Reinforcement Learning: Theory and Practice

Csaba Szepesvári
Associative Computing Ltd.
Budapest 1121, Konkoly Thege M. út 29–33
e-mail: szepes@mindmaker.kfkipark.hu

Abstract

We consider reinforcement learning methods for the solution of complex sequential optimization problems. In particular, the soundness of two methods proposed for the solution of partially observable problems will be shown. The first method suggests a state-estimation scheme and requires mild *a priori* knowledge, while the second method assumes that a significant amount of abstract knowledge is available about the decision problem and uses this knowledge to setup a macro-hierarchy to turn the partially observable problem into another one which can already be handled using methods worked out for observable problems. This second method is also illustrated with some experiments on a real-robot.

1 INTRODUCTION

Reinforcement learning (RL) concerns the problem of learning to control a process such that a long-term performance criterion is optimized which is defined in terms of some observable, local (or immediate) reinforcement values. Most RL algorithms assume that the optimization problem can be transformed into a Markovian Decision Problem (MDP) and thus enjoy a firm theoretical background which can be ascribed to the fact that Bellman's principle of optimality holds in MDPs. In order to explain this remarkable principle let us first define MDPs. A Markovian Decision Process is completely specified by its state, a set of possible actions for each state, a (sometimes stochastic) dynamics which describes the evaluation of states under the execution of actions, the immediate rewards that we encounter when a state-transition occurs and the way immediate rewards should be glued together to yield the long-term reward (this can be as simple as adding together the immediate rewards). In mathematical terms, an MDP M can be represented by a tuple $(X, A, \mathcal{A}, p, r, \mathcal{Q})$, where X is the state-space (this is usually a finite set); A is the action-space (typically another finite set); $\mathcal{A} : X \to 2^A$ (2^A denotes the power set of A) determines the set of admissible actions for each state which must be non-empty[1]; $p : X \times A \times X \to [0,1]$ is a transition probability function, $p(x, a, y)$ being the probability that the next state is y given that action a is executed in state x; $r : X \times A \to \mathbb{R}$ is the immediate reward function and $\mathcal{Q}$ is a rule that prescribes the way immediate rewards determine the long-term outcome of an action sequence.[2] Bellman's principle of optimality claims that the optimal action in a state is the one whose long-term return is maximal if the actions from the next step are always chosen optimally. A MDP can be thought of as a framework for optimal control: the controls are identified by the actions and the (stochastic) dynamics is described by the transition probabilities. When a system that can be modeled by an MDP is controlled the state of the system can be either observable or unobservable (hidden) to the controller. In the unobservable case the controller observes only a (typically not one-to-one) function of the state and should control the system using this restricted information only. Most basic RL algorithms are developed for finite-size (and even small), observable MDPs since these are mathematically very attractive.

In contrast, most real-world applications can only be represented by models in which observability does not hold, nor is the size of the model small and sometimes the model is not even finite. The following list shows common situations where RL has already found good applications or may found applications in the future.

[1] Unless otherwise noted, for most of the developments in this paper we will assume that $\mathcal{A}(x) = A$ holds for all x in which case $\mathcal{A}$ will not be mentioned explicitly.

[2] Specifically, $\mathcal{Q} : \mathbb{R}^X \times \mathbb{R}^{X \times A} \to \mathbb{R}^{X \times A}$. For example, if the rule is to take the expected value of the sum of the immediate rewards then $[\mathcal{Q}(v, r)](x, a) = r(x, a) + \sum_{y \in X} p(x, a, y) v(y)$: here the value $v(y)$ of the generic function v represents the long-term reward when the decision process is started in y. Usually, $\mathcal{Q}$ is not listed in the definition of MDPs as it might be fixed otherwise.

Simulation setup (games, combinatorial optimization with guided local search). Here the dynamics of the controlled process (for games, dynamics are the rules of the game; for combinatorial optimization, dynamics are the rules describing the possible next configurations of a given configuration) are known, the state information is observable (for board games this is usually the state of the board, for combinatorial optimization it is the actual configuration in the search space). For games a simple choice for the immediate reinforcement values could be: the reinforcement is $+1$ when the optimizer wins, -1 when it looses the game and zero otherwise, the long-term reward being the sum of the immediate rewards. Learning to optimize the long-term reward means then the identification of a mapping from the states of the game to the set of possible actions (moves) that yields a dynamical behavior which results in a win whenever it is possible and in a losing only when there is no way to win or to make a draw. Similarly, in the optimization setup, assuming that we seek the maximum of f, one might choose a reward equal to $f(x)$ when x is a configuration which is a "local maximum" of f, and 0 otherwise. The learning process then identifies a strategy to guide the local search such that the global maximum is reached from any given initial state. For applications of RL in games and combinatorial optimization see [17, 20, 14, 2, 1].

Applications of RL in the simulation setup are interesting since these problems are usually too big for computing their solutions explicitly, i.e., the state or actions spaces of these problems have billions of elements.

Autonomous robots. Imagine an indoor robot with a set of sensors that bring information to the robot about its immediate neighborhood. Assume that the robot is assigned a specific task, e.g., hoovering the floors. The robots state is determined by the actual configuration of the building(!) including e.g. the intentions and "states" of people working in the building, or the position and orientation of the robot within the building. Clearly, most of the details of such a state-description would be irrelevant from the point of view of the task to be performed by the robot. The robots actions can be as low-level as the application of a unit current to one of its DC-motors. The immediate-reward function could be the amount of garbage hoovered by the robot at the actual moment in time, while the long-term reward could be the average amount of garbage hoovered by the robot during its life-time. For applications of RL in robot-learning domains see e.g. [6, 8, 9, 5].

Here the states are unobservable, the state-space and action-spaces are infinite, time is continuous, the precise form of the state-dynamics and observation-dynamics (this is the way a state-sequence is transformed to an elementary observation, e.g., sensor-value) are unknown.

Process optimization or optimal control (e.g. [7]). Imagine a factory that produces goods. The optimizer can control the production process through various means. The immediate reinforcement value is the immediate income minus the immediate costs. The long-term value could be the expected discounted sum of the immediate values. This would actually represent the present value of future incomes assuming that the inflation ratio does not change over time. Another example comes from the optimization of continuous control processes, e.g. to maximize the average yield of a bioreactor. In this class of examples the state and action spaces are typically non-countable, states are unobservable, the dynamics is partially known. Time can be either discrete or continuous.

Interestingly, the nice properties of RL methods seem to transfer to these more complicated setups. Of course, in these applications the basic methods are extended with some tools. Some of them are listed next.

Factored representations (see e.g. [16]). Often *a priori* knowledge is available about the independence of one part of the description of the state from another part. The state of an object distant from the robot does not (usually) change as a result of some action performed by the robot. Factored representations can be exploited e.g. to reduce the memory-requirement of the storage of a representation of the dynamics.

Function approximators (e.g. [15, 19]). Central to RL algorithms are value-functions. A value function is a function that renders real-numbers (values) to states. The number associated with a state represents the long-term value of a policy or the best possible long-term value that can be achieved from the given state. In this latter case the value-function is called the *optimal value function* and is denoted by v^* ($v^* : X \to \mathbb{R}$, where X is the set of states). Most RL algorithms operate by modifying a value function to incorporate some new information which is usually given in the form of a transition (x_t, a_t, y_t, r_t). Here x_t is the state (of the system) when action a_t is applied to the system, y_t is the resulting state, and r_t is the immediate reinforcement associated with the transition (x_t, a_t, y_t). Bellman's principle of optimality yields that the knowledge of the optimal value function and the dynamics are sufficient to find an optimal policy or action-selection mechanism [4]. Unfortunately, since the state-space X is usually infinite, value functions cannot be stored (represented) in a tabulated form – one must use function approximators. Unfortunately, most function approximation algorithms are suited for the estimation of a fixed function. In the case of RL, however, the values of the function to be estimated (i.e. the optimal long-term values) are not available for direct measurement, but they are usually estimated on the basis of the *current* (and thus imprecise) estimate of the value-function. This brings in new complications to the estimation procedure. As to day, linear function approximators of the form $\theta^T\phi(x)$, where $\phi : X \to \mathbb{R}^n$ is a fixed function (ϕ is called the feature-function) are known to work with RL algorithms [19]. However, often, memory-based neural networks are used in RL algorithm with pretty good success [10].

Observers (e.g. [5]). Control theorists use filters or observers to estimate the state if it is unobservable. These observers usually require knowledge of the dynamics and exploit the vector-space properties of the state-space. In RL there is usually no natural vector-space representation of the state-space (although, one could probably work with "features" or embed the state-space into a vector-space in another way). Nevertheless, exploiting some other *a priori* knowledge of the task to be solved, good observers which recover the part of the state which is important in solving the task can be designed by hand. Then RL is used on top of the output of the observers as the output of these were the true states. This method is routinely used in autonomous robot domains. Automatic construction of observers, however, is a non-trivial open question. A specific case of this problem will be treated in Section 2.

Macros (see e.g. [18, 13, 5, 11, 3]. Macros are local-controllers (or policies) usually associated with performing some *subtask* of the original task. The attribute "local" denotes that the associated controllers are assumed to perform well only under some restricted conditions. These conditions are thought to be part of the definition of macros. Macros are interesting as they can provide means to both state and time-abstractions if macros can be launched only under specific circumstances (called the precondition of the macro) and once they are launched they continue their working until another condition is met (this may indicate that the subgoal associated with the macro is satisfied or that some specific emergency situation occurred which requires the interruption of the macro or that another macro became recently available for execution). State and time-abstractions yield a "higher-level" description of the task and thus can promote knowledge transfer from one subtask to another. Macros have been shown to be invaluable for speed-up planning and learning [13, 12]. Also macro-hierarchies are being built which are hoped to boost up the learning/planning processes even more substantially. In Section 3 we show that a certain robust design method suggested in [5] is sound.

2 THE PREDICTABILITY OF OBSERVATIONS AND THE RECOVERY OF OPTIMAL POLICIES

The following folklore is widely believed among RL people: choose a representation that makes things predictable (in terms of the chosen representation). Then learn a model on top of the representation and use it in your decision making strategy as if it were the true model. You will end up with a reasonable policy.

In this section we show that this folklore is correct in a certain restricted setting.

First, we need some definitions. Remember that a POMDP is a 6-tuple, $M = (X, A, p, r, g, Y)$, where (X, A, p, r) is a MDP and $g : X \to Y$ is an *observation function* and Y is the *observation space*.[3]

DEFINITION 2.1 *Given a POMDP M, a function $\chi : Y \times A \times \ldots \times Y \to E$ is called a* finite-memory state-estimator.

A state-estimator χ gives us the estimate of the current state x_t via $h_t = \chi(y_t, a_{t-1}, y_{t-1}, a_{t-2}, \ldots, y_{t-n})$.

In terms of χ we can only check if it enables to predict future "states", based on the present state-estimate and action:

DEFINITION 2.2 *Let χ be a finite-memory state-estimator. We say that χ recovers the Markov-property (or simply, χ is Markov) if*

$$\begin{aligned}
&\mathrm{Prob}(\chi(y_t, a_{t-1}, y_{t-1}, \ldots, y_{t-n}) = h' \,|\, a_{t-1} = a, \\
&\qquad \chi(y_{t-1}, a_{t-2}, y_{t-2}, \ldots, y_{t-n-1}) = h, \mathcal{H}_{t-1}) = \\
&\mathrm{Prob}(\chi(y_t, a_{t-1}, y_{t-1}, \ldots, y_{t-n}) = h' \,|\, a_{t-1} = a, \\
&\qquad \chi(y_{t-1}, a_{t-2}, y_{t-2}, \ldots, y_{t-n-1}) = h),
\end{aligned} \tag{1}$$

and similarly for the the reward stream; for all $r \in \mathbb{R}$ we require

$$\begin{aligned}
&\mathrm{Prob}(r_t \le r \,|\, a_{t-1} = a, \chi(y_{t-1}, a_{t-2}, y_{t-2}, \ldots, y_{t-n-1}) = h, \mathcal{H}_{t-1}) = \\
&= \qquad \mathrm{Prob}(r_t \le r \,|\, a_{t-1} = a, \chi(y_{t-1}, a_{t-2}, y_{t-2}, \ldots, y_{t-n-1}) = h).
\end{aligned} \tag{2}$$

[3]More generally, g could be a stochastic mapping. For simplicity, however, we do not consider stochastic observations here.

Here $\mathcal{H}_{t-1}$ is the history-space defined by the variables $y_{t-1}, a_{t-2}, \ldots, y_0$ (actually, $\mathcal{H}_{t-1}$ should be the σ-algebra generated by the random variables $y_{t-1}, a_{t-2}, \ldots, y_0$).

That is, if χ recovers the Markov-property then the next value of h_t can be predicted on the basis of its current value and the action to be executed. Armed with this definition, the original question is reformulated in the following way: can we use h_t as if it were the true state and solve for the optimal policy provided that χ is Markov? Before answering this question, however, let us first consider the problem of testing if χ recovers the Markov-property.

If we know that a finite-length memory is sufficient to predict the next state (i.e., if there exists $m > 0$ such that $\mathrm{Prob}(x_t | a_{t-1}, h_{t-1}, a_{t-2}, \ldots, h_{t-m}) = \mathrm{Prob}(x_t | a_{t-1}, \mathcal{H}_{t-1})$) then (without the loss of generality assuming $n \leq m$) (1) is equivalent to

$$\begin{aligned} \mathrm{Prob}(h_t = h' \mid a_{t-1} = a, h_{t-1} = h, a_{t-2}, h_{t-2}, \ldots, h_{t-m}) = \\ \mathrm{Prob}(h_t = h' \mid a_{t-1} = a, h_{t-1} = h) \end{aligned} \tag{3}$$

since

$$\begin{aligned} & \mathrm{Prob}(h_t = h' \mid a_{t-1} = a, h_{t-1}, \mathcal{H}_{t-1}) \\ & \quad = \mathrm{Prob}(\chi(y_t, a_{t-1}, y_{t-1}, \ldots, y_{t-n}) = h' \mid a_{t-1} = a, h_{t-1}, \mathcal{H}_{t-1}) \\ & \quad = \mathrm{Prob}(\chi(g(x_t), a_{t-1}, y_{t-1}, \ldots, y_{t-n}) = h' \mid a_{t-1} = a, h_{t-1}, \mathcal{H}_{t-1}) \\ & \quad = \mathrm{Prob}(\chi(g(x_t), a_{t-1}, y_{t-1}, \ldots, y_{t-n}) = h' \mid a_{t-1} = a, h_{t-1} = h, a_{t-2}, h_{t-2}, \ldots, h_{t-m}) \\ & \quad = \mathrm{Prob}(h_t = h' \mid a_{t-1} = a, h_{t-1} = h, a_{t-2}, h_{t-2}, \ldots, h_{t-m}). \end{aligned}$$

That is, the property under consideration of χ can be checked by estimating the probabilities in (3) from the observations and using some tests (e.g. a t-test or a test based on Hoeffding's inequality). A similar test can be used for the rewards.

If n, the length of the memory that suffices to predict the future is not known a priori then one can test successively each $n > 0$. Namely, given a positive number, $m > 0$ one should check if

$$\begin{aligned} \mathrm{Prob}(h_t = h' \mid a_{t-1} = a, h_{t-1} = h, a_{t-2}, h_{t-2}, \ldots, h_{t-m}) = \\ \mathrm{Prob}(h_t = h' \mid a_{t-1} = a, h_{t-1} = h) \end{aligned}$$

and

$$\begin{aligned} \mathrm{Prob}(r_t < r \mid a_{t-1} = a, h_{t-1} = h, a_{t-2}, h_{t-2}, \ldots, h_{t-m}) = \\ \mathrm{Prob}(r_t < r \mid a_{t-1} = a, h_{t-1} = h) \end{aligned}$$

hold. If the test is passed, one increases m. One can choose the sample-sizes in the tests such that if n, the history size is smaller than sufficient then the repeated application of the tests will fail when m is large enough. Then the history size (n) should be increased. In this way, the correct order of the history size can be estimated with large probability.[4]

For deterministic problems, (3) reduces to checking if there exists some function $\hat{\delta} : X \times A \to X$ s.t. $\hat{\delta}(h_t, a_t) = h_{t+1}$, or

$$\hat{\delta}(\chi(y_t, a_{t-1}, y_{t-1}, \ldots, y_{t-n}), a_t) = \chi(y_{t+1}, a_t, y_t, \ldots, y_{t+1-n}). \tag{4}$$

If the rewards are also deterministic then (2) reduces to checking if

$$(h_{t-1}, a_{t-1}) \mapsto r_t \tag{5}$$

is a function.[5] If this function exists it will be denoted by $\hat{r}$.

Now, let us return to the original question. For simplicity, we will consider deterministic problems, when the transition probability function, p is represented by some transition function (or dynamics) $\delta : X \times A \to X$.[6]

[4] Of course, this method is not very practical, but from a theoretical point of view it is still interesting. Also, the method could be made more practical e.g. if a factored representation of the state-space is available.

[5] Here we have exploited the assumption that the reward associated with a transition (x, a, y) does not depend on y. However, this assumption could be easily removed.

[6] The identification of transition functions and transition probabilities goes as follows: to every δ we may assign a transition probability function p_δ such that $p_\delta(x, a, x') = 1$ if and only if $\delta(x, a) = x'$.

First, we need some technicalities. Assume (for simplicity) that the MDP is connected in the sense that for all $x \in X$ there exists a state $x' \in X$ and an action $a' \in A$ such that $\delta(x', a') = x$.[7]

Now, let us extend the dynamics δ to action sequences:

$$\delta(x; a_1 \dots a_n) \stackrel{\text{def}}{=} \delta(\dots \delta(\delta(x, a_1), a_2) \dots, a_n). \tag{6}$$

Then one can easily show (e.g. by induction) that for all x there exists a state x' and an action sequence $a'_1, \dots, a'_n$ such that $\delta(x'; a'_1 \dots a'_n) = x$.

We will find it convenient to define

$$S(x; a_1 \dots a_n) = (g(\delta(x; a_1 \dots a_n)), a_n, g(\delta(x; a_1 \dots a_{n-1})), a_{n-1}, \dots, a_1, g(x)) \tag{7}$$

to represent the *observed history* of the process when the action sequence $a_1, \dots, a_n$ is executed starting from state x.[8]

Now, let us define $\tau : E \to 2^X$ (2^X denotes the power-set of X) by

$$\tau(h) = \{x \in X \mid \exists x', a'_1 \dots a'_n : \chi(S(x'; a'_1 \dots a'_n)) = h, \delta(x'; a'_1 \dots a'_n) = x\}, \tag{8}$$

i.e., $\tau(h)$ denotes the states of X at which the state-estimate h could have been observed.

The Markov property of χ can now be expressed as follows:

- for all x, x', $a_1, \dots, a_n$, $a'_1, \dots, a'_n$ and a,

$$\begin{aligned} \chi(S(x; a_1 \dots a_n)) = \chi(S(x'; a'_1 \dots a'_n)) \quad \Rightarrow& \\ \chi(S(\delta(x; a_1), a_2 \dots a_n a)) \quad =& \quad \chi(S(\delta(x'; a'_1), a'_2 \dots a'_n a)) \end{aligned}$$

  and

- for all x, x', $a_1, \dots, a_n$, $a'_1, \dots, a'_n$ and a,

$$\begin{aligned} \chi(S(x; a_1 \dots a_n)) = \chi(S(x'; a'_1 \dots a'_n)) \quad \Rightarrow& \\ r(\delta(x; a_1 \dots a_n), a) \quad =& \quad r(\delta(x'; a'_1 \dots a'_n), a). \end{aligned}$$

The maps $\hat{\delta}$ and $\hat{r}$ can be defined by

$$\hat{\delta}(\chi(S(x; a_1 \dots a_n)), a) = \chi(S(\delta(x; a_1), a_2 \dots a_n a)). \tag{9}$$
$$\hat{r}(\chi(S(x; a_1 \dots a_n)), a) = r(\delta(x; a_1 \dots a_n), a). \tag{10}$$

If a memory length of n is sufficient to predict the future then $E = Y \times (A \times Y)^{n-1}$ with

$$\chi(y_t, a_t, \dots, y_{t-n}) = (y_t, a_t, \dots, y_{t-n})$$

is Markov. The proof of this is trivial and is omitted. However, this is not the only one Markov state-estimator. Indeed, if χ is a Markov state-estimator and $h, h' \in E$ are such that for all $a \in A$, $\hat{\delta}(h, a) = \hat{\delta}(h', a)$, $\hat{r}(h, a) = \hat{r}(h', a)$ (i.e., if the outward connections of h and h' are identical) then h and h' can be "drawn together" to yield a new state-estimator. This new state-estimator will be Markov, too.

Definition 2.3 *The states $x', x'' \in X$ are called twin-states, if $A(x') = A(x'')$, and for all $a \in A(x')$, $r(x', a) = r(x'', a)$ and $\delta(x', a) = \delta(x'', a)$.*

The following proposition shows that the unification of twin-states does not change the optimal value function. For the sake of simplicity in the rest of the section we will restrict ourselves to the discounted optimality criterion.

Proposition 2.4 *If x', x'' are twin states then there exists an optimal policy π^* such that $\pi^*(x') = \pi^*(x'')$.*

[7] In a stochastic context the phrase corresponding to "connected" would be "communicating".

[8] In order χ to be totally consistent with the underlying observation stream we must also require that $h_t \mapsto y_t$ be a mapping, although this property will not be exploited through the following developments. When this mapping exists it could be denoted by $\hat{g}$. Also we may require χ to be self-consistent. This means that when $\hat{\delta}(h; a_1 \dots a_n) = \hat{\delta}(h'; a'_1 \dots a'_n)$ then also $\chi(S_{\hat{g},\hat{\delta}}(h; a_1 \dots a_n)) = \chi(S_{\hat{g},\hat{\delta}}(h'; a'_1 \dots a'_n))$ should hold, where $S_{\hat{g};\hat{\delta}}$ is the obvious generalization of S to the observation – state-transition mapping pair $(\hat{g}, \hat{\delta})$.

Proof: Let π be an optimal policy. Then, by Howard's policy improvement theorem, the greedy improvement of π (let us denote it by π^*) is also optimal. Further, $\pi^*(x') = \mathrm{argmax}_{a\in A(x')}(r(x',a) + \gamma v_\pi(\delta(x',a))) = \mathrm{argmax}_{a\in A(x'')}(r(x'',a)+\gamma v_\pi(\delta(x'',a))) = \pi^*(x'')$, where the second equality holds since x' and x'' are twin-states. Q.E.D.

The next proposition is the main result of this section and shows the importance of the Markov property of χ.

PROPOSITION 2.5 *Assume that χ is Markov. Then the optimal policy with respect to $\hat{M}$ induces an optimal policy in M.*

Proof: First, we show that for every state x, action sequence $a_1,\dots,a_n$ and stationary policy π of M there exists a policy $\hat{\pi}$ of $\hat{M}$ such that

$$v_\pi(\delta(x;a_1\dots a_n)) = v_{\hat{\pi}}(\chi(S(x;a_1\dots a_n)), \tag{11}$$

where v_π is the evaluation function associated with π in the original MDP, and $v_{\hat{\pi}}$ is the evaluation function associated with $\hat{\pi}$ in the MDP defined by $\hat{M} = (E, A, p_{\hat{\delta}}, \hat{r})$.

Let $x_0 = \delta(x;a_1\dots a_n)$ and $h_0 = \chi(S(x;a_1\dots a_n))$. Further, let $x_{i+1} = \delta(x_i;\pi(x_i))$ and $h_{i+1} = \hat{\delta}(h_i,\pi(x_i))$, $i \geq 0$. It is sufficient to prove that $r(x_i,\pi(x_i)) = \hat{r}(h_i,\pi(x_i))$, $i\geq 0$. By the Markov property of χ, $h_i = \chi(S(x_{i-n-1};b_{i-n-1}\dots b_{i-1}))$ and $x_i = \delta(x_{i-n-1};b_{i-n-1}\dots b_{i-1})$. Here b_t represents the action at time $-n\geq t$: $b_{-j} = a_{n-j+1}$, $j = 1,2,\dots,n$ and $b_j = \pi(x_j)$, $j\geq 0$. Similarly, $x_{-j} = \delta(x;a_1\dots a_{n-j+1})$, $j = 1,2,\dots,n$. Therefore, by the definition of $\hat{r}$ (see (10)), $\hat{r}(h_i,b_i) = \hat{r}(\chi(S(x_{i-n-1};b_{i-n-1}\dots b_{i-1})),b_i) == r(\delta(x_{i-n-1};b_{i-n-1}\dots b_{i-1}),b_i) = r(x_i,b_i)$, $i\geq 0$.

The reverse of the above statement, i.e., that for every state x and action sequence $a_1,\dots,a_n$, and every stationary policy $\hat{\pi}$ of $\hat{M}$ there exists a policy π of M such that (11) holds, can be proved in an entirely analogous manner (just now we should choose $b_i = \hat{\pi}(h_i)$). Now let's pick up an optimal policy $\hat{\pi}^*$ of $\hat{M}$ and some initial path $(x;a_1\dots a_n)$. Let π be the policy in M that satisfies (11) and let $h_0 = \chi(S(x;a_1\dots a_n))$. Then

$$\begin{aligned} v_{\hat{\pi}^*}(h_0) &= v_\pi(\delta(x;a_1\dots a_n)) \\ &\leq v^*(\delta(x;a_1\dots a_n)) \\ &= v_{\pi^*}(\delta(x;a_1\dots a_n)) \\ &= v_{\tilde{\pi}^*}(S(\chi(x;a_1\dots a_n))) \\ &= v_{\tilde{\pi}^*}(h_0), \end{aligned}$$

where π^* denotes an optimal policy in M and $\tilde{\pi}^*$ is the corresponding policy in $\hat{M}$. The above inequality cannot be strict since this would yield that $\hat{\pi}^*$ is sub-optimal in h_0. Therefore

$$v_\pi(\delta(x;a_1\dots a_n)) = v^*(\delta(x;a_1\dots a_n))$$

meaning that π is optimal in M at state $\delta(x;a_1\dots a_n)$.

Since M is assumed to be connected, each state of M can be written in the form of $\delta(x;a_1\dots a_n)$ with some x and $a_1\dots a_n$, meaning that for any initial state $y_0\in X$ if h_0 is consistent with y_0 then the actions returned by $\hat{\pi}^*$ will be optimal in M. (Of course, in practice one needs to wait for n steps before $\hat{M}$ and M become synchronized.) Q.E.D.

3 MACROS AS A ROBUST DESIGN METHOD FOR SOLVING POMDPS

Consider a POMDP $M = (X, A, \tilde{A}, p, r, g, Y)$, where $\tilde{A} : X \to 2^A$ determines the set of admissible actions. The following design was suggested in [5]: Choose an observer (ϕ, h, T, F), $\phi : T\times Y \to F$, $h : T\times Y\to F$, where the new observation f_t is given by the recursive equations $\theta_{t+1} = \phi(\theta_t, y_t)$, $f_t = h(\theta_t, y_t)$. Further, choose a set of macros $\Pi = \{(\pi_1, D_1, s_1),\dots,(\pi_n, D_n, s_n)\}$, where $\pi_i : D_i \to A$ is an admissible local policy, $D_i\subseteq F$, and $s_i : F\to\{0,1\}$ which returns zero outside of D_i ($s_i(f) = 0$ if $f\notin D_i$). D_i is called the domain of policy π_i and s_i is a function indicating that macro π_i is ready to be launched. We will denote by d_i the characteristic function of D_i.

In each time-step, there is a single macro that is being executed, called the active macro. The execution of the macro with index i means that at time step t, the action $\pi_i(f_t)$ is carried out. The following rule holds for the switching of macros. The active macro cannot be replaced by another one, unless one of the following conditions is satisfied:

1. The current macro finished its working, i.e., $f_t \notin D_i$. In this case those macros will be available for execution for which $s_i(f_t) = 1$.
2. $s_i(f_t) = 1$ for some macro, for which $s_i(f_{t-1}) = 0$ holds as well, i.e., a macro became available for execution in the last step. In this case the current macro and the macros that just became available will be available for execution.

Care must paid in the design process so that there always exists a macro available for execution.

PROPOSITION 3.1 *The above restriction on the macro-selection can be represented as another POMDP with action set* Π.

Proof: Consider the POMDP $\hat{M} = (X \times T \times F \times \underline{n}, \Pi, \tilde{\Pi}, \hat{p}, r, \hat{g}, F \times F \times \underline{n})$, where $\underline{n} = \{1, 2, \ldots, n\}$. Let us assign to each state $\langle x, \theta', f, j\rangle$ of this new POMDP the following symbols: $f' = h(\theta, g(x))$, $s_i' = s_i(f')$, $s_i = s_i(f)$ (with a slight abuse of notation) and $d_i = d_i(f')$. θ' of the current state is intended to represent the most recent value of θ used in the observer (ϕ, h, T, F) and f is used to cache the previous value of $h(\theta, y)$. The set of admissible actions (macros in this case) for state $\langle x, \theta', f, j\rangle$ consists of those policies π_i for which $(s_i' = 1 \,\&\, s_i = 0)$ or $(d_i = 1 \,\&\, i = j)$ or $(d_j = 0 \,\&\, s_i = 1)$. The non-zero elements of the transition probability function $\hat{p}$ are defined as follows:

$$\hat{p}(\langle x, \theta', f, j\rangle, \pi_i, \langle x', \theta'', f', j'\rangle) = p(x, \pi_i(h(\theta', g(x))), y), \tag{12}$$

where $\theta'' = \phi(\theta', g(x))$, $f' = h(\theta', g(x))$ and $j' = i$. The observation function $\hat{g}$ is defined by $\hat{g}(\langle x, \theta', f, j\rangle) = (f, f', j)$. Q.E.D. A semi-Markov [13] representation is also possible. In such a representation macros can be executed for more then one time-step. Therefore they must all provide two sets of conditions: terminating and start ones. In our case the terminating condition of a macro could be the following: a macro with index i continues to work until another macro with index j becomes available for execution and while it is available for execution. The start-condition of macro i is identical to the condition $s_i = 1$.[9] This semi-Markov representation shows that "time-abstraction" occurs here. However appealing this semi-Markov representation may be it is not really suitable for our purposes since macros would need to monitor the start conditions of other macros which is somewhat unnatural.

The previous proposition shows that using this method, macro hierarchies can be built up, i.e., the POMDP $\hat{M}$ of Proposition 3.1 can be taken as a starting point for another transformation with a different (higher level) macro set. That is, the use of macro actions can be viewed as a kind of transformation of POMDPs.

The purpose of these transformation is to arrive at a kind of "robust" POMDP for which a "traditional" RL method could work.

DEFINITION 3.2 *An MDP $M = (X, A, p, r)$ is said to be* robust *if the optimal solution of $(X, A, \hat{p}, r)$ is the optimal solution of M if p and $\hat{p}$ agree on their non-zero elements: $p(x, a, y) \neq 0$ if and only if $\hat{p}(x, a, y) \neq 0$. The set $\{(x, a, y) : p(x, a, y) \neq 0\}$ is called the support of p.*

Typical robust MDPs are the goal-oriented ones when the optimizer receives a non-zero terminating reward upon reaching a set of special (goal) states. The requirement of optimality in the above definition could be relaxed to "acceptable", enabling a larger set of robust MDPs. Before we give the main proposition of this section we need another definition.

DEFINITION 3.3 *Let us consider a goal-oriented POMDP $M = (X, A, p, r, g, Y)$, where $r : Y \times A \to \{0, 1\}$ (rewards depend only on the observations and actions) and assume that there exists a positive number ε such that for all (y, a, y'), either* $\mathrm{Prob}(y' \,|\, a, y, h) \geq \varepsilon$ *holds for all $t \geq 0$ and almost all history h, or* $\mathrm{Prob}(y' \,|\, a, y, h) = 0$ *holds for all $t \geq 0$ and almost all history h. Assume further that the MDP $(Y, A, \hat{p}, r)$ is robust, where $\hat{p}$ is any transition probability function which satisfies $\hat{p}(y, a, y') \geq \varepsilon$ if* $\mathrm{Prob}(y' \,|\, a, y, h) \geq \varepsilon$ *and $\hat{p}(y, a, y') = 0$ if* $\mathrm{Prob}(y' \,|\, a, y, h) = 0$ *and that the optimal solution $\hat{M}$ induces a solution of the original POMDP. Then, M is said to be* practically observable.

This definition and the next proposition reveal our wish to transfer POMDPs into a form whose observations can be treated as states without the loss of solutions.

[9]The condition that switching among macros is only possible when a macro which was *not* available in the previous step becomes available may seem too restricting. The purpose of this is to reduce the frequency of decision-situations. This condition could be removed without effecting any of the previous and subsequent results.

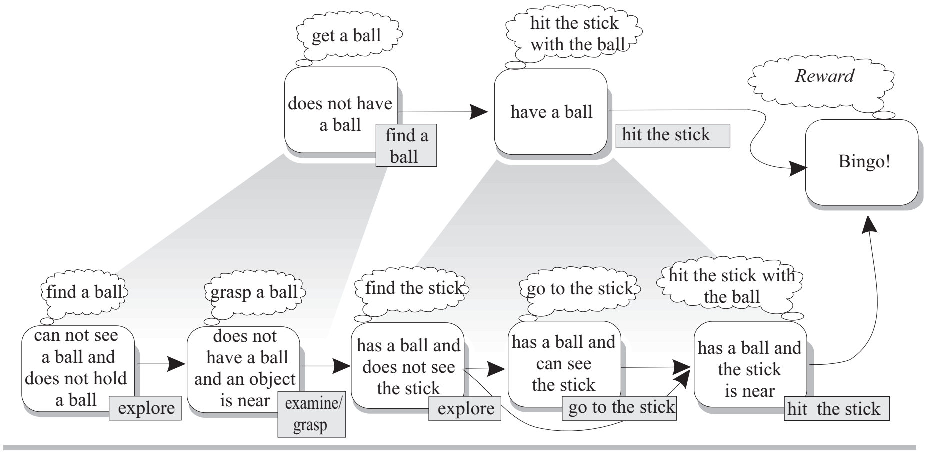

Figure 1: **Subtask decomposition example**.
The main task of hitting the stick with a ball is first broken up into two subproblems: get a ball and then hit the stick with the ball. Then these are again decomposed into smaller subtasks which can already be accomplished by simple controllers. The bubbles show the subtask, the rectangles with rounded corners show the conditions under which the solution of the given subtask is sensible and the shaded rectangles show the macro-actions. The arrows show the ideal flow of working, in practice other transitions are also possible and do exist. Note that the "explore" macro-action is used twice in the ideal flow of working. One particular maintenance (or rescue) subtask is not shown to preserve the clarity of the figure but this task is very important to satisfy the requirement that there always exists an admissible macro.

PROPOSITION 3.4 *Let us consider a practically observable POMDP, M. Then, the model-based RL algorithm, which "estimates" the transition probabilities $\hat{p}$ (as given in definition 3.3) by sample means and computes the optimal policy for the "estimated model" converges to a policy which solves the POMDP M, provided that every (y, a)-pair is visited infinitely often.*

Proof: The proof follows from the observation that if $p_t = (\xi_1 + \ldots + \xi_t)/t$ is the sample mean of $\xi_1, \ldots, \xi_t, \ldots$, where the random variables $\xi_i \in \{0, 1\}$ are conditionally independent, and $P_t = \mathrm{Prob}(\xi_t = 1)$ then $\liminf_{t\to\infty} p_t \geq \liminf_{t\to\infty} P_t$ and $\limsup_{t\to\infty} p_t \leq \limsup_{t\to\infty} P_t$. Therefore, after an initial period the support of the estimated transition probabilities will coincide with that of $\hat{p}$. Then, by the robustness of $\hat{M}$, the estimated optimal policy will be an optimal solution to $\hat{M}$ which (by assumption) will induce a solution of the original POMDP M. Q.E.D.

The above proposition together with Proposition 3.1 provide us a route to solve POMDPs by designing an appropriate observer and a set of macros such that Proposition 3.4 is applicable for the POMDP whose action set consists of the set of macros (as described in Proposition 3.1).

In [5] such a design was carried out for a mobile-robot learning task with a standard Khepera robot. The task of the robot was to grasp a ball and hit the stick, standing in the corner of the "arena", with it. The problem is very difficult due to the limited sensory information available for the robot. Figure 1 shows the suggested subtask decomposition hierarchy whose basic macro-actions were implemented by hand. Different RL learning algorithms were tried to learn the optimal "switching" among these macros. It was found that the learnt policy could complete the task before timeout 90 % of the time, which was comparable to the performance of a hand-crafted policy. For a more detailed description of these experiments see [5].

4 CONCLUSIONS

First, some limitations of the basic RL algorithms and some methods to overcome these limitations were reviewed. Problems related to partial observability were then considered in detail. In particular, for certain deterministic problems, it was shown that the "prediction is sufficient for optimal behavior" principle is valid. It would be interesting to extend these results to general stochastic problems. Next, the soundness of a design method that transforms general POMDPs into solvable ones using *a priori* knowledge, sub-task decomposition and macro-actions was shown. Although RL methods seem to have reached a complexity already sufficient for many true-world applications, much work remains to be done to solve issues related to the automatic discovery of sub-task decompositions on the basis of partial solutions.

5 ACKNOWLEDGEMENTS

The author wishes to thank for Zsolt Kalmár and András Lőrincz the discussions.

References

[1] J. Baxter, A. Tridgell, and L. Weaver. Knightcap: A chess program that learns by combining td(lambda) with game-tree search. In *Proceedings of the International Conference on Machine Learning*, pages 28–36, 1998.

[2] R.H. Crites and A.G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 9*, 1997.

[3] T. Dietterich. The maxq method for hierarchical reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 1998.

[4] R. A. Howard. *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge, MA, 1960.

[5] Zs. Kalmár, Cs. Szepesvári, and A. Lőrincz. Module based reinforcement learning for a real robot. *Machine Learning*, 31:55–85, 1998. joint special issue on “Learning Robots” with the J. of Autonomous Robots (vol.5,pp.273–295,1997).

[6] S. Mahadevan and J. Connel. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence,*, 55:311–355, 1992.

[7] S. Mahadevan, N. Marchalleck, T. Das, and A. Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proceedings of the 14th International Conference on Machine Learning (IMLC '97)*, 1997.

[8] Sridhar Mahadevan. Enhancing transfer in reinforcement learning by building stochastic models of robot actions. In *Machine Learning: Proceedings of the Ninth International Workshop*. Morgan Kaufmann, June 1992.

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

[10] A.W. Moore and C. G. Atkeson. Memory-based reinforcement learning: Converging with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[11] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1997. MIT Press. in press.

[12] D. Precup and R.S. Sutton. Multi-time models for temporally abstract planning. In *Advances in Neural Information Processing Systems 10*. MIT Press, 1998.

[13] D. Precup, R.S. Sutton, and S.P. Singh. Planning with closed-loop macro actions. In *Working notes of the 1997 AAAI Fall Symposium on Model-directed Autonomous Systems*. AAAI Press/The MIT Press, 1997. in press.

[14] S.P. Singh and D.P. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems 9*, 1997.

[15] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. MIT Press, 1996.

[16] P. Tadepalli and D. Ok. Model-based average reward reinforcement learning. *Artificial Intelligence*, 1998. to appear.

[17] G.J. Tesauro. Td-gammon, a self-teaching backgammon program achieves master-level play. *Neural Computation*, 6:215–219, 1994.

[18] Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. The MIT Press, Cambridge, 1995.

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

[20] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intellience*, 1995.