. The ultimate target of learning is to identify an optimal policy. A policy is some function that tells the agent which set of actions should be chosen under which circumstances. A policy π is optimal under the *expected discounted total cost criterion* if, with respect to the space of all possible policies, π results in a minimum expected discounted total cost for all states. The optimal policy can be found by identifying the optimal value-function, defined recursively by

$$v^*(s) = \min_{a \in U(s)} \left(c(s,a) + \gamma \sum_{s'} p(s,a,s')\, v^*(s') \right)$$

for all states $s \in S$, where $c(s,a)$ is the immediate cost for taking action a from state s, γ is the discount factor, and $p(s,a,s')$ is the probability that state s' is reached from state s when action a is chosen. $U(s)$ is the set of admissible actions in state s. The policy which for each state selects the action that minimizes the right-hand-side of the above fixed-point equation constitutes an optimal policy. This yields the result that to identify an optimal policy it is sufficient just to find the optimal value-function v^*. The above simultaneous non-linear equations (non-linear because of the presence of the minimization operator), also known as the *Bellman equations* [3], can be solved by various dynamic programming methods such as the value- or policy-iteration methods [15].

RL algorithms are generalizations of the DP methods to the case when the transition probabilities and immediate costs are unknown. The class of RL algorithms of interest here can be viewed as variants of the value-iteration method: these algorithms gradually improve an estimate of the optimal value-function via learning from interactions with the environment. There are two possible ways to learn the optimal value-function. One is to estimate the model (i.e., the transition probabilities and immediate costs) while the other is to estimate the optimal action-values

Initialization: Let $t = 0$, and initialize the utilized model (M_0) and the Q-function (Q_0)
Repeat Forever
1. Observe the next state s_{t+1} and reinforcement signal c_t .
2. Incorporate the new experience (s_t, a_t, s_{t+1}, c_t) into the model and into the estimate of the optimal Q-function: $(M_{t+1}, Q_{t+1}) = F_t(M_t, Q_t, (s_t, a_t, s_{t+1}, c_t))$.
3. Choose the next action to be executed based on (M_{t+1}, Q_{t+1}): $a_{t+1} = S_t(M_{t+1}, Q_{t+1}, s_{t+1})$ and execute the selected action.
4. $t := t + 1$.

Table 1: **The structure of value-function-approximation based RL algorithms.**

directly. The optimal action-value of an action a given a state s is defined as the total expected discounted cost of executing the action from the given state and proceeding in an optimal fashion afterwards:

$$Q^*(s,a) = c(s,a) + \gamma \sum_{s'} p(s,a,s')v^*(s'). \tag{1}$$

The general structure of value-function-approximation based RL algorithms is given in Table 1.

In the RL algorithms various models are utilized along with an update rule F_t and action-selection rule S_t.

In the case of the Adaptive Real-Time Dynamic Programming (ARTDP) algorithm the model consists (M_t) of the estimates of the transition probabilities and costs, the update-rule F_t being implemented, e.g., as an averaging process. Instead of the optimal Q-function, the optimal value-function is estimated and stored to spare storage space, and the Q-values are then computed by replacing the true transition probabilities, costs and the optimal value-function in Equation 1 by their estimates. An update of the estimate for the optimal value-function is implemented by an asynchronous dynamic programming algorithm using an inner-loop in Step 2 of the algorithm.

Another popular RL algorithm is Q-learning, which does not employ a model but instead the Q-values are updated directly according to the iteration procedure [25]

$$Q_{t+1}(s_t,a_t) = (1-\alpha_t(s_t,a_t))Q_t(s_t,a_t) + \alpha_t(s_t,a_t)(c_t + \gamma \min_a Q_t(s_{t+1},a)),$$

where $\alpha_t(s_t,a_t) \geq 0$, and satisfies the usual Robbins-Monro type of conditions. For example, one might set $\alpha_t(s,a) = \frac{1}{n_t(s,a)+1}$ where $n_t(s,a)$ is the number of time the state-action pair (s,a) was visited before time t. But often in practice $\alpha_t(s,a) =$ const is employed which while yielding increased adaptivities no longer ensures convergence.

Both algorithms mentioned previously are guaranteed to converge to the optimal value-/Q-function if each state-action pair is updated infinitely often. The action selection procedure S_t should be carefully chosen so that it fits the dynamics of the controlled process in a way that the condition is met. For example, the execution of random actions meets this "sufficient-exploration" condition when the MDP is ergodic. However, if on-line performance is important then more sophisticated exploration is needed which, in addition to ensuring sufficient exploratory behaviour, exploits accumulated knowledge. For more details the interested reader is referred to [8].

2.2 The modular architecture

In case of a real-life robot-learning task the dynamics cannot be formulated exactly as a *finite* MDP, nor is the state information available for measurement. This latter restriction is modelled by Partially-Observable MDPs (POMDPs) where (in the simplest case) one extends an MDP with an *observation function* h which maps the set of states S into a set X, called the observation set (which is usually non-countable, just like S). The defining assumption of a POMDP is that the full state s can be observed only through the observation function, i.e., only $h(s)$ is available as input and this information alone is usually insufficient for efficient control since h is usually a non-injection (i.e., h may map different states to the same observations).

The dynamics of the controlled system is defined by $P(s_{t+1} = s'|s_t = s, a_t = a) = p(s,a,s')$ and $x_t = h(s_t)$. The first part of the controller is the feature extraction part. The designer needs to design a feature space F together with a feature-extraction function R mapping $X \times A \times F^k$ to F, where k is another design parameter. The feature extraction function R transforms observation-action pairs into features in a recursive way: the feature f_t at time t is defined by $f_t = R(x_t, a_t, f_{t-1}, \dots, f_{t-k})$, $t \geq 0$, where $f_{t-1}, \dots, f_{t-k}$ are other design parameters. The rest of the system is composed of a finite number of controllers, $M^{(1)}, \dots, M^{(n)}$, where $M^{(i)} = (Z^{(i)}, \delta^{(i)})$, $\delta^{(i)} : Z^{(i)} \times F \to \{0,1\} \times A \times Z^{(i)}$. The ith controller stores an internal "state" $z_t^{(i)}$ which develops in time according to $z_{t+1}^{(i)} = \delta^{(i)}(z_t^{(i)}, f_t)$. $\delta_1^{(i)}(z_t^{(i)}, f_t)$ determines whether the ith controller is available for control in time step t: If $\delta_1^{(i)}(z_t^{(i)}, f_t) = 1$ then the controller is available, otherwise it is not available. Each controller should be thought of as a "local" controller that is able to carry out a particular subtask of the whole problem. The control problem then is to design a switching strategy that switches between the appropriate modules (local controllers) such that the controlled system will eventually show a behavior consistent with the ultimate goal of control. This is formulated as follows: we further restrict the switching function to be a memoryless mapping $S : F' \times \{0,1\}^n \to \{1, 2, \dots, n\}$. Here F' is another design set that comes together with a mapping $\pi : F \to F'$. π maps computed features to "monitored" features. The purpose of π is to bring in some more flexibility in the design procedure. The role of π will be clear soon once the control equations are given. In order to arrive at the definition of control, let use first define the sequence $\tau_0, \tau_1, \dots$ as follows. Let $\tau_0 = 0$ and let $\tau_1, \tau_2, \dots$ be defined by $\tau_{s+1} = \min\{\, t > \tau_s : \delta_1^{(i)}(z_t^{(i)}, f_t) \neq \delta_1^{(i)}(z_{t-1}^{(i)}, f_{t-1})$ for some i or $\pi(f_t) \neq \pi(f_{t-1}) \,\}$. A switching function $S : F' \times \{0,1\}^n \to \{1, 2, \dots, n\}$ is called admissible if from $S(f, c_1, \dots, c_n) = i$ it follows that $c_i = 1$ (i.e. only controllers which are available for control are chosen by the switching function). Given an admissible switching function S the control works as follows: at any given time instant t there is only one module active. The index of this module is denoted by m_t and satisfies $m_{t+1} = m_t$ if $t \notin \{\tau_0, \dots, \tau_s, \dots\}$, otherwise $m_{t+1} = S(\pi(f_t), \delta_1^{(1)}(z_t^{(1)}, f_t), \dots, \delta_1^{(n)}(z_t^{(n)}, f_t))$. The control is then given by $a_t = \delta_2^{(m_t)}(z_t^{(m_t)}, f_t)$.

Assume a goal oriented task, i.e. a POMDP where the success of a controller is measured in terms of if the system state can be driven to a particular set of goal states. Then the goal of the design procedure is to set up the modules and the additional features in such a way that there exists an admissible switching function S that for any given history results in a closed-loop behaviour which fulfills the "goal" of control. It can be extremely hard to prove even the *existence* of such a valid switching function. One approach is to use a so-called *accessibility decision problem* which is a discrete graph with its node set being $F' \times \{0,1\}^n$ and the edges connect features which can be observed in succession. Then, standard *DP* techniques can be used to decide the existence of a proper switching function [8].

Of course, since the definitions of the modules and features depend on the designer, it is reasonable to assume that by clever design a satisfactory decomposition and controllers could be found even if only qualitative properties of the controlled object were known. RL could then be used for two purposes: either to find the best switching function assuming that at least two proper switching functions exist, or to decide empirically whether a valid switching controller exists at all. The first kind of application of RL arises as result of the desire to guarantee the existence of a proper switching function through the introduction of more modules and features than is minimally needed. But then, good switching which exploits the capabilities of all the available modules could well become complicated to find manually.

If the accessibility decision problem were extendible with transition-probabilities to turn it to an MDP [1], then RL could be rightly applied to find the best switching function. For example, if one uses a fixed (maybe stochastic) stationary switching policy and provided that the system dynamics can be formulated as an MDP then there is a theoretically well-founded way of introducing transition-probabilities (see [16]). Unfortunately, the resulting probabilities may well depend on the switching policy which can prevent the convergence of the RL algorithms. However, the following "stability" theorem shows that the difference of the cost of optimal policies corresponding to different transition probabilities is proportional to the extent the transition probabilities differ, so we may expect that a slight change in the transition probabilities does not result in completely different optimal switching policies and hence, as will be explained shortly after the theorem, we may expect RL to work properly, after all.

Theorem 2.1 *Assume that two MDPs differ only in their transition-probability matrices, and let these two matrices be denoted by p_1 and p_2. Let the corresponding*

[1] Note that as the original control problem is deterministic it is not immediate when the introduction of probabilities can be justified. One idea is to refer to the ergodicity of the control problem.

optimal cost-functions be v_1^* *and* v_2^*. *Then,*

$$||v_1^* - v_2^*|| \leq \gamma \frac{nC||p_1 - p_2||}{(1-\gamma)^2},$$

where $C = ||c||$ *is the maximum of the immediate costs,* $||\cdot||$ *denotes the supremum-norm and* n *is the size of the state-space.*

Proof: Let T_i be the optimal-cost operator corresponding to the transition-probability matrix p_i, i.e.,

$$(T_i v)(s) = \min_{a \in U(x)} \left(c(s,a) + \gamma \sum_{s' \in X} p_i(s,a,s')v(s') \right),$$

$$v : S \to \Re, \, i = 1, 2.$$

Proceeding with standard fixpoint and contraction arguments (see e.g. [19]) we get that $||v_1^* - v_2^*|| \leq ||T_1 v_1^* - T_1 v_2^*|| + ||T_1 v_2^* - T_2 v_2^*||$ and since T_1 is a contraction with index γ, and the inequality $||T_1 v - T_2 v|| \leq \gamma ||p_1 - p_2|| \sum_{y \in X} |v(y)|$ we obtain $\delta = ||v_1^* - v_2^*|| \leq \gamma\delta + \gamma||p_1 - p_2|| |X| C/(1-\gamma)$, where $||v_1^*|| \leq C/(1-\gamma)$ has been employed [15]. Rearranging the inequality in terms of δ then yields Theorem 2.1. Q.E.D.

Motivated by the previous theorem we define ε-stationary MDPs as the quadruple (S, A, p, c), where S, A and c are as before but p, the transition probability matrix, may vary in time but with $||p_t - p^*|| \leq \varepsilon$ holding for all $t > 0$. Our expectations are that although the transitions cannot be modelled with a fixed transition probability matrix (i.e., stationary MDP), they can be modelled by an ε-stationary one even if the switching functions are arbitrarily varied and we conjecture that RL methods would then result in oscillating estimates of the optimal value-function, but with the oscillation being asymptotically proportional to ε. Note that ε-stationarity was clearly observed in our experiments which we will describe now.

3 Experiments

The validity of the proposed method was checked with actual experiments carried out using a Khepera-robot. The robot, the experimental setup, general specifications of the modules and the results are all presented in this section.

3.1 The Robot and its Environment

The mobile robot employed in the experiments is shown in Figure 1.

It is a Khepera[2] robot equipped with eight IR-sensors, six in the front and two at the back, the IR-sensors measuring the proximity of objects in the range

[2]The Khepera was designed and built at Laboratory of Microcomputing, Swiss Federal Institute of Technology, Lausanne, Switzerland.

Figure 1: **The Khepera and the experimental environment.** The task was to grasp a ball and hit the stick with it.

0-5 cm. The robot has two wheels driven by two independent DC-motors and a gripper which has two degrees of freedom and is equipped with a resistivity sensor and an object-presence sensor. The vision turret is mounted on the top of the robot as shown. It is an image-sensor giving a linear-image of the horizontal view of the environment with a resolution of 64 pixels and 256 levels of grey. The horizontal viewing-angle is limited to about 36 degrees. This sensor is designed to detect objects in front of the robot situated at a distance spanning 5 to 50 cm. The image sensor has no tilt-angle, so the robot observes only those things whose height exceeds 5 cm.

The learning task was defined as follows: find a ball in an arena, bring it to one of the corners marked by a stick and hit the stick with the ball. The robot's environment is shown in Figure 1. The size of the arena was 50 cm x 50 cm with a black coloured floor and white coloured walls. The stick was black and 7 cm long, while three white-coloured balls with diameter 3.5 cm were scattered about in the arena. The task can be argued to have been biologically inspired because it can be considered as the abstraction of certain foraging tasks or a "basketball game". The environment is highly chaotic because the balls move in an unpredictable manner and so the outcome of certain actions is not completely predictable, e.g., a grasped ball may easily slip out from the gripper.

3.2 The Modules

3.2.1 Subtask decomposition

Firstly, the task was decomposed into subtasks. The following subtasks were naturally: (T1) to find a ball, (T2) grasp it, (T3) bring it to the stick, and (T4) hit the stick with the grasped ball. Subtask (T3) was further broken into two subtasks, that of (T3.1) 'safe wandering' and (T3.2) 'go to the stick', since the robot cannot see the stick from every position and direction. Similarly, because of the robot's limited sensing capabilities, subtask (T1) was replaced by safe-wandering and subtask (T2) was refined to 'when an object nearby is sensed examine it and grasp it if it is a ball'. Notice that subtask 'safe wandering' is used for two purposes (to find a ball or the stick). The operating conditions of the corresponding controllers arose naturally as (T2) – an object should be nearby, (T3.2) – the stick should be detected, (T4) – the stick should be in front of the robot, and (T1,T3.1) – no condition. Since the behaviour of the robot must differ before and after locating a ball, an additional feature indicating when a ball was held was supplied. As the robot's gripper is equipped with an 'object-presence' sensor the 'the ball is held" feature was easy to implement. If there had not been such a sensor then this feature still could have been implemented as a switching-feature: the value of the feature would be `'on'` if the robot used the grasping behaviour, and hence not the hitting behaviour. An 'unstuck' subtask and corresponding controller were also included since the robot sometimes got stuck. Of course yet another feature is included for the detection of "goal-states". The corresponding feature indicates when the stick was hit by the ball. This feature's value is `'on'` iff the gripper is half-closed but the object presence sensor does not give a signal. Because of the implementation of the grasping module (the gripper was closed only after the grasping module was executed) this implementation of the "stick has been hit by the ball" feature was satisfactory for our purposes, although sometimes the ball slipped out from the gripper in which case the feature turned 'on' even though the robot did not actually reach the goal. Fortunately this situation did not happen too often and thus did not affect learning.

The resulting list of modules and features is shown in Table 2. The controllers work as intended, some fine details are discussed here (for more complete description see [8]). For example, the observation process was switched off until the controller of `Module 3` was working so as the complexity of the module-level decision problem is reduced. The dynamics of the controller associated with `Module 1` were based on the maximization of a function which depended on the proximity of objects and the speed of both motors[3]. If there were no obstacles near the robot this module made the robot go forward. This controller could thus serve as one for exploring the environment. `Module 2` was applicable only if the stick was in the viewing-angle of the robot, which could be detected in an unambiguous way because the only black thing that could get into the view of the robot was the stick. The range of allowed behaviour associated with this module was implemented as a proportional controller which drove the robot in such a way that the angle difference between the direction of motion and line of sight to the stick was reduced. The behaviour associated with `Module 3` was applicable only if there was an object next to the robot, which was defined as a function of the immediate values of IR-sensors. The associated behaviour was the following: the robot turned to a direction which brought it to point directly at the object then the gripper was lowered. The "hit the stick" module (`Module 4`) lowers the gripper which under appropriate conditions result in that the ball jumps out of the gripper resulting in the goal state. `Module 5` was created to handle stuck situations. This module lets the robot go backward and is applicable if the robot has not been able to move the wheels into the desired position for a while. This condition is a typical time-window based feature.

Simple case-analysis shows that there is no switching controller that would reach the goal with complete certainty (in the worst-case, the robot could return accidentally to state "10000000" from any state when the goal feature was `'off'`),

---

[3]Modules are numbered by the identification number of their features.

FNo	'on'	Behaviour
1	always	explore while avoiding obstacles
2	if the stick is in the viewing angle	go to the stick
3	if an object is near	examine the object grasp it if it is a ball
4	if the stick is near	hit the stick
5	if the robot is stuck	go backward
6	if the ball is grasped	-
7	if the stick is hit with the ball	-

Table 2: **Description of the features and the modules.** 'FNo.' means 'Feature No.', in the column labelled by 'on' the conditions under which the respective feature's value is `'on'` are listed.

so that an almost-sure switching strategy should always exist. On the other hand, it is clear that a switching strategy which eventually attains the target does exist.

3.3 Details of learning

A dense cost-structure was applied: the cost of using each behaviour was one except when the goal was reached, whose cost was set to zero. Costs were discounted at a rate of $\gamma = 0.99$. Note that from time to time the robot by chance became stuck (the robot's 'stuck feature' was `'on'`), and the robot tried to execute a module which could not change the value of the feature-vector. This meant that the robot did not have a second option to try another module since by definition the robot could only make decisions if the feature-representation changed. As a result the robot could sometimes get stuck in a "perpetual" or so-called "jammed" state. To prevent this happening we built in an additional rule which was to stop and reinitialize the robot when it got stuck and could not unjam itself after 50 sensory measurements. A cost equivalent to the cost of never reaching the goal, i.e., a cost of $\frac{1}{1-\gamma}$ (= 100) was then communicated to the robot, which mimicked in effect that such actions virtually last forever.

Experiments were fully automated and organized in trials. Each trial run lasted until the robot reached the goal or the number of decisions exceeded 150 (a number that was determined experimentally), or until the robot became jammed. The 'stick was hit' event was registered by checking the state of the gripper (see also the description of `Feature 7`).

During learning the Boltzmann-exploration strategy was employed where the temperature was reduced by $T_{t+1} = 0.999\,T_t$ uniformly for all states [2]. During the experiments the cumulative number of successful trials were measured and compared to the total number of trials done so far, together with the average number of decisions made in a trial.

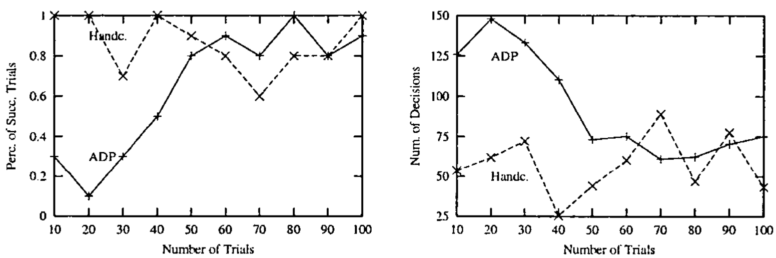

Figure 2: **Learning curves.** In the first graph the percentage of successful trials out of ten are shown as a function of the number of trials. In the second graph the number of decisions taken by the robot and averaged over ten trials are both shown, as well as a function of the number of learning trials.

3.4 Results

Two sets of experiments were conducted. The first set was performed to check the validity of the module based approach, while the second was carried out to compare different RL algorithms. In the first set the starting exploration parameter T_0 was set to 100 and the experiment lasted for 100 trials. These values were chosen in such a way that the robot could learn a good switching policy, the results of these experiments being shown in Figure 2.

One might conclude from the left subgraph which shows the percentage of task completions in different stages of learning that the robot could solve the task after 50 trials fairly well. Late fluctuations were attributable to unsuccessful ball searches: as the robot could not see the balls if they were far from it, the robot had to explore to find one and the exploration sometimes took more than 150 decisions, yielding trials which were categorized as being failures. The evaluation of behaviour-coordination is also observed in the second subgraph, which shows the number of decisions per trial as a function of time. The reason for later fluctuations is again due to a ticklish ball search. The performance of a handcrafted switching policy is shown on the graphs as well. As can be seen the differences between the respective performances of the handcrafted and learnt switching functions are eventually negligible. In order, to get a more precise evaluation of the differences the average number of steps to reach the goal were computed for both switchings over 300 trials, together with their standard deviations. The averages were 46.61 and 48.37 for the learnt and the handcrafted switching functions, respectively, with nearly equal std-s of 34.78 and 34.82, respectively.

Theoretically, the total number of states is $2^7 = 128$, but as learning concentrates on feature-configurations that really occur this number transpires to be just 25 here. It was observed that the learnt policy was always consistent with a set of handcrafted rules, but in certain cases the learnt rules are more refined than their handcrafted counterparts. For example, the robot learnt to exploit the fact that the arena was not completely level and as a result balls were biased towards the

Method	$T_0 = 100$	$T_0 = 50$	$T_0 = 25$	$T_0 = 0$
ADP	(61;19;6)	(52;20;4)	(45;12;5)	*(44;16;4)*
ARTDP	36	29	50	24
RTQL-1	53	69	47	66
RTQL-2	71	65	63	73
RTQL-3	(83;1;2)	(79;26:3)	(65;24;4)	*(61;14;2)*

Table 3: **Regret.** The table shows the number of unsuccessful trials among the first 100 trials. The entries with three number in them show cases when more than one experiment was run. In those entries the first number shows the average of the number of unsuccessful trials, the second is the standard deviation while the third is the number of experiments run.

stick and as a result if the robot did not hold a ball but could see the stick it moved towards the stick.

In the rest of the experiments we compared two versions of ARTDP and three versions of *real-time Q-learning* (RTQL). The two variants of ARTDP which we call ADP, and "ARTDP", corresponding to the cases when in the inner loop of ARTDP the optimal value function associated with the actual estimated model (transition probabilities and immediate cost) is computed and when only the estimate of the value of the actual state is updated. Note that due to the small number of states and module-based time discretization even ADP could be run in real-time. But variants of RTQL differ in the choice of the learning-rate's time-dependence. RTQL-1 refers to the choice of the so-called *search-then-converge* method, where $\alpha_k(s,a) = \frac{50}{100+n_k(s,a)}$, $n_k(s,a)$ being the number of times the event $(s,a) = (s_t, a_t)$ happened before time k plus one (the parameters 50 and 100 were determined experimentally as being the best choices). In the other two cases (the corresponding algorithms were denoted by RTQL-2 and RTQL-3 respectively) constant learning rates (0.1 and 0.25, respectively) were utilized.

The online performances of the algorithms were measured as the cumulative number of unsuccessful trials, i.e., the regret. The regret R_t at time t is the difference between the performance of an optimal agent (robot) and that of the learning agent accumulated up to trial t, i.e., it is the price of learning up to time t. A comparison of the different algorithms with different exploration ratios is given in Table 3. All algorithms were examined with all the four different exploration parameters since the same exploration rate may well result in different regrets for different algorithms, as was also confirmed in the experiments.

First note that in order to evaluate statistically the differences observed for different exploration strategies much more experiments would be needed but running these experiments would require an enormous amount of time (approximately 40 days) and have not been performed yet. Thus we performed the following procedure: Based on the first runs with every exploration-parameter and algorithm the algorithms that seemed to perform the best were selected (these were the ADP and the RTQL-3 algorithms) and some more experiments were carried out with these.

The results of these experiments (15 more for the ADP and 7 more for the RTQL-3) indicated that the difference between the performances of the RTQL-3 and ADP is significant at the level $p = 0.05$ (Student's t-test was applied for testing this).

We have also tested another exploration strategy which Thrun found the best among several undirected methods[4] [21]. These runs reinforced our previous findings that estimating a model (i.e., running ADP or ARTDP instead of Q-learning) could reduce the regret rate by as much as 40%.

4 Related Work

There are two main research-tracks that influenced our work. The first was the introduction of features in RL. Learning while using features were studied by Tsitsiklis and Van Roy to deal with large finite state spaces, and also to deal with infinite state spaces [22]. Issues of learning in partially observable environments have been discussed by Singh et al. [16].

The work of Connell and Mahadevan complements ours in that they set-up subtasks to be learned by RL and fixed the switching controller [13].

Asada et al. considered many aspects of mobile robot learning. They applied a vision-based state-estimation approach and defined "macro-actions" similar to our controllers [1]. In one of their papers they describe a goal-shooting problem in which a mobile robot shot a goal while avoiding another robot [24]. First the robot learned two behaviours separately: the "shot" and "avoid" behaviours. Then, the two behaviours were synthetised by a handcrafted rule and later this rule was refined via RL. The learnt action-values of the two behaviours were reused in the learning process while the combination of rules took place at the level of state variables.

Matarić considered a multi-robot learning task where each robot had the same set of behaviours and features [14]. Just as in our case, her goal was to learn a good switching function by RL. She considered the case when each of the robots learned separately and the ultimate goal was that learning should lead to a good collective behaviour, i.e., she concentrated mainly on the more involved multi-agent perspective of learning. In contrast to her work, we followed a more engineer-like approach when we suggested designing the modules based on well-articulated and simple principles and contrary to her findings it was discovered that RL can indeed work well at the modular level.

In the AI community there is an interesting approach to mobile robot control called Behaviour-Based Artificial Intelligence in which "competence" modules or behaviours have been proposed as the building blocks of "creatures" [12, 4]. The decision-making procedure is, on the other hand, usually quite different from ours.

The technique proposed here was also motivated by our earlier experiences with a value-estimation based algorithm given in the form of "activation spreading" [20]. In this work activation spread out along the edges of a dynamically varying graph,

---

[4] An exploration strategy is called undirected when the exploration does not depend on the number of visits to the state-action pairs.

where the nodes represented state transitions called triplets. Later the algorithm was extended so that useful subgoals could be found by learning [6, 7]. In the future we plan to extend the present module-based learning system with this kind of generalization capability. Such an extension may in turn allow the learning of a hierarchical organization of modules.

5 Summary and Conclusions

In this article module-based reinforcement learning was proposed to solve the coordination of multiple "behaviours" or controllers. The extended features served as the basis of time- and space discretization as well as the operating conditions of the modules. The construction principles of the modules were: decompose the problem into subtasks; for each subtask design controllers and the controllers' operating conditions; check if the problem could be solved by the controllers under the operating and observability conditions, add additional features or modules if necessary, set-up the reinforcement function and learn a switching function from experience.

The idea of our approach was that a partially observable decision problem could be usually transformed into a completely observable one if appropriate features (filters) and controllers were employed. Of course, some *a priori* knowledge of the task and robot is always required to find those features and controllers. It was argued that RL could work well even if the resulting problem was only an ϵ-stationary Markovian. The design principles were applied to a real-life robot learning problem and several RL-algorithms were compared in practice. We found that estimating the model and solving the optimality equation at each step (which could be done owing to the economic, feature-based time-discretization) yielded the best results. The robot learned the task after 700 decisions, which usually took less than 15 minutes in real-time. We conjecture that using a rough initial model good initial solutions could be computed off-line which could further decrease the time required to learn the optimal solution for the task.

The main difference between earlier works and our approach here is that we have established principles for the design modules and found that our subsequent design and simple RL worked spendidly. Plans for future research include extending the method via module learning and also the theoretical investigation of ϵ-stationary Markovian decision problems using the techniques developed in [10].

Acknowledgements

The authors would like to thank Zoltán Gábor for his efforts of building the experimental environment. This work was supported by the CSEM Switzerland, OTKA Grants No. F20132 and T017110, Soros Foundation No. 247/2/7011.

References

[1] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.

[2] A. Barto, S. J. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 1(72):81–138, 1995.

[3] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

[4] R. Brooks. Elephants don't play chess. In *Designing Autonomous Agents.* Bradford-MIT Press, 1991.

[5] T. Jaakkola, M. Jordan, and S. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, November 1994.

[6] Z. Kalmár, C. Szepesvári, and A. Lőrincz. Generalization in an autonomous agent. In *Proc. of IEEE WCCI ICNN'94*, volume 3, pages 1815–1817, Orlando, Florida, June 1994. IEEE Inc.

[7] Z. Kalmár, C. Szepesvári, and A. Lőrincz. Generalized dynamic concept model as a route to construct adaptive autonomous agents. *Neural Network World*, 5:353–360, 1995.

[8] Z. Kalmár, C. Szepesvári, and A. Lőrincz. Module based reinforcement learning: Experiments with a real robot. *Machine Learning*, 31:55–85, 1998. joint special issue on "Learning Robots" with the J. of Autonomous Robots;.

[9] M. Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Department of Computer Science, Brown University, February 1996. Also Technical Report CS-96-09.

[10] M. Littman and C. Szepesvári. A Generalized Reinforcement Learning Model: Convergence and applications. In *Int. Conf. on Machine Learning*, pages 310–318, 1996.

[11] M. L. Littman, A. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, San Francisco, CA, 1995. Morgan Kaufmann.

[12] P. Maes. A bottom-up mechanism for behavior selection in an artificial creature. In J. Meyer and S. Wilson, editors, *Proc. of the First International Conference on Simulation of Adaptive Behavior.* MIT Press, 1991.

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

[14] M. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4, 1997.

[15] S. Ross. *Applied Probability Models with Optimization Applications.* Holden Day, San Francisco, California, 1970.

[16] S. Singh, T. Jaakkola, and M. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proc. of the Eleventh Machine Learning Conference*, pages pp. 284–292, 1995.

[17] R. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[18] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, 8, 1996.

[19] C. Szepesvári and M. Littman. A unified analysis of value-function-based reinforcement-learning algorithms. *Neural Computation*, 1997. submitted.

[20] C. Szepesvári and A. Lőrincz. Behavior of an adaptive self-organizing autonomous agent working with cues and competing concepts. *Adaptive Behavior*, 2(2):131–160, 1994.

[21] S. Thrun. *The role of exploration in learning control.* Van Nostrand Rheinhold, Florence KY, 1992.

[22] J. Tsitsiklis and B. Van Roy. An analysis of temporal difference learning with function approximation. Technical Report LIDS-P-2322, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1995.

[23] J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[24] E. Uchibe, M. Asada, and K. Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robot and Sytems*, pages 1329–1336, 1996.

[25] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 3(8):279–292, 1992.

Received May, 1999