Learning State and Action Hierarchies for Reinforcement Learning Using Autonomous Subgoal Discovery and Action-Dependent State Space Partitioning

Mehran Asadi and Manfred Huber
{asadi, huber}@cse.uta.edu

Technical Report CSE-2005-12
Learning State and Action Hierarchies for Reinforcement Learning Using Autonomous Subgoal Discovery and Action-Dependent State Space Partitioning

Mehran Asadi and Manfred Huber
The University of Texas at Arlington
Arlington, TX 76019
{asadi,huber}@cse.uta.edu

Abstract—This paper presents a new method for the autonomous construction of hierarchical action and state representations in reinforcement learning, aimed at accelerating learning and extending the scope of such systems. In this approach, the agent uses information acquired while learning one task to discover subgoals for similar tasks. The agent is able to transfer knowledge to subsequent tasks and to accelerate learning by creating useful new subgoals and by off-line learning of corresponding subtask policies as abstract actions (options). At the same time, the subgoal actions are used to construct a more abstract state representation using action-dependent state space partitioning. This representation forms a new level in the state space hierarchy and serves as the initial representation for new learning tasks. In order to ensure that tasks are learnable, value functions are built simultaneously at different levels of the hierarchy and inconsistencies are used to identify actions to be used to refine relevant portions of the abstract state space. Together, these techniques permit the agent to form more abstract action and state representations over time. Experiments in deterministic and stochastic domains show that the presented method can significantly outperform learning on a flat state space representation.

Index Terms—Machine Learning, Reinforcement Learning, Markov Decision Process.

I. INTRODUCTION

REINFORCEMENT learning (RL) [1], [2] is an active area of machine learning research that is also receiving attention from the fields of decision theory, operations research, and control engineering. RL algorithms address the problem of how an agent can learn to approximate an optimal behavioral strategy while interacting directly with its environment. In control terms, this involves the off-line approximation of solutions to stochastic optimal control problems, usually under conditions of incomplete knowledge of the system being controlled. Most RL algorithms adopt standard methods of stochastic dynamic programming (DP) so that they can be used on-line for problems with large state spaces. By focusing computational effort along behavioral trajectories and by using function approximation methods for accumulating value function information, RL algorithms have produced good results on problems that pose significant challenges for standard methods [3], [4], [5], [6]. However, current RL methods by no means completely circumvent the curse of dimensionality, i.e., the exponential growth of the number of parameters to be learned with the size of any compact encoding of system state.

Recent attempts to combat the curse of dimensionality have turned to principled ways of exploiting temporal abstraction, where decisions are not required at each step, but rather invoke the execution of temporally extended activities which follow their own policies until termination. This leads naturally to hierarchical control architectures and hierarchical learning algorithms (HRL).

Artificial intelligence researchers have addressed the need for large-scale planning and problem solving by introducing various forms of abstraction into problem solving and planning systems [7], [8], [9]. Abstraction allows a system to ignore details that are irrelevant for the task at hand. One of the simplest types of abstraction is the idea of a “macro operator,” or just a “macro,” which is a sequence of operators or actions that can be invoked by name as if it were a primitive operator or action. Macros form the basis of hierarchical specifications of operator or action sequences because macros can include other macros in their definitions: a macro can “call” other macros. Also familiar is the idea of a subroutine that can call other subroutines as well as execute primitive commands. Most of the current research on hierarchical RL focuses on action hierarchies that follow roughly the same semantics as hierarchies of macros or subroutines.

One of the fundamental steps toward HRL is to automatically establish subgoals. Since subgoals can be treated as a termination states for macro actions. Methods for automatically introducing subgoals have been studied in the context of adaptive production systems, where subgoals are created based on examinations of problem-solving protocols. For RL systems, several researchers have proposed methods by which policies learned for a set of related tasks are examined for commonalities or are probabilistically combined to form new policies. Subgoal discovery has been addressed by several researchers [10], [11], [12]. The most closely related research is that of Digney [11] where states that are visited frequently or states where the reward gradient is high are chosen as
subgoals.
The presented work introduces a new method for the autonomous construction of hierarchical actions and state representations in reinforcement learning. In this approach, the agent uses information acquired while learning one task to discover subgoals for similar tasks by analyzing the learned policy using Monte Carlo sampling. The agent is able to transfer knowledge and to accelerate learning of subsequent tasks by creating new subgoals and by off-line learning corresponding subtask policies as abstract actions (options). At the same time, the subgoal actions are used to construct a more abstract state representation using action-dependent state space partitioning. This representation forms a new level in the state space hierarchy and serves as the initial representation for new learning tasks. To ensure that tasks are learnable, value functions are built at different levels of hierarchy and inconsistencies are used to identify actions to be used to refine relevant portions of the abstract state space. Figure 1 illustrates these procedures.

II. MARKOV DECISION PROCESS

Most RL research is based on the formalism of Markov decision processes (MDPs). Although RL is by no means restricted to MDPs, this discrete-time, countable (in fact, usually finite) state and action formalism provides the simplest framework in which to study basic algorithms and their properties. Here we briefly describe this well-known framework, with a few twists characteristic of how it is used in RL research; additional details can be found in many references (e.g., Bertsekas [13], Bertsekas and Tsitsiklis [1], Puterman [14], Ross [15] and Sutton and Barto [2]). A finite MDP models the following type of problem:

At each stage in a sequence of stages, an agent (the controller) observes a system’s state $s$, contained in a finite set $S$, and executes an action (control) $a$ selected from a finite, non-empty set, $A$, of admissible actions. The agent receives an immediate reward having expected value $R(s,a)$, and the state at the next stage is $s'$ with probability $P(s'|s,a)$. The expected immediate rewards, $R(s,a)$, and the state transition probabilities, $P(s'|s,a)$, $s,s' \in S$, together comprise what RL researchers often call the one-step model of action $a$. A (stationary, stochastic) policy $\pi : S \times A \rightarrow [0,1]$, specifies that
the agent executes action \( a \) with probability \( \pi(s, a) \) whenever it observes state \( s \).

For any policy \( \pi \) and \( s \in S \), \( V^\pi(s) \) denotes the expected infinite-horizon discounted return from \( s \) given that the agent uses policy \( \pi \). Letting \( s_t \) and \( r_{t+1} \) denote the state at stage \( t \) and the immediate reward for acting at stage \( t + 1 \) this is defined as:

\[
V^\pi(s) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, \pi\}
\]

where \( \gamma \), \( 0 \leq \gamma \leq 1 \), is a discount factor. \( V^\pi(s) \) is the value of \( s \) under \( \pi \), and \( V^\pi \) is the value function corresponding to \( \pi \).

The objective is to find an optimal policy, i.e., a policy, \( \pi^* \), that maximizes the value of each state. The unique optimal value function, \( V^* \), is the value function corresponding to any of the optimal policies.

Most RL research addresses discounted MDPs since they comprise the simplest class of MDPs, and here we restrict attention to discounted problems. However, RL algorithms have also been developed for MDPs with other definition of return, such as average reward MDPs [16]. Playing important roles in many RL algorithms are action-value functions, which assign values to admissible state-action pairs. Given a policy \( \pi \), the value of \((s, a)\), denoted \( Q^\pi(s, a) \), is the expected infinite-horizon discounted return for executing \( a \) in state \( s \) and thereafter following \( \pi \):

\[
Q^\pi(s, a) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, a_t = a, \pi\}
\]

The optimal action-value function, \( Q^* \), assigns to each admissible state-action pair \((s, a)\) the expected infinite-horizon discounted return for executing \( a \) in \( s \) and thereafter following an optimal policy. Action-value functions for other definition of return are defined analogously.

Dynamic programming (DP) algorithms exploit the fact that value functions satisfy various Bellman equations, such as:

\[
V^\pi(s) = \sum_{a \in A} \pi(s, a) \left[ R(s, a) + \gamma \sum_{s'} P(s' | s, a) V^\pi(s') \right]
\]

and

\[
V^*(s) = \sum_{a \in A} \pi(s, a) \left[ R(s, a) + \gamma \sum_{s'} P(s' | s, a) V^*(s') \right]
\]

for all \( s \in S \). Analogous equations exists for \( Q^\pi \) and \( Q^* \). For example, the Bellman equation for \( Q^* \) is:

\[
Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s' | s, a) \max_{a' \in A} Q^*(s', a')
\]

for all \( s \in S \) and \( a \in A \).

### III. REINFORCEMENT LEARNING

DP algorithms have complexity polynomial in the number of states, but they still require prohibitive amounts of computation for large state sets, such as those that result from discretizing multi-dimensional continuous spaces or representing state sets consisting of all possible configurations of a finite set of structural elements (e.g., possible configurations of a backgammon board [6]). Many methods have been proposed for approximating MDP solutions, in particular, RL methods that use Monte Carlo, stochastic approximation, and function approximation techniques. Specifically, RL algorithms combine some, or all, of the following features [17]:

- Avoid the exhaustive sweeps of DP by restricting computation to states on, or in the neighborhood of, multiple sample trajectories, either real or simulated. Because computation is guided along sample trajectories, this approach can exploit situations in which many states have low probabilities of occurring in actual experience.
- Simplify the basic DP backup by sampling. Instead of generating and evaluating all of a state’s possible immediate successors, estimate a backup’s effect by sampling from the appropriate distribution.
- Represent value functions and/or policies more compactly than lookup-table representations by using function approximation methods, such as linear combinations of basis functions, neural networks, or other methods.

The first two features reflect the nature of the approximations usually sought when RL is used. Instead of policies that are close to optimal uniformly over the entire state space, RL methods arrive at non-uniform approximations that reflect the behavior of the agent. The agent’s policy does not need high precision in states that are rarely visited. The last feature is the least understood aspect of RL, but results exist for the linear case (notably in Tsitsiklis and Van Roy [18]) and numerous examples illustrate how function approximation schemes that are nonlinear in the adjustable parameters (e.g., multilayer neural networks) can be effective for difficult problems (e.g., [3], [16], [19], [6]). Of the many RL algorithms, perhaps the most widely used are Q-learning [20], [21] and Sarsa [22].

Q-learning is based on the DP backup (2) but with the expected immediate reward and the expected maximum action-value of the successor state on the right-hand side of Equation 2 replaced by a sample reward and the maximum action-value for a sample next state respectively. The most common way to obtain these samples is to generate sample trajectories by simulation or by observing the actual decision process over time.

### IV. RELATED WORK

Hierarchical approaches to RL generalize the macro idea to closed-loop policies, or more precisely, closed-loop partial policies because they are generally defined for a subset of the state set. The partial policies must also have well-defined termination conditions. These partial policies are sometimes called temporally-extended actions or options [23].

When option policies are learned, they usually are policies for efficiently achieving subgoals, where a subgoal is often a state, or a region of the state space, such that reaching that state or region is assumed to facilitate achieving the overall goal of the task. The canonical example of a useful subgoal is a doorway in a robot navigation scenario: the doorway has to be passed through to reach any goal outside the room.
A. Temporal Abstraction of SMDPs

Artificial intelligence researchers have addressed the need for large-scale planning and problem solving by introducing various forms of abstraction into problem solving and planning systems, e.g., Fikes et al. [7] and Korf [8]. Abstraction allows a system to ignore details that are irrelevant for the task at hand. One of the simplest types of abstraction is the idea of a “macro-operator,” or just a “macro,” which is a sequence of operators or actions that can be invoked by name as if it were a primitive operator or action. Macros form the basis of hierarchical specifications of operator or action sequences because macros can include other macros in their definition: a macro can “call” other macros. Also familiar is the idea of a subroutine that can call other subroutines as well as execute primitive commands. Most of the current research on hierarchical RL focuses on action hierarchies that follow roughly the same semantics as hierarchies of macros or subroutines.

From a control perspective, a macro is an open-loop control policy and, as such, is inappropriate for most interesting control purposes, especially the control of stochastic systems. Hierarchical approaches to RL generalize the macro idea to closed-loop policies, or more precisely, closed-loop partial policies because they are generally defined for a subset of the state set. The partial policies must also have well-defined termination conditions.

These partial policies are sometimes called temporally-extended actions, options [23], skills [24] or behaviors [25], [26]. For MDPs, this extension adds to the sets of admissible actions, \( a \in A \), sets of activities, each of which can itself invoke other activities, thus allowing a hierarchical specification of an overall policy. The original one-step actions, now called the “primitive actions,” may or may not remain admissible. Extensions along these general lines result in decision processes modeled as Semi-Markov Decision Process (SMDP), where the waiting time in a state now corresponds to the duration of the selected activity. If \( \tau \) is the waiting time in state \( s \) upon execution of activity \( a \), then \( \alpha \) takes \( \tau \) steps to complete when initiated in \( s \), where the distribution of the random variable \( \tau \) now depends on the policies and termination conditions of all of the lower-level activities that comprise \( a \).

Sutton et al. [23] formalize this approach to including activities in RL with their notion of an option. Starting from a finite MDP, which we call the core MDP, the simplest kind of option consists of a (stationary, stochastic) policy \( \pi : S \times A \rightarrow [0, 1] \), a termination condition \( \beta : S \rightarrow [0, 1] \), and an input set \( I \subseteq S \). The option \( (I, \pi, \beta) \) is available in state \( s \) if and only if \( s \in I \). If the option is executed, then actions are selected according to \( \pi \) until the option terminates stochastically according to \( \beta \). For example, if the current state is \( s \), the next action is \( a \) with probability \( \pi(s, a) \), the environment makes a transition to state \( s' \), where the option either terminates with probability \( \beta(s) \) or else continues, determining the next action \( a' \) with probability \( \pi(s', a') \), and so on. When the option terminates, the agent can select another option.

It is usual to assume that for any state in which an option can continue, it can also be initiated, that is, \( \{ s : \beta(s) < 1 \} \subset I \). This implies that an option’s policy only needs to be defined over its input set \( I \). Note that any action of the core MDP, a primitive action \( a \in A \), is also an option, and it is called a one-step option, with \( I = \{ s : a \in A \} \) and \( \beta(s) = 1 \) for all \( s \in S \). Sutton et al. [23] give the example of an option named open-the-door for a hypothetical robot control system. This option consists of a policy for reaching, grasping and turning the door knob, a termination condition for recognizing that the door has been opened, and an input set restricting execution of open-the-door to states in which a door is within reach.

To allow more flexibility, especially with respect to hierarchical architectures, one must include semi-Markov options whose policies can set action probabilities based on the entire history of states, actions, and rewards since the option was initiated [23]. Semi-Markov options include options that terminate after a pre-specified number of time steps, and most importantly, they are needed when policies over options are considered, i.e., policies \( \pi : S \times O_s \rightarrow [0, 1] \), where \( O_s \) is the set of admissible options for state \( s \) (which can include all the one-step options corresponding to the admissible primitive actions in \( s \)). A policy \( \pi \) over options selects option \( o \) in state \( s \) with probability \( \pi(o, s) \); \( o \)'s policy in turn selects other options until \( o \) terminates. The policy of each of these selected options selects other options, and so on. Expanding each option down to primitive actions, we see that any policy over options, \( \pi \), determines a conventional policy of the core MDP, which Sutton et al. [23] call the flat (i.e., non-hierarchical) policy corresponding to \( \pi \), denoted \( \text{flat}(\pi) \). Flat policies corresponding to policies over options are generally not Markov even if all the options are Markov. The probability of a primitive action at any time step depends on the current core state plus the policies of all the options currently involved in the hierarchical specification. This dependence is made more explicit in the work of Parr [27] and Dietterich [5]. Using this machinery, one can define hierarchical options as triples \( (I, \pi, \beta) \), where \( I \) and \( \beta \) are the same as for Markov options but \( \pi \) is a semi-Markov policy over options.

Value functions for option policies can be defined in terms of value functions of semi-Markov flat policies. For a semi-Markov flat policy \( \pi \):

\[
V^\pi(s) = E\{r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^\tau - 1 r_{t+\tau} + \ldots | \varepsilon(\pi, s, t)\}
\]

where \( \varepsilon(\pi, s, t) \) is the event of \( \pi \) being initiated at time \( t \) in \( s \). Note that this value can depend on the complete history from \( t \) onwards, but not on events earlier than \( t \) since \( \pi \) is semi-Markov. Given this definition for flat policies, \( V^{\text{flat}(\pi)}(s) \), the value of \( s \) for a policy \( \pi \) over options, is defined to be \( V^{\pi}(s) \).

Similarly, one can define the option-value function for \( \pi \) as follows:

\[
Q^\pi(s, o) = E\{r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^\tau - 1 r_{t+\tau} + \ldots | \varepsilon(o_\pi, s, t)\}
\]

where \( o_\pi \) is the semi-Markov policy that follows \( o \) until it terminates after \( t \) time steps and then continues according to \( o_\pi \).

Adding any set of semi-Markov options to a core finite MDP yields a well-defined discrete-time SMDP whose actions are
the options and whose rewards are the returns delivered over
the course of an option’s execution. Since the policy of each
option is semi-Markov, the distributions defining the next state
(the state at an option’s termination), waiting time, and rewards
depend only on the option executed and the state in which its
execution was initiated. Thus, all of the theory and algorithms
applicable to SMDOs can be appropriated for decision making
with options.

In their effort to treat options as much as possible as if
they were conventional single-step actions, Sutton et al. [23]
introduced the interesting concept of a multi-time model of
an option that generalizes the single-step model consisting of
\( R(s, a) \) and \( P(s'|s, a) \), \( s, s' \in S \), of a conventional action
\( a \). For any option \( o \), let \( \varepsilon(o, s, t) \) denote the event of \( o \) being
initiated in state \( s \) at time \( t \). Then the reward part of the multi-
time model of \( o \) for any \( s \in S \) is:

\[
R(s, o) = E\{r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{t-1} r_{t+r} + \ldots | \varepsilon(o, s, t) \}
\]  

(3)

where \( t + r \) is the random time at which \( o \) terminates. The
state-prediction part of the model of \( o \) for \( s \) is:

\[
F(s'|s, o) = \sum_{k=1}^{\infty} P(s_{t+k} = s'|s_t = s, o) \gamma^k
\]  

(4)

for all \( s \in S \). Though not itself a probability, \( F(s'|s, o) \) is a
combination of the probability that \( s' \) is the state in which \( o \)
terminates together with a measure of how delayed that
outcome is in terms of \( \gamma \). The quantities \( R(s, o) \) and \( F(s'|s, o) \)
respectively generalize the reward and transition probabilities,
\( R(s, a) \) and \( P(s'|s, a) \), of the usual MDP in such a way that
one can write a generalized form of the Bellman optimality
equation. If \( V^*_o \) denotes the optimal value function over an
option set \( o \), then

\[
V^*_O(s) = \max_{o \in O} [R(s, o) + \sum_{s} F(s'|s, o)V^*_o(s')]
\]

which reduces to the usual Bellman optimality Equation 1,
if all the options are one-step options \( \beta(s) = 1, s \in S \). A
Bellman equation can be written for \( Q^*_o \):

\[
Q^*_O(s, o) = R(s, o) + \sum_{s} F(s'|s, o) \max_{o' \in O} Q^*_o(s', o')
\]

for all \( s \in S \) and \( o \in O \).

The DP backup analogous to Equation 2 for computing option-
values is:

\[
Q_{k+1}(s, o) = R(s, o) + \max_{s' \in S} \sum_{o' \in O} Q_k(s', o')
\]

and the corresponding Q-learning update is:

\[
Q_{k+1}(s, o) = (1 - \alpha_k) Q_k(s, o) + \alpha_k [r + \gamma \max_{o' \in O} Q_k(s', o')]
\]

This update is applied upon the termination of \( o \) at state \( s' \)
after executing for \( t \) time steps, and \( r \) is the return accumulated
during \( o \)'s execution.

The primary motivation for the options framework is to permit
one to add temporally-extended activities to the repertoire of
choices available to an RL agent, while at the same time not
precluding planning and learning at the finer grain of the core
MDP. The emphasis is therefore on simplification rather than
simplification of the core MDP. If all the primitive actions
remain in the option set as one-step options, then clearly the
space of realizable policies is unrestricted so that the optimal
policies over options are the same as the optimal policies for
the core MDP. But since finding optimal policies in this case
takes more computation via conventional DP than does just
solving the core MDP, one is tempted to ask what one gains
from this augmentation of the core MDP. One answer is to be
found in the use of RL methods. For RL, the availability of temporally-extended activities can dramatically improve
the agent’s performance while it is learning, especially in the
initial stages of learning. Options also can facilitate transfer
of learning to related tasks. Of course, only some options can
facilitate learning in this way, and a key question is how does
a system designer decide on what options to provide.

On the other hand, if the set of options does not include the
one-step options corresponding to all of the primitive actions,
then the space of policies over options is a proper subset of the
set of all policies of the core MDP. In this case, the resulting
SMDP can be much easier to solve than the core MDP [17].

B. Subgoal Discovery

In the current state-of-the-art, the designer of an RL system
typically uses prior knowledge about the task to add a speci-
cific set of options to the set of primitive actions available
to the agent. In some cases, complete option policies can be
provided; in other cases, option policies can be learned using,
for example, intra-option learning methods together
with option-specific reward functions that are provided by the
designer [10], [11], [12].

Providing options and their policies a priori is an opportunity
to use background knowledge about the task to try to accelerate
learning and/or provide guarantees about system performance
during learning. Perkins and Barto [28], for example,
consider collections of options each of which descends on a
Lyapunov function. Not only is learning accelerated, but the
goal state is reached on every learning trial while the agent
learns to reach the goal more quickly by approximating a
minimum-time policy over these options.

When option policies are learned, they usually are policies
for efficiently achieving subgoals, where a subgoal is often a
state, or a region of the state space, such that reaching that
state or region is assumed to facilitate achieving the overall
goal of the task. The canonical example of a useful subgoal is
a doorway in a robot navigation scenario: the doorway has
a goal state is reached on every learning trial while the agent
learns to reach the goal more quickly by approximating a
minimum-time policy over these options.

When option policies are learned, they usually are policies
for efficiently achieving subgoals, where a subgoal is often a
state, or a region of the state space, such that reaching that
state or region is assumed to facilitate achieving the overall
goal of the task. The canonical example of a useful subgoal is
a doorway in a robot navigation scenario: the doorway has
to be passed through to reach any goal outside the room.
Given a collection of subgoals, one can define subgoal-specific
reward functions that positively reward the agent for achieving
the subgoal (while possibly penalizing it until the subgoal is
achieved).

Options are then defined which terminate upon achieving a
subgoal, and their policies can be learned using the subgoal-
specific reward function and standard RL methods. Precup [23]
discusses one way to do this by introducing subgoal values,
and Dietterich [5], proposes a similar scheme using pseudo-
reward functions.
A natural question, then, is how are useful subgoals determined? McGovern and Barto [10] developed a method for automatically identifying potentially useful subgoals by detecting regions that the agent visits frequently on successful trajectories but not on unsuccessful trajectories. An agent using this method selects such regions that appear early in learning and persist throughout learning, creates options for achieving them and learns their policies, and at the same time learns a higher-level policy that invokes these options appropriately to solve the overall task. Experiments with this method suggest that it can be useful for accelerating learning on single tasks, and that it can facilitate knowledge transfer as previously-discovered options are reused in related tasks. This approach builds on previous work in artificial intelligence that addresses abstraction, particularly that of Iba [29], who proposed a method for discovering macro-operators in problem solving. Related ideas have been studied by Digney [11] where states that are visited frequently or states where the reward gradient is high are chosen as subgoals.

### C. Approaches to Hierarchical Reinforcement Learning

Parr [27] and Parr and Russell [30] developed an approach to hierarchically structuring MDP policies called Hierarchies of Abstract Machines or HAMs. Like the options formalism, HAMs exploit the theory of SMDPs, but the emphasis is on simplifying complex MDPs by restricting the class of realizable policies rather than expanding the action choices. In this respect, as pointed out by Parr [27], it has much in common with the multilayer approach for controlling large Markov chains that considered a two-layer structure in which the lower level controls the plant via one of a set of predefined regulators. The higher level, the supervisor, monitors the behavior of the plant and intervenes when its state enters a set of boundary states. In the options framework, each option corresponds to a low-level regulator, and when the option set does not contain the one-step options corresponding to all primitive actions, the same simplification results. HAMs extend this idea by allowing policies to be specified as hierarchies of stochastic finite-state machines.

Dietterich [31] developed another approach to hierarchical RL called the MAXQ Value Function Decomposition, which we call simply MAXQ. Like options and HAMs, this approach also relies on the theory of SMDPs. Unlike options and HAMs, however, the MAXQ approach does not rely directly on reducing the entire problem to a single SMDP. Instead, a hierarchy of SMDPs is created whose solutions can be learned simultaneously [17].

The most strongly related work is the method introduced by Dean et al. [9] as a mechanism to derive state space partitions of a MDP that ensure approximately optimal policies to be learned. These partitions depend on the action space and the particular reward function of the task. Kim and Dean, [4] introduced an algorithm to derive a set of such partitions and used it to learn a policy for the task indicated by the reward function. The resulting policy is ensured to be within an $\epsilon$-dependent quality bound. Two shortcomings of this algorithm is that it does not address temporally abstract actions and requires a complete re-computation of the partitions when a new action is introduced. Moreover, it requires knowledge of the reward function prior to partitioning, and thus no part of the partitioning transfers across tasks.

### V. HIERARCHICAL ACTIONS AND STATE SPACE CONSTRUCTION

This section describes the main contribution of this paper. In this approach, the agent uses information acquired while learning one task to discover subgoals for similar tasks. The agent is able to transfer knowledge to subsequent tasks and to accelerate learning by creating useful new subgoals and by offline learning of corresponding subtask policies as abstract actions (options). At the same time, the subgoal actions are used to construct a more abstract state representation using action-dependent state space partitioning. This representation forms a new level in the state space hierarchy and serves as the initial representation for new learning tasks. In order to ensure that tasks are learnable, value functions are built simultaneously at different levels of the hierarchy and inconsistencies are used to identify actions to be used to refine relevant portions of the abstract state space. Together, these techniques permit the agent to form more abstract action and state representations over time.

#### A. Autonomous Subgoal Discovery

In this section we introduce our subgoal discovery algorithm and we show how it can be improved by using Monte Carlo sampling. These discovered subgoals will be used in the next section, in order to build more abstract state space which is action dependent.

An example that shows the importance of a subgoal is a room to room navigation task where the agent should discover the utility of doorways. If the agent discovers that a doorway is a subgoal it can learn an option to reach the doorway which, in turn, can accelerate learning of new navigation tasks. The idea of using subgoals is not, however, limited to grid worlds. For example, for a robot arm to pick up an object, an important subtask is the recognition of the object and thus being aware of its presence would be a subgoal.

The main goal of automatic subgoal discovery is to find useful subgoals in the agent’s state space. Once they are found, options to those subgoals can be learned and added to the behavioral repertoire of the agent. In the approach presented here, subgoals are identified as states with particular structural properties in the context of a given policy. In particular, we define subgoals as states that, under a given policy, lie on a substantially larger number of paths than would be expected by looking at its successor states. In other words, we are looking for states that form a “funnel” for state space trajectories under the learned policy.

**Definition 1** A state $s'$ is a direct predecessor of state $s$, if under a learned policy the action in state $s'$ can lead to $s$ i.e., $P(s'|s, a) > 0$.

**Definition 2** The count metric for state $s$ under a learned policy, $\pi$, is the sum over all possible state space trajectories weighed by their accumulated likelihood to pass through state

$$\text{count}(s, \pi) = \sum_{s' \in S} P(s'|s, a)$$
Let $C^*_\pi(s)$ be the count for state $s$, then:

$$C^*_\pi(s) = \frac{1}{|S|} \sum_{s' \neq s} P(s|s', \pi(s'))$$

where $S$ is a number of states and

$$C^*_\pi(s) = \sum_{s' \neq s} P(s|s', \pi(s')) C^*_\pi(s') C^*_\pi(s) \tag{5}$$

where $n$ is such that $C^*_\pi(s) = C^*_\pi(s) + 1(s)$ or $n = |S|$. The condition $s' \neq s$ prevents the counting of self loops and $P(s|s', \pi(s'))$ is the probability of reaching state $s$ from state $s'$ by executing action $\pi(s')$. The slope of $C^*_\pi(s_t)$ along a path, $\rho$, under policy $\pi$ is:

$$\Delta_\pi(s_t) = C^*_\pi(s_t) - C^*_\pi(s_{t-1})$$

where $s_t$ is the $t^{th}$ state along the path. In order to identify subgoals, the gradient ratio $\Delta_\pi(s_t) / \max(1, \Delta_\pi(s_{t+1}))$ is computed for states where $\Delta_\pi(s_t) > 0$. A state $s_t$ is considered a potential subgoal candidate if the gradient ratio is greater than a specified threshold $\mu > 1$. Appropriate values for this user-defined threshold depend largely on the characteristics of the state space and result in a number of subgoal candidates that is inversely related to the value of $\mu$. This approach is an extension of the criterion in [32] with $\Delta_\pi(s_{t+1})$ addressing the effects of potentially obtaining negative gradients due to nondeterministic transitions. In this approach we focus our attention on states that are regular, i.e., every state has approximately the same expected number of direct predecessors according to a learned policy.

Assuming that there are a small number of subgoals as compared to the size of the space, the theory of the $t$-test is applied to compute a threshold. Based on this assumption it can be stated that the distribution of gradient ratios over the entire space represents approximately the distribution in a space free of subgoals. Another assumption made here is that the gradient ratios at any state are randomly drawn from the cumulative distribution. Given the assumption, it can be tested using a $t$-test whether the gradient ratio distribution at any state belongs to the cumulative distribution and its affirmation means that the state is not a subgoal. To run a $t$-test for a given state $s$, the mean, $\rho$, of a sample of gradient ratios of size $N$ is computed by randomly choosing $N$ trajectories going through state $s$ under the learned policy. Then equation 6 is used to compute the $t$ value. To avoid performing $t$-test at every state a slightly different approach is taken here. Based on desired $p$ value (the probability of obtaining a particular sample result given the null hypothesis) sample mean $\rho$ required to pass the test is computed using formula:

$$t = \frac{\mu - \rho}{\sigma / \sqrt{N}} \tag{6}$$

$\mu$ plays the role of threshold required in this approach because in order for the $t$-test to provide a positive result for the sample distribution at a given state $s$, there should be at least one path through $s$ along which the gradient ratio at state $s$ is equal or higher than $\mu$. In order to reduce the computational complexity of the above method in large state spaces, the gradient ratio is here computed using Monte Carlo sampling.

### Definition 3

Let $H = \{h_1, ..., h_N\}$ be $N$ sample trajectories induced by policy $\pi$, then the sampled count metric, $C_H^*(s)$, for each state $s$ that is on the path of at least one path $h_i$ can be calculated as the average of the accumulated likelihoods of each path, $h_i$, $1 \leq i \leq N$, rescaled by the total number of possible paths in the environment.

If we need to estimate the value of $C^*_\pi(s)$ from independent, identically distributed (i.i.d) sample induced by policy $\pi$, after taking $N$ samples $h_i$, $i \in \{1, \ldots, N\}$ we have:

$$C^*_H(s) = \frac{1}{N} \sum_i C(h_i, s)$$

The expected value of this estimator is $C^*_\pi(s)$.

**Theorem 1 (Bernstein [33])** Let $\xi_1, \xi_2, \ldots$ be independent random variables with means $E\xi_i$, bounded by some constant $E\xi_i \leq a$, $a > 0$. Also let $\text{Var}(M_N) = E\xi^2_1 + \ldots + E\xi^2_N \leq L$ and $Z_N = E\xi_1 + \ldots + E\xi_N$. Then the partial sum $M_N = \xi_1 + \ldots + \xi_N$ obey the following inequality for all $\epsilon_N > 0$:

$$P\left(\frac{1}{N} M_N - \frac{1}{N} Z_N > \epsilon_N\right) \leq 2 \exp\left(-\frac{1}{2L + a\epsilon_N}\right)$$

The following theorem shows that for a large number of samples the difference between $C^*_\pi$ and $C^*_H$ would be smaller than a real value $\epsilon_N$.

**Theorem 2** In a regular space, for sample size

$$N \geq \frac{\max_i C_H^*(s)}{\epsilon^2_N} \cdot 2(1 + \epsilon_N) \log\left(\frac{2}{1 - p}\right) \tag{7}$$

it is true that $|C^*_H(s) - C^*_\pi(s)| \leq \epsilon_N$ with probability $p$.

**Proof:** See appendix.

**Theorem 3** Let $H = \{h_1, ..., h_N\}$ be $N$ sample trajectories induced by policy $\pi$ with $N$ selected according to Equation 7. If

$$\frac{\max_i (\Delta_\pi(s))}{\max(1, \Delta_\pi(s_{t+1}))} > \mu + \frac{2\xi}{\max(1, \Delta_\pi(s_{t+1}))},$$

then $\frac{\Delta_\pi(s)}{\max(1, \Delta_\pi(s_{t+1}))} > \mu$ with probability $\geq p$.

**Proof:** See appendix.

Theorem 3 implies that for a sufficiently large sample size the exhaustive and the sampling method predict the same subgoals with high probability.

1. **Example:** Figure 2(a) shows a two-room example environment on a $10 \times 6$ grid. For this experiment, the goal state is placed in the upper right hand portion (gray cell) and each trial is started from the same state in the lower left corner. The action space consists of eight primitive actions (North, East, South, West, Northwest, Northeast, Southwest and Southeast). The world is deterministic and each action succeeds in moving the agent in the chosen direction. With every action the agent receives a negative reward of $-1$ for a straight action and $-1.2$ for a diagonal action. In addition, the agent receives a reward of $+10$ when it reaches the goal state. Policy $\pi$ is learned using Q-learning and the count metric for every state is computed.
only the discounted reward estimate has to be re-computed for each new learning task.

In the two phase partitioning approach presented here we construct the initial blocks of the partition by distinguishing terminal states (subgoals) for available option from non-terminal states, in order to construct action specific blocks of a partition, and finally these blocks will be refined based on the transition probability function.

Let \( \{s_1, \ldots, s_n\} \) be \( n \) discovered subgoals and \( \{o_1, \ldots, o_n\} \) be the corresponding options. We construct a partition \( P = \{B_1, \ldots, B_n\} \) of state space \( S \) such that each set \( B_i \) contains all states \( s \in I_i \) such that \( F(s_i|s,o_i) > 0, \beta(s_i) = 1 \) and \( \bigcup_{i=1}^n B_i = S \).

**Definition 4** A partition \( P = \{B_1, \ldots, B_n\} \) of the state space of an MDP has approximate stochastic bisimulation homogeneity if and only if for each \( B_i, B_j \in P \) and for each \( s, s' \in B_i \):

\[
\left| \sum_{s'' \in B_j} F(s''|s,o_i(s)) - \sum_{s'' \in B_j} F(s''|s',o_i(s')) \right| \leq \delta \tag{8}
\]

where \( 0 \leq \delta \leq 1 \).

We say that a block \( B_i \) is \( \delta \)-stable with respect to block \( B_j \) if and only if Inequality 8 holds. \( B_i \) is \( \delta \)-stable if \( B_i \) is \( \delta \)-stable with respect to all blocks of \( P \).

To form partitions, each block is constructed according to it's available options and it is checked for \( \delta \)-stability and unstable blocks are split until no unstable blocks remain. When a block \( B_k \) is found to be unstable with respect to block \( B_i \), we replace \( B_k \) by a set of sub-blocks \( B_{k1}, \ldots, B_{km} \) such that \( B_{k1} \) is maximal sub-block of \( B_k \) that is \( \delta \)-stable with respect to \( B_i \). To facilitate modifications in the action space, this process is first performed for each option individually. The blocks of the final partition are then formed by intersecting all blocks for each \( o_i \) that are used followed by a refining stage that achieves \( \delta \)-stability for the intersections \( [34], [35] \). This reduces the overhead required when the action set changes to the intersection and the final refinement step.

If the reward structure becomes available, the second phase of the partitioning technique further refines the partition with the following reward criterion:

\[
|R(s,o_i) - R(s',o_i)| \leq \epsilon \tag{9}
\]

Given a particular subset of options, an appropriate abstract state space representation for the learning task can thus be derived which is stable according to criteria in inequalities 8 and 9. Furthermore, representation changes due to changes in the action set can be performed efficiently and a simple mechanism can be provided to use the previously learned value function as a starting point when such representation changes occur. This is particularly important if actions are added over time to permit refinement of the initially learned policy by permitting finer-grained decisions.

1) **Example:** In this example we assume a grid world with a mobile robot which can perform four primitive deterministic actions: LEFT, RIGHT, UP and DOWN. Rewards for actions that lead the agent to another cell are assumed to be -1 and the state marked with * has a reward of 20. The goal of the agent...
is to find the state marked with *. The bold bars in the grid world represent obstacles for the robot. In order to construct an option we define a policy with each action. The termination condition is hitting the wall and the option repeats each action until it terminates. Figure 3 shows this scenario. Let $\epsilon = 1$ and $\delta = 20$, we can now perform the partitioning using options and $\epsilon, \delta$. Figures 4 shows the possible partitions for the four options. Each partition in these Figures is divided only in two blocks as all the state satisfy the probability criterion but the state * is different from other states in reward criterion. Let $B_i^j$ be the block $j$ for partition $i$ derived by option $o_i$.

This blocks can be combined by intersecting them in order to derive a partition that consists of new blocks that are defined for each option, for example:

$$\begin{align*}
B_1 &= B_1^1 \cap B_2^1 \cap B_3^1 \cap B_4^1 \\
B_2 &= B_2^2 \cap B_4^2 \cap B_5^2 \cap B_6^2
\end{align*}$$

(10)

Figure 5.a illustrates the intersection of the partitions. These blocks form the initial blocks for the reduction technique. The result of refinement is illustrated in Figure 5.b. While performing an action on each state, the result would be another block instead of a state so each block of Figure 5.b can be considered a single state in the resulting state space. Now if the agent is in block $B_1$ and executes the action RIGHT the the agent will arrive in a state in block $B_6$.

C. Learning Method

Let $P = \{B_1, \ldots, B_n\}$ be a partition for state space $S$ derived by the action-dependent partitioning method, using subgoals $\{s_1, \ldots, s_k\}$ and options to these subgoals $\{o_1, \ldots, o_k\}$. If the goal state $G$ belongs to the set of subgoals $\{s_1, \ldots, s_k\}$, then $G$ is achievable by options $\{o_1, \ldots, o_k\}$ and the task is learnable according to Theorem 4. However, if $G \notin \{s_1, \ldots, s_k\}$ then the task may not be solvable using only the options that terminate at subgoals. The proposed approach solves this problem by maintaining a separate value function for the core MDP while learning a new task on the partition space derived from only the subgoal options. During learning, the agent has access to the original actions as well as all options, but makes decisions only based on the abstract partition space information.

**Theorem 4** For any policy $\pi$ for which the goal $G$ can be represented as a conjunction of terminal sets (subgoals) of the available actions in the original MDP $M$, there is a policy $\pi_P$ in the reduced MDP, $M_P$, that achieves $G$ as long as for each state $s_i$ in $M$ for which there exists a path to $G$, there exists a path such that $F(G|s_i, \pi_P(s_i)) > \delta$.

**Proof:** See appendix.

While the agent tries to solve the task on the abstract partition space, it computes the difference in Q-values between the best actions in the current state in the abstract state space and in the original state space. If the difference is larger than a constant value (given by Theorem 5), then there is a significant difference between different states underlying the particular block that was not captured by the subgoal options. This approach is similar to McCallum’s utile distinction memory [36] which combines Hidden Markov Models (HMMs) and Q-learning to solve tasks with only a few fields by splitting “inconsistent” HMM states whenever the agent fails to predict their utilities. In the presented learning method, a similar idea is used which determines inconsistent model states based on Q-value Inconsistencies.

**Theorem 5** Given an MDP $M = (S, A, T, R)$ and a partition $P$ of the state space $M_P$, the optimal value function of $M$ given as $V^*$ and the optimal value function of $M_P$ given as $V_P^*$ satisfy the bound on the distance

$$\| V^* - V_P^* \|_{\infty} \leq 2 \left( 1 + \frac{\gamma}{1 - \gamma} \max \{ \epsilon, \delta \} \right)$$

Let $M = (S, A, P, R)$ be a MDP and $M_P$ be the corresponding reduce MDP with $P = \{B_1, \ldots, B_n\}$, where $B_i$, $1 \leq i \leq n$, are $\delta$-stable (and they are possibly stable according to inequality 9). This method maintains two separate tables for computing $V$ and $V_P$, where $V_P$ is the value function for $M_P$. If $V(s) - V_P(s) > 2(1 + \frac{\gamma}{1 - \gamma} \max \{ \epsilon, \delta \})$ for $s \in B_i$ then the primitive action that achieves the highest value on the original state in the MDP will be added to the action space of those
states that are in block $B_i$, i.e.:

$$A_{B_i} \leftarrow A_{B_i} \cup \{a | \max_a V(s)\} \quad \forall s \in S$$

and block $B_i$ is refined according to inequalities 8 and 9 until it is stable for the new action set. Once no such significant difference exists, the goal will be achievable in the resulting state space according to Theorem 4. This procedure is illustrated in Figure 6.

VI. Empirical Results

This section describes two simulations in order to evaluate the analytical result of the presented work. Analytical results are confirmed with simulation results in stochastic and deterministic domains.

A. Experiment in a deterministic domain

To illustrate the result of two phase partitioning using subgoal discovery, a number of experiments have been performed with randomly changing goal locations on a chosen environment. This environment consists of three grid worlds, each of which worlds consists of different randomly chosen rooms. The termination states of the available actions are illustrated in black in Figure 7. The actions for each state are multi-step actions which terminate when they reach a subgoal in the same room. The underlying deterministic actions are North, West, South and East and the reward of goal state is +100.

In order to extract the subgoals in this environment, Q-learning has been used to learn a policy for a defined goal in order to compute the counts of predecessors for every state using Equation 5. The mean of the distribution of gradient ratios over the space is 5.427, the standard deviation is 18.09, using t-test and probability 0.025 the computed threshold is 35. Finally this state space has been partitioned according to two phase method with $\epsilon = 10$ and $\delta = 0.3$.

The number of extracted subgoals is less than the total number of hand designed subgoals, so the number of options learned with extracted subgoals are smaller than the number of options learned using predefined subgoals.

In order to show the efficiency of the subgoal discovery methods, the performance of the subgoal discovery method is compared with these hand designed subgoals.

Figure 9(b) shows the difference between the average of 50 different policies computed on the original state space and the SMMDPs constructed based on the hand designed subgoals and the subgoals extracted with the subgoal discovery method. The results show that the difference in values of policies is small and thus executing the tasks on the constructed SMMDPs is almost optimal, however the average Q-value converges in 300 iterations in the constructed SMMDPs while the original state space Q-value converge only after 500 iterations. As illustrated in Figure 9(a) this difference is more significant in the partition space where partitions are build with hand design subgoals and the subgoal discovery method.
Since the average number of blocks in partitions is 63 which is substantially smaller than the number of states (2900) in the original space thus the Q-values in partition spaces converge after 30 iterations as illustrated in Figure 9(a).

B. Experiment in a stochastic domain

The main goal of this experiment is to show the potential of hierarchical learning with autonomous subgoal discovery using Monte Carlo sampling and action-dependent partitioning, in accelerating learning in a stochastic environment. The task of the agent is to find an object in a randomly chosen cell and to pick it up and drop it in another randomly chosen state. Figure 8 illustrates the environment for this experiment. The state space consists of three grid worlds that are connected through stairways (arrows). The dark cells represent obstacles and the actions are GoNorth, GoEast, GoSouth, GoWest, GoUp, GoDown, OpenArm, CloseArm, Pickup and Drop. The available actions for stairways are macro actions defined by sequences of GoUp or GoDown where the size of each sequence is 10. The cost for each single step action is $-1$ and each action for navigation succeeds with probability 0.5 and with probability 0.25 causes the agent to move to the sides and actions OpenArm, CloseArm, Pickup and Drop always succeed with probability 1. The reward in the goal state is 100. In this experiment, the agent first learns a policy to move from a fixed starting location to a particular goal point. It then uses this policy to extract subgoals by generating random samples according to the policy. The samples are paths of length 40 and the subgoals are discovered as described in the autonomous subgoal discovery section. By calculating the count metric exhaustively, the mean of the distribution of gradient ratios over the space will be 8.839 with standard deviation 20.12, using t-test and probability 0.025 the threshold $\mu$ is chosen to be 30.

In order to accelerate the subgoal discovery task, let $\epsilon_N = 2$ and $\max_t C^*_H(s_t) = 40$, since the length of each sample trajectory is 40.

The total number of samples that is required to ensure $|C^*_H(s_t) - C^*_H(s_t)| \leq 2$ with probability 0.99 is

$$\begin{align*}
N &\geq \frac{40}{2(1 + 2)\log\frac{2}{1 - 0.99}} = 138.06.
\end{align*}$$

After collecting 140 samples, we compute the count metric and gradient ratio according to collected samples and we choose the threshold $\mu \geq 30 + \frac{\sqrt{2}}{2} = 31.9$, in order to select the same subgoals that are discovered by calculating the count metric exhaustively, with probability 0.99. Figure 10(b) illustrates the number of subgoals that are discovered by Monte Carlo sampling. As illustrated in Figure 10(b), the total number of samples that are needed to learn almost all of the subgoals is 140. Figure 10(a) shows that the total time spend on subgoal discovery for these 140 samples is less than 30 seconds, which is 5 times faster than using the entire state space for extracting the subgoals. The extracted subgoals in this experiment consist of a set of doorways, opening and closing points of stairways and holding the object by the agent.

Figure 10(a) illustrates the difference between the running time for subgoal discovery using Monte Carlo sampling and using the entire state space.

Figure 11 shows the quality of the learned policy and the
acceleration of learning with and without refinement of abstract states based on Q-value inconsistencies. This experiment shows that when the goal state is not reachable with a subgoal option, learning is impossible. However, according to Theorems 2 and 3, further refinement of the blocks that have inconsistencies in their flat and partition space value functions, ensures the achieveability of the goal state. The comparison of number of block resulted by solving the inconsistencies is illustrated in Figure 12.

VII. CONCLUSION

This paper presents an efficient method for constructing a hierarchical state and action space for SMDPs. To do this, it first discovers subgoals by analyzing previously learned policies for states with particular structural properties. Once subgoals are derived, it learns options to achieve these subgoals off-line and includes these into the action space available to the agent. Using the subgoal options, it then uses action-dependent state space partitioning to derive an abstract state space. Learning of subsequent tasks is addressed on the dependent state space partitioning to derive an abstract state based on Q-value inconsistencies. This experiment is illustrated in Figure 12.

Theorem 2 In a regular space, for sample size

$$N \geq \frac{\max_i C_i^*(s_t)}{\epsilon_N} 2(1 + \epsilon_N) \log\left(\frac{2}{1 - p}\right) \quad (11)$$

it is true that

$$|C_i^*(s_t) - C_n^*(s_t)| \leq \epsilon_N$$

with probability $p$.

Proof: Let $\xi_i = C_i^*(s_t)$ then $E\xi_i = E[C_i^*(s_t)]$. $Z_N = E\xi_1 + \ldots + E\xi_N = E[C_{h_1}^*(s_t)] + \ldots + E[C_{h_N}^*(s_t)] = E[C_{h_1}^*(s_t) + \ldots + C_{h_N}^*(s_t)]$, since $\xi_i$ are independent.

Thus $\frac{1}{N} Z_N = \frac{1}{N} E[C_{h_1}^*(s_t) + \ldots + C_{h_N}^*(s_t)] = E[C_H^*(s_t)] = C_n^*(s_t)$.

Let $a = \max_i C_i^*(s_t)$ then $|\xi_i| \leq a, a > 0$. Also for $M_N = E\xi_1 + \ldots + E\xi_N$ we have $Var(M_N) = E\xi_1^2 + \ldots + E\xi_N^2 \leq N(max_i C_i^*(s_t))^2$. 

APPENDIX

Theorem 1 (Bernstein [33]) Let $\xi_1, \xi_2, \ldots$ be independent random variables with means $E\xi_i$, bounded by some constant $E\xi_i \leq \alpha, \alpha > 0$. Also let $Var(M_N) = E\xi_1^2 + \ldots + E\xi_N^2 \leq L$ and $Z_N = E\xi_1 + \ldots + E\xi_N$. Then the partial sum $M_N = \xi_1 + \ldots + \xi_N$ obey the following inequality for all $N \geq 0$:

$$Pr \left( \frac{1}{N} M_N - \frac{1}{N} Z_N > \epsilon_N \right) \leq 2 \exp \left( \frac{1}{2} \epsilon_N^2 N \right)$$
Let $L = N(\max C_H(s_t))^2$ and according to Bernstein’s inequality:

$$Pr \left( |C^*_H(s_t) - C^*_\pi(s_t)| > \epsilon_N \right) \leq 2\exp \left( -\frac{1}{2} \max_t C_H(s_t)^2 + \max C_H(s_t)\epsilon_N \right)$$

then

$$Pr \left( |C^*_H(s_t) - C^*_\pi(s_t)| > \epsilon_N \right) \leq 2\exp \left( -\frac{1}{2} \max C_H(s_t)\epsilon_N \right)$$

by letting

$$1-p = Pr \left( |C^*_H(s_t) - C^*_\pi(s_t)| > \epsilon_N \right)$$

we have

$$1-p \leq 2\exp \left( -\frac{1}{2} \max C_H(s_t)\epsilon_N \right)$$

After solving for $N$, we get the statement of theorem 2. □

**Theorem 3** Let $H = \{h_1, ..., h_N\}$ be $N$ sample trajectories induced by policy $\pi$ with $N$ selected according to Equation 7. If $\frac{\Delta_H(s_t)}{\max \{1, \Delta_H(s_{t+1})\}} > \mu + \frac{2\epsilon}{\max \{1, \Delta_H(s_{t+1})\}}$, then $\frac{\Delta_\pi(s_t)}{\max \{1, \Delta_\pi(s_{t+1})\}} \geq \mu$ with probability $\geq p$.

**Proof:** Let $N \geq \frac{\max C_H(s_t)}{\epsilon_N}2(1 + \epsilon_N)\log \left( \frac{1}{\epsilon_N} \right)$ then $|C^*_H(s_t) - C^*_\pi(s_t)| \leq \epsilon_N$ and Similarly $|C^*_H(s_{t+1}) - C^*_\pi(s_{t+1})| \leq 2\epsilon_N$, thus

$$\frac{\Delta_\pi(s_t) - \Delta_H(s_t)}{\max \{1, \Delta_\pi(s_{t+1})\}} \leq 2\epsilon_N$$

and

$$2\epsilon_N \leq \Delta_\pi(s_t) - \Delta_H(s_t) \leq 2\epsilon_N$$

and

$$\Delta_\pi(s_t) - 2\epsilon_N \leq \Delta_H(s_t) \leq \Delta_\pi(s_t) + 2\epsilon_N$$

Similarly we can show that

$$\Delta_\pi(s_{t+1}) - 2\epsilon_N \leq \Delta_H(s_{t+1}) \leq \Delta_\pi(s_{t+1}) + 2\epsilon_N$$

by dividing inequality Equation 12 by $\max \{1, \Delta_H(s_{t+1})\}$ we have:

$$\frac{\Delta_H(s_t)}{\max \{1, \Delta_H(s_{t+1})\}} \leq \frac{\Delta_\pi(s_t) + 2\epsilon_N}{\max \{1, \Delta_\pi(s_{t+1})\}}$$

if $\frac{\Delta_\pi(s_t)}{\max \{1, \Delta_\pi(s_{t+1})\}} \geq \mu + \frac{2\epsilon}{\max \{1, \Delta_\pi(s_{t+1})\}}$, then according to Equation 13 $\frac{\Delta_\pi(s_{t+1}) + 2\epsilon_N}{\max \{1, \Delta_H(s_{t+1})\}} \geq \mu + \frac{2\epsilon_N}{\max \{1, \Delta_H(s_{t+1})\}}$ and as a result $\frac{\Delta_\pi(s_{t+1})}{\max \{1, \Delta_\pi(s_{t+1})\}} \geq \mu$.

Theorem 3 implies that for a sufficiently large sample size the exhaustive and the sampling method predict the same subgoals with high probability. □

**Theorem 4** For any policy $\pi$ for which the goal $G$ can be represented as a conjunction of terminal sets (subgoals) of the available actions in the original MDP $M$, there is a policy $\pi_P$ in the reduced MDP $M_P$, that achieves $G$ as long as for each state $s_t$ in $M$ for which there exists a path to $G$, there exists a path such that $F(G|s_t, \pi_P(s_t)) > \delta$.

**Proof:** The blocks of partition $\Phi = \{B_1, ..., B_n\}$ have the following property

$$\sum_{s \in B_j} F(s|s_1, o_i(s_1)) - \sum_{s \in B_j} F(s|s_2, o_i(s_2)) \leq \delta$$
For every policy $\pi$ that fulfills the requirements of the proposition, there exists a policy $\pi_{\Phi}$ in partition space such that for each $n \in \mathbb{N}$, if there is a path of length $n$ from state $s_0$ to a goal state $G$, under policy $\pi$, then there is a path for block $B_{s_0}$ containing $s_0$ to block $B_G$ containing $G$, under policy $\pi_{\Phi}$.

**Case** $k = 1$: if $F(G(s_0, \pi(s_0))) > \delta$ then by inequality 14 for all $s \in B_{s_0}$, we have

$$\left| \sum_{s' \in B_G} F(s'|s_0, \pi(s_0)) - \sum_{s' \in B_G} F(s'|s, \pi(s_0)) \right| \leq \delta$$

thus $\forall s \in B_{s_0}$ we have

$$F(G(s, \pi(s_0))) > F(G(s_0, \pi(s_0))) - \delta > 0.$$ 

policy $\pi_\Phi$ such that $\pi_\Phi(B_{s_0}) = \pi(s_0)$, then $F(B_G|B_{s_0}, \pi_\Phi(B_{s_0})) > 0$.

**Case** $k = n - 1$: Assume for each path of length less than or equal to $n - 1$ that reaches state $G$ from $s_0$ under policy $\pi$, there is a path under policy $\pi_{\Phi}$ in the partition space. 

**Case** $k = n$: Each path that reaches with $G$ from $s_0$ under policy $\pi$ in $n$ steps contains a path with $n - 1$ steps, that reaches $G$ from $s_1$ under policy $\pi$. By induction hypothesis, there is a policy $\pi_{\Phi}$ that leads to $B_G$ from $B_{s_1}$. Now if $s_0$ is an element of $B_{s_0} \cup B_{s_{n-1}} \cup \ldots \cup B_{s_1}$, the blocks already chosen by paths with length less than or equal $n - 1$, then there is a policy $\pi_{\Phi}$ that leads to $B_G$ from $B_{s_0}$ under policy $\pi_{\Phi}$ and the policy $\pi_{\Phi}(B_{s_0})$ is already defined. But if $s_0 \notin B_{s_0} \cup B_{s_{n-1}} \cup \ldots \cup B_{s_1}$ then by induction hypothesis it has only to be shown that there is a policy $\pi_{\Phi}$ that fulfills the induction hypothesis and which leads from $B_{s_0}$ to $B_{s_1}$ such that $F(B_{s_1}|B_{s_0}, \pi(s_0)) > 0$. By inequality 14 $\forall s_1, s_2 \in B_{s_0}$ we have

$$\left| \sum_{s' \in B_G} F(s'|s_0, \pi(s_0)) - \sum_{s' \in B_G} F(s'|s, \pi(s_0)) \right| \leq \delta$$

thus

$$F(B_{s_1}|B_{s_0}, \pi_{\Phi}(B_{s_0})) = \sum_{s' \in B_{s_1}} F(s'|s_0, \pi(s_0)) \geq \sum_{s_0 \in B_{s_1}} F(s'|s_0, \pi(s_0)) - \delta > 0. \square$$

**REFERENCES**


