An Evaluation Criterion for Macro Learning and Some Results

Zsolt Kalmár and Csaba Szepesvári
e-mails: {kalmar,szepes}@mindmaker.hu

[1] Mindmaker Ltd.
Budapest, 1112
Konkoly Thege M. u. 29-33
HUNGARY

Abstract

It is known that a well-chosen set of macros makes it possible to considerably speed-up the solution of planning problems. Recently, macros have been considered in the planning framework, built on Markovian decision problem. However, so far no systematic approach was put forth to investigate the utility of macros within this framework. In this article we begin to systematically study this problem by introducing the concept of multi-task MDPs defined with a distribution over the tasks. We propose an evaluation criterion for macro-sets that is based on the expected planning speed-up due to the usage of a macro-set, where the expectation is taken over the set of tasks. The consistency of the empirical speed-up maximization algorithm is shown in the finite case. For acyclic systems, the expected planning speed-up is shown to be proportional to the amount of "time-compression" due to the macros. Based on these observations a heuristic algorithm for learning of macros is proposed. The algorithm is shown to return macros identical with those that one would like to design by hand in the case of a particular navigation like multi-task MDP. Some related questions, in particular the problem of breaking up MDPs into multiple tasks, factorizing MDPs and learning generalizations over actions to enhance the amount of transfer are also considered in brief at the end of the paper.

Keywords: *Reinforcement learning, MDPs, planning, macros, empirical speed-up optimization.*

Electronic version: http://victoria.mindmaker.hu/~szepes/papers/macro-tr99-01.ps.gz

1 Introduction

Recently several researchers begin to study issues related to learning and planning at multiple levels of temporal abstraction within the framework of reinforcement learning (RL) and Markov decision processes (MDPs) (e.g. [16, 5, 19, 17, 6, 10, 11]). Extension of MDPs with temporally extended actions (i.e. macros) provides a way to model multiple time-scales in an efficient way. Empirically, macros have been shown to be useful in many tasks. For example, the application of macros made it possible to solve larger problems than ever before when primitive actions were eliminated from the MDP [11, 6] and well-designed macros have been shown to speed-up planning considerably [11]. Most of the authors assume that macros are given (e.g. designed by a well-informed human). However, one would like to have fully automatic and autonomous methods that need a minimal amount of a priori information about their environments. This raises the issue of learning macros, being the subject of the recent post-NIPS workshop on RL (a page of this workshop with a list of presented papers can be found at http://envy.cs.umass.edu/~dprecup/schedule.html). However, in our view before rushing to considering various macro-learning heuristics, a framework that helps to compare macro-sets needs to be set-up.

The primary purpose of this paper is to propose such a framework. Namely, here we suggest a criterion of comparing macro-sets using which it is possible to compare different macro-learning algorithms in terms of their performance. We will then measure the performance of a macro-learning algorithm in terms of the utility of the macro-set constructed by it (thus temporally neglecting other issues such as memory and time-efficiency). We want to measure the utility in the most natural way given the traditional view that macros speed-up planning: given a fixed planning algorithm the utility of a macro-set is defined to be the expected planning speed-up due to the usage of that macro-set. Here the expectation is taken over a probability distribution over the tasks. In the simplest case, the training set compromises a finite sample of tasks from the given distribution. In the more realistic cases the learner has only limited knowledge about the training tasks themselves: for example, the learner might also have to learn the common underlying dynamics, etc. However, the simplest (fully informed) case is still interesting and not just from the theoretical point of view but also from the practical point of view as it captures our expectation that given a sufficiently large number of tasks the problem of approximating the dynamics can (and should) be decoupled from the problem of learning a good set of macros. Additionaly, there are practical problems (simulation-problems) when the dynamics is known but the construction of a good set of macros is still of big importance.

In any way, our approach makes it clear that questions analogous to those questions considered in the probabilistic account of pattern recognition (see [1]) arise here, too:

- Existence of optimal solutions: does there exist an optimal solution?
- Consistency: does there exist an algorithm that converges to the optimal solution (set of macros) as the number of training instances goes to infinity?
- Efficiency: time and memory boundedness questions (scaling issues) can be considered, where size can be the number of training instances and/or the problem size.
- Measuring performance, comparing algorithms: given an algorithm and a finite set of training instances, how to measure (empirically) the performance of the algorithm, how to compare algorithms?

One big difference to the theory of pattern recognition is that here the random variable (a measure of the speed-up) over which the expectation is taken is real-valued whilst in pattern recognition the random variable (error for a given input-output pair) is binary. Another difference is that here the training instances (tasks) are complicated objects. Nevertheless, it seems that a number of techniques developed for pattern recognition can be reused here with some modifications.

An important problem not having a counterpart in pattern recognition is how to choose the underlying planning algorithm such that the overall efficiency of the planning algorithm together with a macro-learning algorithm is optimal.

In this paper, we start to consider the above issues. Namely, we put forth a rigorous definition for the utility of macro-sets, which includes the definition of multi-task MDPs, which we also define here. Afterwards, the consistency of the empirical speed-up optimization algorithm (the counterpart of empirical risk-minimization algorithm of pattern recognition) is shown in the table-lookup case.

Navigation like problems are considered in some more detail and an algorithm is proposed which approximates the empirical speed-up optimization algorithm in this particular case. Some computer experiments are also presented. Related problems for further research are discussed. One such problem is breaking up MDPs into multiple tasks. This is shown to be equivalent to the problem of hidden states. Another problem is the issue of learning "good" representations. By means of an example, we show that good macros could in theory be used to change the problem-representation in a useful and non-trivial way. Changing the problem-representation may then further increase the speed-up of planning. On the negative side, it is shown that the problem of learning a factorized representation, which can be argued to be the first step towards turning a flat representation into a structured one, is as difficult as the graph isomorphism problem.

The rest of the paper is organized as follows: In Section 2 we introduce the basic concepts needed for the definition of our utility. Most notably, we introduce here the concept of multi-task MDPs. In Section 3 we put forth the definition of the utility of macro-sets and prove a result about the asymptotic consistency of the so-called empirical speed-up (utility) maximization algorithm. In Section 4 an algorithm for navigation like problems is proposed and results of some computer experiments are presented. Finally, the paper is concluded in Section 5.

2 Notation and Definitions

In this section, we introduce the definitions and notations that will be used throughout the paper. Since we introduce quite a bit of non-standard notation, for the sake of clarity here we shall list some pieces of the basic MDP theory, too. However, we basically assume that readers are familiar with the theory of MDPs and thus define some basic concepts only loosely and list results without proofs. Readers familiar with MDPs may try to read this article by first entirely skipping this section, stopping only at Section 2.3 where the concept of multi-task MDPs is given and/or returning later to the relevant portion(s) of this section when clarification is needed. In order to facilitate this kind of reading we organized this section into several subsections. Cited results related to MDP theory can be found in [13] and results related to macros can be found in [17, 10, 11].

2.1 Markovian Decision Problems

For a set S, 2^S will denote the power set of S. The symbol $\chi(condition)$ equals 1 if '*condition*' holds and equals 0 otherwise. $|S|$ will denote the cardinality of S. In probabilistic contexts, "a.s." abbreviates "almost surely".

By a *Markovian decision problem* (MDP) $\mathcal{P}$ we mean the 5-tuple $\mathcal{P} = (X, A, p, c, \gamma)$, where X is called the state set, A is called the action set, $p : X \times A \times X \to [0,1]$ is the transition probability function satisfying

$$\sum_y p(x, a, y) = 1 \qquad (1)$$

for all $(x, a) \in X \times A$, $c : X \times A \to \mathbf{R}$ is the cost function, and $\gamma \in [0,1]$ is the so-called discount factor. (X, A, p) will be called the *skeleton* of the MDP or an *MDP-skeleton*. Actually, the definition of MDPs should be extended so that we can restrict the availability of actions to certain states, called the domain of the action. Similarly, one could list the set of available actions for each state x. A *stopped* MDP is an MDP $\mathcal{P}$ together with a probabilistic stopping condition $\psi : X \to [0,1]$ which defines in each step the probability of stopping in that step. When the MDP is stopped no more costs are incurred (this could also be modeled by a transition to a single auxiliary absorbing state with zero transition costs).

Informally, a policy is a set of rules which determine given the past the action to be chosen. The objective of decision making is to minimize the total expected discounted cost.

With each MDP $\mathcal{P}$ we associate a number of objects. The backup operator $T_{\mathcal{P}} : B(X) \to B(X)$ is defined by

$$(T_{\mathcal{P}} V)(x) = \min_{a \in A} \sum_{y \in X} p(x, a, y) \left(c(x, a, y) + \gamma V(y) \right).$$

$V^*_{\mathcal{P}}$ will denote the fixed point of $T_{\mathcal{P}}$ which we will assume to exist. The basic results of MDP theory ensure us that $V^*_{\mathcal{P}}$ equals to the optimal cost function that gives us the values of the minimum expected total (discounted) costs when the decision process is started from a given state. An optimal policy is a policy that achieves $V^*_{\mathcal{P}}$. It is also known that we can find optimal policies among the policies that prescribe to choose actions deterministically. Moreover, optimal policies can further be restricted to depend only on the last state in the history of the decision process ($\pi(\text{history}, x_t) = \pi(x_t)$, where x_t is the last state visited in the history). Such optimal policies are called optimal (deterministic) stationary (Markovian) policies. They can be characterized by the property that they are induced by some $\pi^* : X \to A$ function satisfying $\pi^*(x) \in \mathrm{Argmin}_{a\in A} \sum_{y\in X} p(x,a,y)\left(c(x,a,y) + \gamma V^*_{\mathcal{P}}(y)\right) = \mathrm{Argmin}_{a\in A} Q^*(x,a) = \Pi^*(x)$. A stationary Markovian policy will be identified with the function inducing it. We will need the notion of stationary multi-policies. A (stationary) *multi-policy* is a function $X \to 2^A$. Therefore $\Pi^*(x)$ defined above can also be viewed as a multi-policy. A policy π is compatible with the multi-policy Π if $\pi(x) \in \Pi(x)$ for all $x \in X$. A policy $\pi : X \to A$ will be identified with the multi-policy defined by $x \mapsto \{\pi(x)\}$.

A policy is called ε-optimal if $V_\pi(x) \le V^*(x) + \varepsilon$, where $V_\pi(x)$ is the total expected (discounted) cost accumulated over an infinite time-horizon if policy π is used from a trajectory starting with x.

The myopic policy w.r.t. a real-valued function V is defined as a policy compatible with the multi-policy defined by

$$\mathop{\mathrm{Argmin}}_{a\in A} \sum_{y\in X} p(x,a,y)\left(c(x,a,y) + \gamma V(y)\right).$$

An *ε-precise MDP planning algorithm* (or simply planning algorithm) is an algorithm, whose input is an MDP $\mathcal{P}$ and a number $\varepsilon > 0$ and whose output is an ε-optimal policy. The *ε-running time* of a planning algorithm on an MDP $\mathcal{P}$ is interpreted in the natural way. This will also be called the *ε-planning time* when the planning algorithm is fixed.

2.2 Macros

A (Markov) *macro*[1] μ is a 3-tuple (S, ϕ, σ), where $S \subset X$ is called the domain of μ, $\phi : X \to A$, $\sigma : X \to [0,1]$ is a probability distribution over the states, which is interpreted as a probabilistic stopping rule, i.e., $\sigma(x)$ is the probability of stopping macro on arriving in state x ($x \in X$). The domain of a macro μ will also be denoted by $\mathrm{Dom}(\mu)$. The set of states which can be reached from some $x \in S$ with positive probability when macro μ is used from state x are called the *reachable states* under macro μ. The reachable states for which $\sigma(x) < 1$ are called the *continuation states*.

In this article we will also consider the special case when $\sigma(x) = 0$ for all $x \in S$ and $\sigma(x) = 1$ otherwise. In such a case ϕ is not needed to be defined for states outside of S. Such macros will be denoted by $\mu = (S, \phi)$,

[1] Other authors would call our Markov macros “options” or “generalized actions”.

where $\phi : S \to A$ and can be considered as partial (partially specified) policies. Accordingly, these macros will be called *region-based* macros [4].

The *exit-states* of a macro $\mu = (S, \phi)$ are those states x of S for which action $\phi(x)$ brings us outside of S with positive probability. This set will be denoted by $\partial^- \mu$. The *target* or *goal states* of μ are those states y outside of S for which there exists a state x in S such that $p(x, \phi(x), y) > 0$. The set of these states will be denoted by $\partial^+ \mu$. $\partial^- \mu$ and $\partial^+ \mu$ can be considered as the inner and outer boundaries of the targets of μ. The reachable states under the region-based macro μ are exactly the states in $S \cup \partial^+ \mu$. The set of continuation states equals to S.

Sets of macros will be denoted by $M, M', M_1, M_2, \ldots$. The execution protocol of macros is taken to be the familiar call-and-return protocol. [2] This means that macros will be treated in the MDP as actions extended in time, i.e., if the decision maker chooses a macro for execution then she has to follow the actions prescribed by the macro until the execution of the macro stops. The stopping criterion is defined by flipping a biased coin in each time-step, the bias given by the probabilities $P(Head) = \sigma(x), P(Tail) = 1 - \sigma(x)$. Execution is stopped if the result of a flip is $Head$. The execution protocol together with a policy over primitive actions (i.e., elements of A) and macros yield a semi-Markov decision process [13].

It is well known that given a set of macros M, one may define a *derived or augmented "MDP"*, denoted by $\mathcal{P}_M$, in which the basic action set A is extended with M (M and A are assumed to be disjoint), c and p are defined appropriately such that $V^*_{\mathcal{P}_M} = V^*$ and any policy of $\mathcal{P}$ derived in the natural way from any optimal policy of $\mathcal{P}_M$ is an optimal policy in $\mathcal{P}$. The reason of apostrophes is that the derived MDP is not a valid MDP in the sense that its "transition probability function" does not satisfy Equation 1. If the discount factor γ of the original MDP $\mathcal{P}$ equals to 1 than the derived MDP will be a "true" MDP: in this case $p(x, \mu, y)$ corresponds to the probability of stopping at y given that policy μ is followed from state x, $c(x, \mu, y)$ is the expected total cost experienced when μ is followed from state x until μ stops at y ($x \in Dom(\mu)$).

In Section 4 where we consider particular algorithms, we will look at macros (and policies) as sets. The reason of this is that in the discrete domains we consider equivalent solutions are possible and an arbitrary choice from these solutions would break some symmetries, resulting in a sub-optimal behavior.

[2] The relation of this protocol to other protocols is discussed in some detail in [18].

Therefore we identify $\mu = (S, \phi)$ with the set $\{(x, a) : x \in S,\ a = \phi(x)\}$ and we assign the set $\{(x, a) : a = \phi(x), x \in X\}$ to $\mu = (S, \phi, \sigma)$. The definition is extended to policies and multi-policies in the natural way. Therefore, we can talk about the intersection, union, difference, etc. of (region-based) macros, policies and multi-policies. We call two macros *orthogonal* if their intersection is empty. This notion extends to non-region-based macros in the natural way.

2.3 Multi-task MDPs

According to the traditional view in artificial intelligence macros provide a useful tool for solving a series of search problems: knowledge about how to search can be transformed into macros and can be later reused in the solution of new instances of the same search problem (see e.g. [2, 7, 8, 14] and Chapters 10 and 11 of [9]).[3] More recently, macros have been considered in the domain of planning in MDPs [12].

In this paper we also take the viewpoint that the primary goal of macros is to speed-up the planning of optimal policies for new, unseen tasks. Different tasks are defined by different cost functions over the same MDP skeleton. This might sound a bit artificial and one might worry about loosing optimality when approximating MDPs with multiple-task MDPs. However, since big MDPs are in general intractable one always needs to rely on some kind of heuristics. We consider the multi-task model as an approximation to the underlying MDP and the solution method as a heuristic method for solving the original MDP.[4] The problem of "factorizing out" task IDs from a monolithic state will be considered later.

If the tasks are recognizable (i.e., the decision-maker is given the task ID

[3]In this traditional setup macros are simply sequences of actions (open-loop non-stationary policies). This is appropriate for search problems where there is no real plan-execution, whilst if plan-execution is non-trivial (e.g. because the domain is stochastic) then closed-loop macros (plans) are more appropriate. Some authors have considered more flexible notions of macro-operators, e.g. [15, 20].

[4]Note that the approximation with multi-task MDP fits average cost or undiscounted MDPs better than discounted models. In a discounted model the discounting of future cost values is a non-negligible part of the model. This discounting clearly cannot be captured by a multi-task approximation, which assumes complete decoupling of the cost-streams encountered during different tasks. The multi-task model further assumes that the expected time to finish a task is finite. Note that the exact value of the finish-time is not related to the precision of the approximation to the monolithic MDP, but may effect the heuristics in other ways.

from which she can reconstruct the appropriate cost structure) then there appears the tradeoff between memory-usage/off-line computation versus on-line computation (determining what to do next). Using more memory and more off-line computational power it becomes possible to cache e.g. precomputed macros (sub-policy) that can be reused when a new task comes in. This reduces the demands on on-line computation. Note that although the off-line computation should not be intractable it may work on a difference time-scale.

Turning back to the formalism, let us first introduce the notion of *multi-task MDPs*.

Definition 2.1 *An 8-tuple $\mathcal{P}_{\mathcal{I}} = (X, A, p, \gamma, I, C, P_I, \Psi)$ is called a multi-task MDP if (X, A, p) is an MDP-skeleton, $0 \le \gamma \le 1$ is a discount factor and I is a set, called the* task set*. The elements of I are called the tasks. C is a mapping that maps tasks to (cost-)functions over $X \times A \times X$, P_I is a probability distribution over I, and Ψ is a mapping that maps tasks to probabilistic stopping rules. The image of task $i \in I$ under C (Ψ) will be denoted by c_i (ψ_i, respectively).*

The interpretation of a multi-task MDP $\mathcal{P}_{\mathcal{I}}$ is as follows: In each time-step there is single *active* task. If task $i \in I$ is active then the decision maker has to solve the stopped MDP $\mathcal{P}_i = (X, A, p, c_i, \gamma, \psi_i)$ (i.e., it incurs costs from this problem). The time when the active task i "stops" is determined by the stopping-rule induced by ψ_i (the stopping condition is identical as for macros). After task i has stopped a new task is selected from I according to the probability distribution P_I. Clearly, a multi-task MDP could be viewed as one MDP if the states of X were extended by task indices. In addition, the notion of multi-task MDPs could be extended to involve a Markovian dynamics over the tasks by considering e.g. a transition probability function $p_I : I \times I \to [0, 1]$ instead of the distribution $p_I : I \to [0, 1]$. In this case, $p_I(i, j)$ would determine the probability of chosing task j after task i has been finished. More complicated dynamics are also possible. Because of lack of space we do not consider issues concerning such dynamics over the tasks here.

In the computer experiments, we will mainly be interested in multi-goal MDPs. A goal-oriented MDP is a stopped MDP, the stopping probability being 1 over the set of 'goals' G and otherwise being zero. All the costs are taken to be equal to one. Assume a finite MDP. It follows that if there is a single policy π bringing the decision-maker from any state to any of the goal states with positive probability then the total expected undiscounted cost, i.e., the expected number of steps to reach the goal set will be finite. This is because in finite MDPs the probability of staying outside of G while using π decays exponentially. Such a goal-oriented MDP will be called proper. We shall only consider proper goal-oriented MDPs. A (proper) multi-goal MDP is a special multi-task MDP which is composed of a series of (proper) goal-oriented MDPs such that for task i $\psi_i(x) = 1$ whenever $x \in G_i$. Here G_i denotes the goal-set corresponding to task i.

Random tasks will be denoted by $\tau, \tau_1, \ldots, \tau_n$. These will always denote independent random variables with distribution P_I. We will consider finite multi-task MDPs, where I,X and A are finite sets. For a probability distribution over the tasks, P_I, will be identified with the vector $(p_I(1), \ldots, p_I(n))^T$, where $n = |I|$. In general, n will denote the number of tasks. If $I_N = (i_1, \ldots, i_N)$ are (not necessarily distinct) tasks then the empirical probability distribution over I, denoted by $\hat{p}_{I_N}$, is defined by $\hat{p}_{I_N}(i) = (1/N)\sum_{k=1}^{N} \chi(i_k = i)$. The multi-task MDP $(X, A, p, \gamma, I_N, C_{I_N}, \hat{P}_{I_N}, \Psi_{I_N})$, where $C_{I_N}(i_k) = c_{i_k}$, $\Psi_{I_N}(i_k) = \Psi(i_k)$ $(k = 1, 2, \ldots, N)$ is called the empirical multi-task MDP and will be denoted by $\mathcal{P}_{\{i_1,\ldots,i_N\}}$.

The size of a finite multi-task MDP is $\text{size}(\mathcal{P}_I) = |I| + |X| + |A| + \text{size}(p) + \sum_i \text{size}(c_i)$, where $\text{size}(p)$ denotes the memory requirement of storing p (apart from a constant factor). Similarly, $\text{size}(c_i)$ denotes the memory requirement of storing the cost function c_i. Here we assumed that p and c_i can be represented by a finite number of bits.

3 A Criterion for Evaluating Macro-sets

The criterion we propose suggests to compare macro-sets based on the notion of expected speed-up of planning in multi-task MDPs. In order to elaborate this we need to fix an arbitrary MDP planning algorithm with a finite running time. Specifically let us denote the running time of the planning algorithm on MDP $\mathcal{P}$ by $R(\mathcal{P}, \varepsilon)$. For simplicity we will assume a fixed $\varepsilon \geq 0$ and will suppress ε in the following expressions.

Definition 3.1 *Let $\mathcal{P}_{\mathcal{I}}$ be a multi-task MDP.*

The expected planning time is

$$R(\mathcal{P}_{\mathcal{I}}) = E[R(\mathcal{P}_\tau)],$$

where τ is a random task with distribution P_I.

Let M denote a set of macros over (X, A, p). Remember that if $\mathcal{P}$ is an MDP then $\mathcal{P}_M$ denotes the derived MDP whose action set is extended by the macros of M. We shall assume that our planning algorithm is such that it can handle derived MDPs even if the discount factor of the original MDP is smaller than 1. Then the speed-up of planning due to M is

$$U(M; \mathcal{P}) = R(\mathcal{P}) - R(\mathcal{P}_{\mathcal{M}}). \tag{2}$$

Note that one would normally expect that $U(M; \mathcal{P}) \geq 0$, i.e., that $R(\mathcal{P}) \geq R(\mathcal{P}_{\mathcal{M}})$. However, in certain cases, due to the increased branching factor of $\mathcal{P}_{\mathcal{M}}$, it might happen that $U(M; \mathcal{P}) < 0$.

Definition 3.2 *The expected speed-up of the macro set in the multi-task MDP $\mathcal{P}_{\mathcal{I}}$ is*

$$U(M; \mathcal{P}_{\mathcal{I}}) = E[U(M; \mathcal{P}_\tau)], \tag{3}$$

where τ is a random task with distribution P_I.

We evaluate macro sets according to $U(M; \mathcal{P}_{\mathcal{I}})$. It seems perfectly reasonable to call a macro set M' better than the macro set M if $U(M; \mathcal{P}_{\mathcal{I}}) > U(M'; \mathcal{P}_{\mathcal{I}})$. However, often the memory available for storing macros is limited. Define $\text{size}(\mu)$ to be the size of memory required to store macro μ ($\text{size}(\mu) = O(\text{Dom}(\mu))$). Also denote the memory requirements of a macro set M by $\text{size}(M) = \sum_{\mu \in M} \text{size}(\mu)$. Then one would like to look at the set of macro-sets

$$\mathcal{M}(K) = \{M : M \ \text{ is a macro set s.t. } \text{size}(M) \leq K\},$$

where $K > 0$ is a bound on the available memory. Then

$$\mathcal{M}^*(\mathcal{P}_{\mathcal{I}}; K) = \operatorname*{Argmax}_{M \in \mathcal{M}(K)} U(M; \mathcal{P}_{\mathcal{I}})$$

gives macros with size less than K that yield the maximal planning speed-up. Since the maximum is taken over a finite set, the existence of a maximizing element, i.e., an optimal macro-set is guaranteed. We define

$$U^*(\mathcal{P}_{\mathcal{I}}; K) = \max_{M \in \mathcal{M}(K)} U(M; \mathcal{P}_{\mathcal{I}}).$$

Moreover, a generic macro-set from $\mathcal{M}^*(\mathcal{P}_{\mathcal{I}}; K)$ will be denoted by $M^*(\mathcal{P}_{\mathcal{I}}; K)$.

We shall need the following definition:

Definition 3.3 *If the finite multi-task MDP, $\mathcal{P}_\mathcal{I}$, is fixed and $p = (p(1), \ldots, p(n))^T$ is a distribution over the task indices not necessarily being equal to p_I then let*

$$U(M;p) = \sum_{i \in I} U(M;\mathcal{P}_i)p(i).$$

Moreover, $M^*(p;\mathcal{M})$ shall denote a generic element of $\operatorname{Argmax}_{M\in\mathcal{M}} U(M;p)$ (here $\mathcal{M}$ is a set of macro-sets).

We propose to consider the following problem:

Definition 3.4 (Speed-Up Optimization Task) *Solve the following problem:*
Input: *$\mathcal{P}_\mathcal{I}$ multi-task MDP, $K > 0$*
Output: *an element of $\mathcal{M}^*(\mathcal{P}_\mathcal{I};K)$.*
Size: *The size of a problem instance $(\mathcal{P}_\mathcal{I}, K)$ is defined to be $K + \mathrm{size}(\mathcal{P}_\mathcal{I})$.*

Since the complete multi-task MDP is in general unknown, we shall not try to solve this optimization task directly, but we are much more interested in the following *learning problem*:

Definition 3.5 (Learning of Speed-Up Optimization) *Solve the following problem:*
Fix an arbitrary multi-task MDP $\mathcal{P}_\mathcal{I}$.
Input: *Skeleton of $\mathcal{P}_\mathcal{I}$: $\Xi = (X, A, p)$ and a sequence of i.i.d. tasks from P_I: $\{\tau_1, \ldots, \tau_N\}$, the corresponding decision problems $\mathcal{P}_{\tau_j}$, $j = 1, 2, \ldots, N$ and a bound on the macro-memory $K > 0$.*
Output: *a macro $M_N = M(\Xi; \{\tau_1, \ldots, \tau_N\})$ defined over the skeleton Ξ, s.t., $\mathrm{size}(M_N) \le K$.*
Size: *The size of a problem instance $(\Xi, \{\tau_1, \ldots, \tau_N\}, K)$ is defined to be $\mathrm{size}(\Xi) + K + \sum_{j=1}^{N} \mathrm{size}(\mathcal{P}_{\tau_j})$.*

According to this definition, when facing a learning problem one does not know the whole set of tasks, nor one knows the distribution over these tasks. In the definition it was assumed that the MDP-skeleton is known. In practice, this is not a real limitation since we believe that learning a model of the dynamics is much easier than guessing a good macro-set. However, the rigorous investigation of the effect of imprecise models is an important problem for future research. A (macro-)learning algorithm is an algorithm whose inputs and outputs are specified as above in Definition 3.5.

In this paper, we will be interested primarily in the asymptotic behavior of the learning algorithms:[5]

Definition 3.6 *A learning algorithm is called* consistent *if for any fixed multi-task MDP $\mathcal{P}_{\mathcal{I}}$ the sequence of macros, M_N, produced by the algorithm satisfy*

$$U(M_N; \mathcal{P}_{\mathcal{I}}) \to U^*(\mathcal{P}_{\mathcal{I}}; K) \quad \text{a.s.} \tag{4}$$

as $N \to \infty$.

We will consider the empirical speed-up maximization (ESM) algorithm (which is motivated by empirical risk-minimization algorithms well known in the pattern-recognition literature):

Definition 3.7 *Fix a multi-task MDP $\mathcal{P}_{\mathcal{I}}$. Let the input be the skeleton of $\mathcal{P}_{\mathcal{I}}$, $\Xi = (X, A, p)$, and the sequence of i.i.d. tasks from P_I: $\{\tau_1, \ldots, \tau_N\}$ with the corresponding decision problems $\mathcal{P}_{\tau_j}$, $j = 1, 2, \ldots, N$ and a bound on the macro-memory $K > 0$.*

Then the empirical speed-up maximization (ESM) *algorithm's output is an element of the set $\mathcal{M}^*(\mathcal{P}_{\{\tau_1,\ldots,\tau_N\}}; K)$.*

The algorithm selects (in some unspecified way) a macro-set which is optimal for the empirical multi-task MDP $\mathcal{P}_{\{\tau_1,\ldots,\tau_N\}}$. The algorithm is well defined as one can construct the empirical multi-task MDP $\mathcal{P}_{\{\tau_1,\ldots,\tau_N\}}$ from the available inputs.

We prove the following theorem:

Theorem 3.1 *The ESM algorithm is consistent for finite multi-task MDPs.*

We shall need the following lemma:

Lemma 3.1 *Fix a finite multi-task MDP $\mathcal{P}_{\mathcal{I}}$. Let $\mathcal{M}$ be the set of available macro-sets. Assume that p_n converges to $p = P_I$ (the distribution over the tasks in $\mathcal{P}_{\mathcal{I}}$). Then also*

$$\lim_{n\to\infty} U(M^*(p_n; \mathcal{M}); p_n) = U(M^*(p, \mathcal{M}); p),$$

i.e., $U(M^*(\cdot; \mathcal{M}); \cdot)$ is continuous.

[5] Sample and time-complexity issues should also be considered in the future.

Proof. Let $M_N = M^*(p_N; \mathcal{M})$ and let $M^* = M^*(p; \mathcal{M})$. We prove the lemma by contradiction. Assume that $U(M_n; p_n) \not\to U^* = U(M^*; p)$. Since the number of available macros is finite, we may take a subsequence n_k s.t. $M_{n_k} = \hat{M}$ for some macro $\hat{M}$ and $U(\hat{M}; p_{n_k}) \not\to U^*$. Since $U(\hat{M}; \cdot)$ is continuous, $\lim_{k\to\infty} U(\hat{M}; p_{n_k}) = U(\hat{M}; p)$. Therefore $U(\hat{M}; p) \neq U^*$ and since $\hat{M} \in \mathcal{M}$ also $U(\hat{M}; p) < U^*$. Let $\delta = U^* - U(\hat{M}; p)$. If k is big enough then $|U(M^*; p_{n_k}) - U(M^*; p)| < \delta/2$ and therefore $U(M^*; p) - \delta/2 < U(M^*; p_{n_k})$. Similarly, if k is big enough then $|U(\hat{M}; p_{n_k}) - U(\hat{M}; p)| < \delta/2$ and therefore $U(\hat{M}; p_{n_k}) < U(\hat{M}; p) + \delta/2$. Consequently, by the choice of δ, also $U(\hat{M}; p_{n_k}) < U(M^*; p_{n_k})$. This is a contradiction and therefore $U(M_n; p_n) \not\to U^*$ cannot hold. *qed.*

Proof. [**Proof of Theorem 3.1**] By the law of large numbers the empirical distribution $\hat{p}_N$ converges to p a.s. Now take in Lemma 3.1 $\mathcal{M} = \mathcal{M}(K)$ the set of macro-sets with size smaller than or equal to K. Since $\mathcal{M}^*(\mathcal{P}_{\tau_1,\ldots,\tau_N}; K) = \mathrm{Argmax}_{M\in\mathcal{M}} U(M; \hat{p}_N)$ the lemma applies and shows that 4 holds for the sequence of macro-sets chosen by the ESM algorithm. *qed.*

We remark that using uniform convergence theorems it seems to be quite possible to extend the above consistency result to multi-task problems with infinite task-sets (the number of different tasks is infinite even for multi-task MDPs with finite MDP-skeletons).[6] However, the extension to infinite macro sets (e.g. when macros were represented on some language) does not seem to be trivial. In order to prove such a result one would presumably need to use tools such as the VC-dimension extended to this case. We consider this as an important open theoretical problem. Note that for infinite macro-sets the requirement of asymptotic consistency brings in the usual general methods that are used to ensure consistency in pattern recognition problems, such as e.g. structural risk-minimization, regularization, the minimum description length principle or the Bayesian information criterion.

Of course, the above result is mainly of theoretical interest as it does not seem to be any simple way to find an element of $\mathcal{M}^*(\mathcal{P}_{\{\tau_1,\ldots,\tau_N\}}; K)$ since the set of possible macros may be quite large (though being finite by assumption). [7] In the next section we consider a heuristic approach to approximate the ESM algorithm in the finite case and for navigation-like tasks. The importance of

[6]Note that the above theorem remains valid for any finite optimization problem where the optimized value is an expectation of a distribution from which a finite sample is generated.

[7]Note that if the size constraints are loose enough then the optimal solutions to the individual macros form an empirically optimal macro set.

these algorithms lies in that they are derived from some special properties of navigation-like tasks. We hope to extend the proposed algorithms to the more compelling infinite case in the future.

4 A Macro-Learning Algorithm and Some Computer Experiments

The planning algorithm we consider is synchronous value iteration which, for an MDP $\mathcal{P}$, computes the sequence function $V_{N+1} = T_{\mathcal{P}} V_N$ and with a pessimistic initialization. This means that initially $V_0 \geq V^* = V_{\mathcal{P}}$ should hold (remember that we use a cost-based setup). Actually, in our implementation we have chosen multi-goal MDPs and $V_0(x) = +\infty$ for all $x \in X \setminus G$, where G is the actual goal-set, $V_0(x) = 0$ for all $x \in G$. We used the usual arithmetic over the extended reals. With such a V_0 we will say that "value iteration is initialized high". [8]

We will consider value-iteration which stops when the max-norm of the Bellman residual $(T_{\mathcal{P}}^{N+1} V_0 - T_{\mathcal{P}}^N V_0)$ becomes smaller than ε for the first time. The policy returned by the algorithm is the myopic one w.r.t. V_N, where V_N denotes the last estimate of the optimal cost function.

In order to motivate the development of the algorithm we shall consider a modified definition of running time that depends only on the iteration number. Obviously, this number favors larger macro sets. [9] This is also equal to assuming that the minimization over the actions can be solved in one step, e.g. by means of some parallel computation mechanism.

Consider the following proposition:

Proposition 4.1 (Reduction rules) *Let $\mu_1, \mu_2 \in M$ be distinct macros from*

[8] Although we know that synchronous value iteration is not the best planning algorithm available, we have chosen it because of its simplicity and since Sutton and his coworkers have also utilized it in their work [11]. Asynchronous value-iteration (with a fixed ordering) would probably represent a more efficient alternative since for goal oriented problems it converges faster when the updates are performed in reverse order according to the distance from the goal-set. The relation of this order to the goal-set would probably interfere in an unpredictable way.

[9] This comes from the following observation: if M is a macro-set then $T_{\mathcal{P}_{\{M\}}} \leq T_{\mathcal{P}}$ and $T_{\mathcal{P}_{\{M\}}} V^* = T_{\mathcal{P}} V^*$. Therefore, by induction $V^* \leq T_{\mathcal{P}_{\{M\}}}^N V_0 \leq T_{\mathcal{P}}^N V_0 \leq V_0$ holds for all N. This was also pointed out by [3].

the macro-set M. If $\partial^+\mu_2 \subseteq \partial^+\mu_1$ and $\mu_2 \subseteq \mu_1$ then synchronous value iteration stops at exactly the same time for both $\mathcal{P}_M$ and $\mathcal{P}_{M\setminus\{\mu_2\}}$.

If μ_1 and μ_2 are distinct macros, $\mu_1, \mu_2 \notin M$ and the set of continuation states under μ_1 and μ_2 are disjoint then synchronous value iteration stops at exactly the same time for both $\mathcal{P}_{M\cup\{\mu_1,\mu_2\}}$ and $\mathcal{P}_{M\cup\{\mu_1\cup\mu_2\}}$.

Proof. The first part of the proposition follows from the simple observation that for any $x \in \mathrm{Dom}(\mu_2)$ there is a one-to-one correspondence between the trajectories resulting from using μ_2 until it stops and the trajectories resulting from using μ_1 until it stops. Therefore $c(x, \mu_1, y) = c(x, \mu_2, y)$ for all $y \in X$ and $p(x, \mu_1, y) = p(x, \mu_2, y)$. This shows that $T_{\mathcal{P}_M} = T_{\mathcal{P}_{M\setminus\{\mu_2\}}}$.

The second part follows along the same line of reasoning than the first part. *qed.*

This proposition gives us a tool to eliminate superfluous macros and thus reduce the branching factor [10]. Note that for region-based macros the assumption of the second part is satisfied if the domains of the macros are disjoint.

The main heuristics of our algorithm is that macros should be sub-policies of the optimal multi-policies for the known tasks (multi-policies are needed because of the possible symmetries in the MDP). This is a reasonable heuristics since a well-chosen (informally "broad and long") sub-policy of an optimal policy would clearly speeds-up the planning time. This is illustrated below in Figure 1 showing a usual 2D navigation task and a macro. The big dots denote walls, small dots denote states in the open space, and the thin sticks starting at a small dot give the direction of the action chosen by the macro at the corresponding state. States without sticks are outside of the domain of the macro. There were two tasks defined by goal-states placed in the lower left corner.The intersection of the corresponding two optimal policies is shown in the figure as a macro. This macro yields a speed-up of about 20 (and a speed-up of about 10 when the total computation time is measured). The length of the corridor plays the strongest influence on the speed-up factor here.

We conjecture that value iteration is not speeded-up by macros having an empty intersection with the optimal multi-policy. Unfortunately, we could not prove or disprove this. We consider this also as an interesting topic for further research.

If the optimal policies are acyclic then a macro orthogonal to the optimal multi-policy does not help, as it follows from the next proposition:

[10] Other reductions are also possible which we do not list here for brevity. The above reductions were utilized to optimize our algorithm.

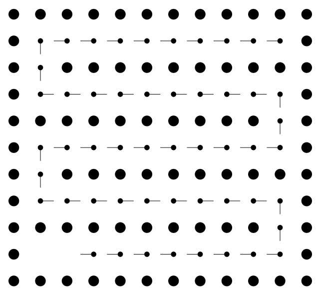

Figure 1: **The intersection of two optimal policies for tasks with neighboring goals.** The figure shows a macro in a specific environment. The big dots denote walls, small dots denote states in the open space, and the thin sticks starting at a small dot give the direction of the action chosen by the macro at the corresponding state. States without sticks are outside of the domain of the macro. The intersection of the policies defined for the two tasks defined by goal-states placed in the lower left corner are shown in the figure as a macro. .

Proposition 4.2 *Consider a goal-oriented MDP in which the optimal stationary policies are acyclic. Let the region-based macro μ be a subset of the optimal multi-policy. Define the following,* time-compressed *MDP:* $\tilde{\mathcal{P}}_{\{\mu\}} = (X, A \cup \{\mu\}, p, \tilde{c}, \gamma)$, *where* $\tilde{c}(x,a,y) = 1$ *for all* $a \in A \cup \{\mu\}$, *unless* $x \in G$ *when* $\tilde{c}(x,a,y) = 0$. *In other words, in* $\tilde{\mathcal{P}}$ *the total cost of a policy is the expected time to reach the goal set* G, *where the execution of the macro* μ *costs a unit-time. Then value iteration, when initialized high, will finish in* $\|V^*_{\tilde{\mathcal{P}}_{\{\mu\}}}\|$ *time-steps in* $\mathcal{P}_{\{\mu\}}$, *i.e., the total planning speed-up is* $\|V^*_{\mathcal{P}}\| - \|V^*_{\tilde{\mathcal{P}}_{\{\mu\}}}\|$.

The proof is elementary but is rather tedious and thus we omit it.

Let $\pi^*_i = \pi_{\mathcal{P}_{\tau_i}}$, $i = 1, \ldots, N$. By the virtue of the above reasoning the interesting macros come from $\mathcal{B} = \{\mu : \mu \text{ is a macro and } \mu \subseteq \pi^*_i \text{ for some } i\}$. Equally, for any macro μ we may count the number of optimal policies which supersede it. We denote this number by $F_N(\mu)$ and prefer macros μ with large $F_N(\mu)$. We will view $\mathcal{B}$ as ordered by $F_N(\mu)$. It turns out that it is not sufficient to look at maximal elements of $\mathcal{B}$ since this rules out too many macros. In order to reintroduce diversity one may look at the maximal elements of $\mathcal{B}_{(x,a)} = \{\mu \in \mathcal{B} : x \in \mathrm{Dom}(\mu), \mu(x) = a\}$. These elements are denoted by $\pi_{(x,a)}$ and are defined by

$$\pi_{(x,a)} = \cap \mathcal{B}_{(x,a)}.$$

This set is non-empty if $\mathcal{B}_{(x,a)}$ is non-empty. Let π_S with $S \subseteq I$ be defined by

$$\pi_S = \cap_{i \in S} \pi_i^*.$$

In practice, one computes $\pi_{(x,a)}$ by first computing the index-sets $S(x,a) = \{i \in I : a \in \pi_i^*(x)\}$ and then exploiting the relationships $\pi_{(x,a)}(y) = \pi_{S(x,a)}(y) = \{a' : S(x,a) \subseteq S(y,a')\}$ that holds for all x, a and y.

The proposed algorithm computes $\pi_{(x,a)}$ for all pairs (x,a) and then computes the set-theoretic closure of

$$M_0 = \{\pi_{(x,a)} : (x,a) \in X \times A\}$$

under the intersection operator. Although this closure operation could in theory produce an exponential blow-up of macros, in our experiments it usually added only a few (but useful) macros. In practice, one would stop the recursive generation of the closure when the limit on the available memory is reached.

After computing this closure we tailor every macro as follows. Given a macro μ we delete those states from the domain of the macro that are elements of both the set source-states of the macro and the set $\partial^- \mu$.

Note that other interesting operations are also possible on M_0. Specifically, the closure of M_0 under intersection and difference produces a very rich set of candidates from which one could select (using e.g. a local search procedure) those few macros which yield the best speed-up while satisfying the size constraints. The subtraction operator is important as, together with a suitable selection algorithm, it enables one to substantially decrease the average branching factor per state while retaining most of the information.

One further possibility would be to identify cutting nodes in the Venn-diagram of M_0 (the directed graph of the lattice $(M_0, \subseteq)$).[11] This would clearly amount to identifying critical points in the original MDP: tasks that define the policies above and below of the cutting node $\pi_{(x,a)}$ are separated by $\pi_{(x,a)}$. We have computed the set-theoretic closure of these cutting nodes for the well-known 4-rooms environment of [11] and obtained exactly the macros that take from one door to the other door of the same room (for all rooms). The environment is shown in Figure 2.

In other cases, e.g. for an environment composed of long corridors no cutting point exists. For such an environment the closure of the largest elements

[11] A cutting node in a directed graph $G = (V, E)$ is a node $v \in V$ such that the number of components of $G \setminus \{v\}$ is one more larger than the number of components in G.

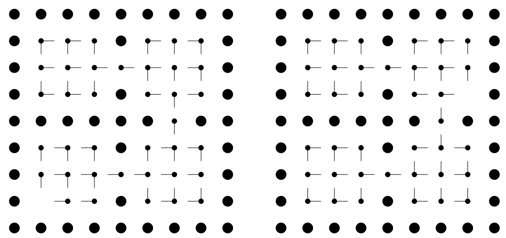

Figure 2: **An example environment.** A 4-room environment, used in the computer experiments is shown with the solution of two distinct tasks. Note that the optimal solutions are shown as multi-policies, i.e., all the optimal actions are shown for all the states. .

of M_0 under intersection gives a good base-set of macros. Figure 3 below shows some macros found by the algorithm for such an environment. Note that the algorithm found some other macros, including the symmetric versions of the macros shown in the figure.

Some further results for the simple 4-room deterministic grid world similar to that of considered in [11] are shown in Figure 4. The left-hand-side subfigure shows the average planning speed-up rate (i.e. how many times faster was the planning when macros were used) and its standard deviation as a function of the number of random tasks. The error bars are obtained by computing the standard deviations for 10 independent runs. As it can be seen, the found macros speed-up planning from numbers beginning from 2 and up to times 4.5. Some nice macros found by the algorithm are shown in Figure 5.

We have also tried an on-line experiment when tasks were added randomly one at a time. The theoretical expected planning speed-up was computed in each time-step by an exhaustive procedure. Results can be seen on the right-hand-side subfigure of Figure 4. Note that despite these results look very encouraging we observed that (presumably because of the increased branching factor) the overall planning time measured e.g. in seconds increased after an initial period of decay. In the future we plan to modify the algorithm to compensate for this, e.g., by using more sophisticated planning methods. In the case of navigation-like tasks one could clearly employ state-abstraction techniques to enhance the speed-up. In many runs the algorithm resulted in macros corresponding to ones that one would design by hand, except that people would often consider overly small macros (see Prop. 4.2).

The algorithm was also tried in stochastic grid worlds but since value it-

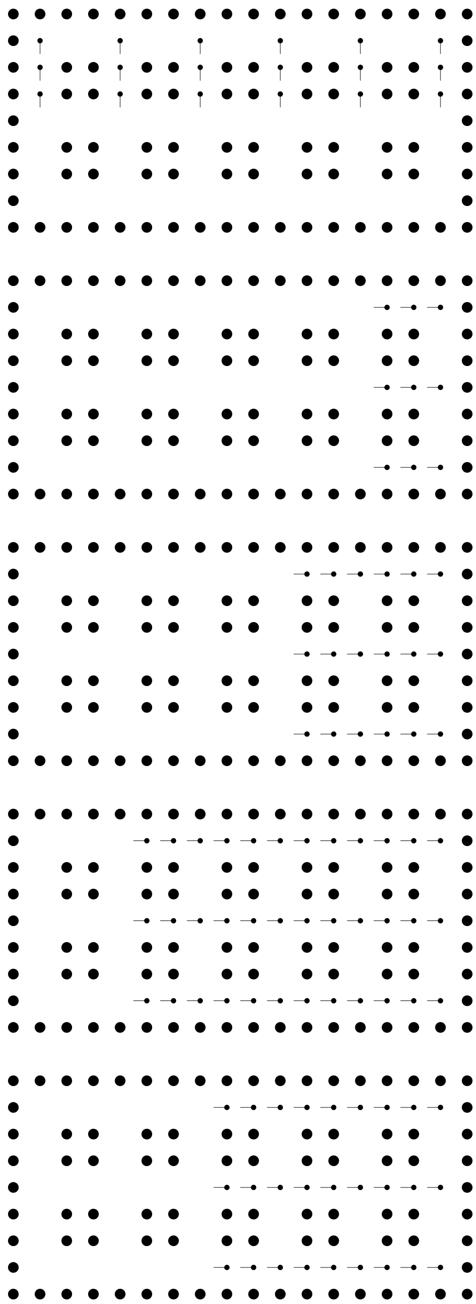

Figure 3: **Some macros found for a "long-corridor"environment.**

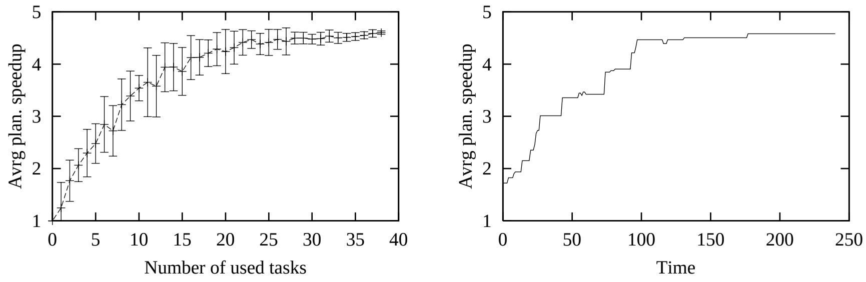

Figure 4: **Measures of the planning speed-up** In the left figure, the planning speed-up rate is shown as a function of the number of random tasks inputted to the algorithm. In the right figure the planning speed-up rate is shown as a function of time for the on-line learning case when in each time step a random task is added to the list of known tasks. .

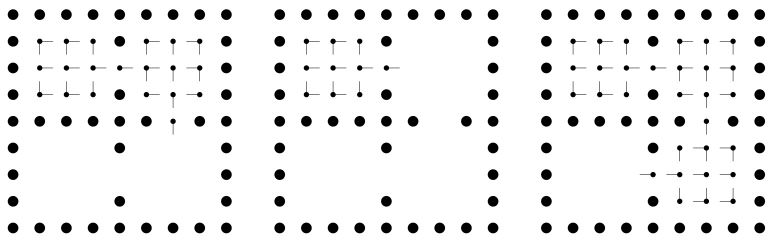

Figure 5: **Some nice macros for the 4-room environment.**

eration runs much slower in stochastic grid worlds and since by inspection we did not found any differences between the macro-sets found in the deterministic and stochastic worlds, we have chosen to present results for deterministic grid-worlds first. [12]

[12]The main reason for this was that in small deterministic worlds we can compute in a reasonable time the *theoretical value* of the expected planning speed-up. Here we wanted to avoid problems related to measuring performance and therefore we considered the choice of deterministic grid-worlds as an acceptable compromise. Results for stochastic worlds are under way.

5 Discussions

5.1 Representational Issues

In any realistic situation, the MDP-skeleton is too big for direct enumeration. The methods of the paper should be extended to cope with this problem. There are several ways to handle this. Firstly, instead of enumerating each state-action pair one could select a subset of them by means of a (cleverly biased) Monte-Carlo selection procedure. Secondly, if the MDP is given in a structured representation (e.g. a Bayesian networks, or many sorted first- or higher-order formulas, or a combination of these) then one can make use of this representation in the computations. Assume that an MDP representation is defined on a given language. By a language here we mean a syntactic construction, i.e. some symbols, connectives and rules for composing formulas out of these. The role of the language is to give a bias to approximate the various quantities (such as the dynamics, the cost structure, the policies, the macros, value functions, etc.) of MDPs. The language could be an extension of the first order logic and/or situation calculus and the algorithms could make use of the various forms of the usual deduction procedures developed in model-theory. However, in our approach formulas would represents statements about the (given) MDP, i.e. formulas were used for approximation purposes. This view enables to feed back observations of the MDP in the extension of the language, thus making it possible to introduce *non-conservative extensions* of the language. Function approximators and Bayesian networks are just particular examples of simple languages (in the case of function approximators defined over the reals the language would include the set of real numbers as constants with a fixed denotation).

Presumably, sets, macros, etc. were represented by formulas of the language. Then, since the language may have some constructs to represent set theoretic operations (e.g. the "and" operation for intersections), the proposed algorithm (or a similar more sophisticated algorithm) could be transformed to make use of that language. Most notably, if a powerful representation is used then this could boost-up the learning process. Interestingly, one can also feed back the macro-learning process to refine the terms and relations of the language. For example, one might try to learn formulas representing various *invariances over the tasks*: A possible learning task is to find the map $f : X \to X$ under which two (or more) macros become unified:

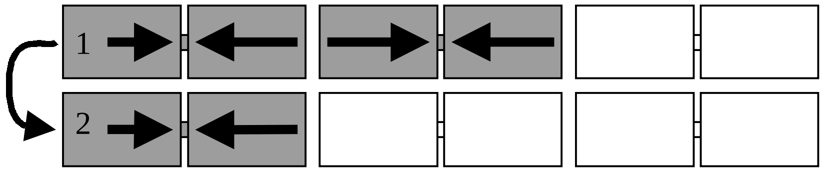

Figure 6: **A 12-room environment.** For an explanation of the figure see the text.

$\mu_i(x) = \mu_j(f(x))\ \forall x \in \mathrm{Dom}(\mu_i)$. Such a mapping could be exploited when solving subsequent tasks or to guess solutions of unseen tasks. This idea can be further extended to the actions themselves. In that case we are looking for a pair of mappings (f,g), $f : X \to X$ and $g : A \to A$ such that $\mu_i(x) = g(\mu_j(f(x)))$. This would enable us to capture very general invariancies.

As an example, consider the following language that is an extension of situation calculus. The language describes 2D navigation tasks. The actions are represented by the parametrized action 'go(direction)', where 'direction' ranges over the directional constants 'left', 'right', 'up', 'down'. The functional fluents of the language are 'x-pos(s)', 'y-pos(s)' and a relational fluent is 'wall-at(s,direction)'. Here 's' denotes a situation. The action precondition axiom for 'go(direction)' is '!wall-at(s,direction)'. The effect axioms are '(direction=left) ⇒ x-pos(do(go(direction), s)) = x-pos(s)+1', '(direction=right) ⇒ x-pos(do(go(direction), s)) = x-pos(s)-1', '(direction=up) ⇒ y-pos(do(go(direction), s)) = y-pos(s)-1' and '(direction=down) ⇒ y-pos(do(go(direction), s)) = y-pos(s)+1'.

Consider the environment shown in Figure 6 and assume that the tasks are the navigation tasks. Assume that the agent has solved the navigation tasks corresponding to the shaded area. Assume further that the agent is able identify the unification function f for the macros defined over area 1 and 2. Then f maps the upper half to the lower half (e.g. $x-pos(f(s)) = x-pos(s)$ and $y-pos(f(s)) = -y-pos(s))$) and thus the agent may introduce a fluent defined by '$c(s) \equiv f(s) \in f(\mathrm{Dom}(\mu_1))$', where μ_1 is the macro for area 1. Human would name c to 'upper-half', but of course, an agent does not have a preference for any string over the other. The complement of c could also be defined. Further, the mapping f could be used to generalize the learnt macros to the unexplored lower half quite easily. Similarly, the agent could learn the concept of gates by abstracting away the properties of interface-states of the macros. One definition would just list the gates, whilst another definition, which is the most general one would define gates to be '$at - wall(s, up) \cap at - wall(s, down) \cap !at - wall(s, left) \cap !at - wall(s, right)$'. Other, more involved examples can be constructed easily.

Of course, there remains the question of how does one come up with a language that already fits well a given domain. In practice, this does not seem to be a problem, but in theory the learning of such a language "from scratch" seems to be hard. As we have just seen macro-learning might help a bit, but still, learning a language seems to hard.

5.2 Factorization of MDPs

A related problem is to factorize an MDP, i.e., given an MDP find a minimal Bayesian network representation of it. It turns out that this problem can be translated to the graph isomorphism problem, a well-known problem in discrete optimization that is widely believed to be NP-hard (note that this problem is not known to be NP-hard). The proof of the subsumption goes as follows: Assume a single action, deterministic MDP, whose state-transitions (dynamics) are given by the function $f : \hat{X} \times \hat{Y} \to \hat{X} \times \hat{Y}$. Let the set of states Z be defined by $Z = \hat{X} \times \hat{Y}$ and assume that $f(x, y) = (g(x), y)$ for some $g : \hat{X} \to \hat{X}$. Let us assume that the learning agent has access to the state-transition graph $G = (Z, E)$ (e.g. after sufficient exploration), where $E = \{(z, f(z)) : z \in Z\}$. Note that if the second components of two states $z = (x, y)$ and $z' = (x', y')$ are different then they are in different components of G. Since arbitrary graph-pairs can be represented as state-transition graphs, checking if a particular factorization is possible is at least as hard as the graph isomorphism problem. Note that factorizing a state-transition graph is different from checking if a particular factorization is possible. Therefore the hardness of the factorization problem remains to be an open question, although we conjecture that this problem is hard.

5.3 Finding Sub-tasks in Flat MDPs and POMDPs

Finally, we note that breaking up a single MDP into a multi-task MDP is not an easy problem either. Assuming a "flat" representation, this problem is related to the hidden state problem. Indeed, given a multi-task MDP, let us define an MDP with its state composed of the state of the multi-task MDP's MDP-skeleton and the task index: $\tilde{X} = X \times I$. Further, let us assume that the task-index is unobservable. Then the multi-task MDP becomes a monolithic partially observed MDP. Therefore, finding a task-structure in a monolithic MDP is similar to reconstructing the state in a POMDP. Again, a structured representation having the right bias might make this task easier.

6 Conclusion

In this paper we proposed a utility of macro-sets defined for multi-task MDPs. We view the results of this paper as a first step towards the systematic exploration of issues related to learning of macros and macro-sets. One promising line of research for future work is to consider hierarchical state-abstraction based approaches together with macro learning. From the theoretical point of view, the most interesting problems are the construction of consistent macro-learning algorithms and tractability questions related to the macro-learning task.

References

[1] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition.* Applications of Mathematics: Stochastic Modelling and Applied Probability. Springer-Verlag New York, 1996.

[2] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[3] M. Hauskrecht. Planning with temporally abstract actions. Cs-98-01, Brown University, Providence, 1998.

[4] M. Hauskrecht, C. Boutilier, N. Meuleau, L.P. Kaelbling, and T. Dean. Hierarchical solution of Markov decision processes using macro-actions. In *Proc. of 14th UAI*, pages 220–229. Morgan Kaufman, 1998.

[5] L. P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, MA, 1993. Morgan Kaufmann.

[6] 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).

[7] R.E. Korf. *Learning to solve problems by searching for macro-operators.* Pitman Publisher, Massachusetts, 1985.

[8] R.E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[9] P. Langley. *Elements of Machine Learning.* MOrgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[10] 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.

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

[12] 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.

[13] M.L. Puterman. *Markov Decision Processes — Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, 1994.

[14] J.W. Shavlik. Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning*, 5:39–70, 1990.

[15] P. Shell and J.G. Carbonell. Towards a general framework for composing disjunctive and iterative macro-operators. In *Eleventh International Joint Conference on Artificial Intelligence*, pages 596–602. Detroit, MI: Morgan Kaufmann, 1989.

[16] S.P. Singh. Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Proceedings of the Ninth International*

Conference on Machine Learning, pages 406–415, Aberdeen, Scotland, UK, 1992. Morgan Kaufmann.

[17] R.S. Sutton. TD models: Modeling the world at a mixture of time scales. In *Proc. of the 12th Int. COnf. on Machine Learning*. Morgan Kaufmann, 1995.

[18] Cs. Szepesvári. Reinforcement learning: Theory and practice. In *Proceedings of the 2nd Slovak Conference on Artificial Neural Networks (SCANN'98)*, 1998.

[19] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, Hillsdale, NJ, 1993. Lawrence Erlbaum.

[20] K. VanLehn. Learning one subprocedure per lesson. *Artificial Intelligence*, 31:1–40, 1987.