We introduce a computationally tractable surrogate objective that is
with the problem of an agent learning from interactions with its
(a domain). An agent learning in this setting should be able exploit
Under Review (AAMAS 2018), 2018
corresponds to a sequence of decisions or actions:
τ
s
state
expected sum of rewards. A
algorithm is to learn an optimal policy
environment. In this framework, an agent is at a state
τ
s
policy
In many practical RL problems, however, such prior information
tasks, when seen through an appropriate state abstraction (i.e., given
appropriate state features). For instance, let us say that tasks cor-
rules when analyzed over an appropriate state abstraction (i.e., given
information (about related problems) when deployed on a new task.
In many practical RL problems, however, such prior information
correspond to mazes with different configurations. We assume there
exist a (possibly unknown) state abstraction ϕ(s) that maps a state to,
e.g., its (x, y) coordinates on the maze. The states that may result
from executing action Right, when in a state with abstraction ϕ(s),
may vary from maze to maze, since that particular location may
or may not be next to a wall. However, those possible next states
belong to a same set that is shared by all tasks: either the agent
successfully moves to the right, thereby transitioning to a state with
abstraction (x + 1, y); or it hits a wall, thereby transitioning to a state
with abstraction (x, y). This implies that even though mazes may
differ, they do share common properties in terms of the possible
outcomes of executing an action in a given state: the dynamics of
the tasks, when seen through an appropriate state abstract ϕ, are not
arbitrarily different, since tasks are assumed to be related. More
formally, let us assume that there exists a (possibly unknown) transition
function f(a, ϕ(s)) → ϕ(s′) determining the (abstract representation)
of the state s′ that results from executing a in state s. Since tasks in
a domain are assumed to be related, the states that may result of
executing an action in a state are not arbitrarily different and form a
set Φ(a, ϕ(s)) which is common to all tasks. This implies that tasks in
a domain do share similar underlying dynamics—in the case of
sample mazes, for instance, the possible transitions resulting from
collisions against a wall are common to all tasks. State abstraction
similar to ϕ, which allow for different but related MDPs to be ana-
lyzed and compared, have been studied by others and shown to allow

We introduce a computationally tractable surrogate objective that is
with the problem of an agent learning from interactions with its
(a domain). An agent learning in this setting should be able exploit
Under Review (AAMAS 2018), 2018
corresponds to a sequence of decisions or actions:
τ
s
state
expected sum of rewards. A
algorithm is to learn an optimal policy
environment. In this framework, an agent is at a state
τ
s
policy
In many practical RL problems, however, such prior information
tasks, when seen through an appropriate state abstraction (i.e., given
appropriate state features). For instance, let us say that tasks cor-
rules when analyzed over an appropriate state abstraction (i.e., given
information (about related problems) when deployed on a new task.
In many practical RL problems, however, such prior information
correspond to mazes with different configurations. We assume there
exist a (possibly unknown) state abstraction ϕ(s) that maps a state to,
e.g., its (x, y) coordinates on the maze. The states that may result
from executing action Right, when in a state with abstraction ϕ(s),
may vary from maze to maze, since that particular location may
or may not be next to a wall. However, those possible next states
belong to a same set that is shared by all tasks: either the agent
successfully moves to the right, thereby transitioning to a state with
abstraction (x + 1, y); or it hits a wall, thereby transitioning to a state
with abstraction (x, y). This implies that even though mazes may
differ, they do share common properties in terms of the possible
outcomes of executing an action in a given state: the dynamics of
the tasks, when seen through an appropriate state abstract ϕ, are not
arbitrarily different, since tasks are assumed to be related. More
formally, let us assume that there exists a (possibly unknown) transition
function f(a, ϕ(s)) → ϕ(s′) determining the (abstract representation)
of the state s′ that results from executing a in state s. Since tasks in
a domain are assumed to be related, the states that may result of
executing an action in a state are not arbitrarily different and form a
set Φ(a, ϕ(s)) which is common to all tasks. This implies that tasks in
a domain do share similar underlying dynamics—in the case of
sample mazes, for instance, the possible transitions resulting from
collisions against a wall are common to all tasks. State abstraction
similar to ϕ, which allow for different but related MDPs to be ana-
lyzed and compared, have been studied by others and shown to allow

Identifying Reusable Macros for Efficient Exploration via Policy Compression

Francisco M. Garcia
University of Massachusetts - Amherst
Amherst, Massachusetts, USA
fmgarcia@cs.umass.edu

Bruno C. da Silva
Universidade Federal do Rio Grande do Sul
Porto Alegre, Rio Grande do Sul, Brazil
bruno.silva@inf.ufrgs.br

ABSTRACT
Reinforcement Learning agents often need to solve not a single task,
but several tasks pertaining to a same domain; in particular, each
task corresponds to an MDP drawn from a family of related MDPs
(a domain). An agent learning in this setting should be able exploit
policies it has learned in the past, for a given set of sample tasks, in
order to more rapidly acquire policies for novel tasks. Consider, for
instance, a navigation problem where an agent may have to learn to
navigate different (but related) mazes. Even though these correspond
to distinct tasks (since the goal and starting locations of the agent
may change, as well as the maze configuration itself), their solutions
do share common properties—e.g. in order to reach distant areas of
the maze, an agent should not move in circles. After an agent has
learned to solve a few sample tasks, it may be possible to leverage
the acquired experience to facilitate solving novel tasks from the
same domain. Our work is motivated by the observation that trajectory
samples from optimal policies for tasks belonging to a common
domain, often reveal underlying useful patterns for solving novel
tasks. We propose an optimization algorithm that characterizes the
problem of learning reusable temporally extended actions (macros).
We introduce a computationally tractable surrogate objective that is
equivalent to finding macros that allow for maximal compression of
a given set of sampled trajectories. We develop a compression-based
approach for obtaining such macros and propose an exploration
strategy that takes advantage of them. We show that meaningful
behavioral patterns can be identified from sample policies over discrete
and continuous action spaces, and present evidence that the proposed
exploration strategy improves learning time on novel tasks.

KEYWORDS
Reinforcement Learning; Autonomous Agents; Exploration

1 INTRODUCTION
Reinforcement Learning (RL) is an active area of research concerned
with the problem of an agent learning from interactions with its
environment. In this framework, an agent is at a state s_t at time step
t, takes an action a_t, receives reward r_t, and moves to state s_{t+1}. A
policy π defines a mapping from states to actions and determines
the behavior of an agent in its environment. The objective of an RL
algorithm is to learn an optimal policy π∗ that achieves the maximum
expected sum of rewards. A sample from a policy, from some initial
state s_0 to some terminal state s_T, is referred to as a trajectory, τ, and
corresponds to a sequence of decisions or actions:
τ = \{a_1, \ldots, a_T\}.
for option policies to be defined over abstract state representations—
thereby making it possible for such options to be used to reused and
deployed when tackling different related tasks [8].

In this work we propose to identify reusable options by exploiting
the observation that the possible consequences of executing an action
in a particular state are shared by many related tasks. This suggests
the existence of a (possibly unknown) common transition dynam-
ics structure underlying all tasks. We propose to identify options
reflecting this common structure—in particular, options that encode
recurring temporally extended actions (also called macros, [20], [17],
[1]) that occur as parts of the solution to many related tasks in the
domain. We also propose an exploration strategy making use of
such recurring options/action patterns in order to facilitate learning of
novel tasks. Whereas primitive actions last for one time-step, taking
the agent from state $s_t$ to $s_{t+1}$, a temporally extended action lasts for
$n > 1$ time-steps, taking the agent from $s_t$ to $s_{t+n}$.

In order to identify these macros, we propose to solve a surrogate
problem: to identify macro actions that allow for a given set of
sample trajectories to be compactly reconstructed. In particular, if
we represent different action sequences of length $l$ (which may occur
as part of a trajectory $\tau$) by a unique symbol, we can borrow ideas
from the compression literature and find binary representations for
the possible corresponding macros. Given binary representations of
this type, we can then evaluate the expected number of bits required
to encode a set of trajectories (drawn from optimal policies to a set
of sample tasks) in a compact way. By construction, solving this
surrogate problem formulation implies that the identified macros will
be recurring behavioral patterns in the policies for different tasks;
they do, therefore, reflect common structures and action patterns in
the solutions to different (but related) problem instances drawn from
a same domain.

In this work we focus on identifying a set of open-loop temporally
extended actions which are used for exploration (when tackling novel
tasks) in order to bias learning efforts towards more promising action
sequences. Intuitively, such macros allow an agent to more quickly
reach parts of the state space that would be unlikely to be reached
by random or unguided exploration, and that are often encountered
when solving tasks drawn from the domain.

In this work, we make the following contributions:

- propose a compression-based method for obtaining reusable
  macros from a set of sample trajectories;
- demonstrate how two different compression algorithms (Huff-
  man coding and LZW) can be used in this framework;
- introduce an exploration strategy that leverages the empirical
distribution of obtained macros in both discrete and continu-
ouos action spaces;
- provide experiments that show the benefits of our compression-
based approach for learning exploration strategies in standard
RL benchmarks.

2 RELATED WORK

The proposed work lies at the intersection between transfer learning
(TL), option discovery, and learned exploration strategies. We show
how to discover useful recurring macro-actions from previous ex-
periences, and leverage this knowledge to derive useful exploration
strategies for solving new problems.

The literature on these related research areas is rich. [22] pro-
vides a detailed survey on existing TL methods and the different
metrics used to evaluate how much it is gained from transferring
knowledge. From this survey two main metrics are relevant to our
work: jumpstart (the improvement of initial performance of an agent
when solving a novel task (over an agent without this knowledge),
and total reward (the total cumulative reward received by an agent
during training).

Our approach requires the action-set to be shared throughout tasks,
but no explicit state variable mapping between tasks is assumed or
given. In a similar manner, [9] proposed separating the problems
being solved by an agent into an agent-space and a problem-space,
such that options defined over the agent-space are transferable across
different tasks. In their experiments, options or macros shared similari-
ties in that they had access to a same set of sensors; the authors
did not exploit the fact that they were also being deployed over
tasks with a shared set of abstract transition rules. Similarly, [7]
developed a framework for obtaining an approximation to the value
function $V$ when defined in the problem-space, resulting in a method
to jumpstart learning. More recent work has focused on using neural
networks to learn problem-invariant feature representations to trans-
fer knowledge between tasks instances [5], [16]. These approaches,
however, do not directly address the problem of learning transfer-
able exploration strategies, and instead rely on standard exploration
techniques such as $\epsilon$-greedy or Boltzmann.

Option extraction techniques have also been studied at length. One
such approach to learn options is based on proto-value functions
[13], [12]. Here, an agent builds a graph of the environment and uses
such a model to identify temporally extended behaviors that
allow promising areas of the state space to be reached. Another
recent approach to option discovery relies on learning a model of
the environment and using the graph Laplacian to cluster states into
abstract states [10]. These methods depend on fully exploring the
environment a priori in order to build an accurate model, which
implicitly assumes that an efficient exploration strategy is available.

The problem of learning efficient exploration strategies has not
been overlooked. Most existing approaches seek to find efficient
ways to use the knowledge gained about a given task that the agent
is currently solving, as opposed to leveraging existing knowledge
of policies learned in previously-experienced related tasks. [21]
proposed a count-based exploration strategy by hashing frequently-
encountered states and adding a corresponding exploration bonus
to the reward function. In this work, high dimensional states are
mapped to a lower dimensional representation, thus allowing esti-
mates of the reward function to be transferable across tasks. Another
recently proposed work in learning to explore is that of [4], where
a generative adversarial network (GAN) is trained to estimate state
visitation frequencies and use them as a reward bonus. In both cases,
the resulting change in agent behavior is implicitly determined by
a new reward function, and not part of an explicitly-derived explo-
ration strategy. Furthermore, most of these methods have in common
the reuse of knowledge of the value function; they do not directly
analyze the behavior (policies) of the agent under optimal policies
for previously-solved tasks.

In this work, we aim at addressing these limitations. We analyze
sample trajectories drawn from optimal policies to related tasks and
use them to derive an exploration strategy that is agnostic to the
state space representation and which is transferable and applicable to novel tasks drawn from a same domain. The main requirement of our method is that the action space remains constant across tasks.

3 PROBLEM FORMULATION

We are interested in obtaining macro-actions that would allow an agent to more rapidly acquire an optimal policy \( \pi^* \) for a given task \( v \) drawn from a domain \( \Upsilon \). A particular task instance drawn from the problem domain is an MDP. Let \( A = \{a_1, a_2, \ldots, a_k\} \) be an action set containing all primitive actions available in all tasks in \( \Upsilon \). We seek to find a set of macros \( M \) such that if a new action set \( A' = A \cup M \) were to be used, it would allow for a set of sample trajectories (drawn from optimal policies for sample tasks in the domain) to be expressed in a more compact manner. Given one candidate binary representation for each primitive action and macro in \( M \), we define \( B_A(\tau, v) \) as the minimum number of bits needed to represent a given trajectory \( \tau \), sampled from the optimal policy for some task \( v \), by using action set \( A \). Let \( E_{\Upsilon-1}[B_A(\tau_{\pi^*}, v)] \) be the expected number of bits needed to represent a trajectory \( \tau_{\pi^*} \) (sampled from an optimal policy \( \pi^* \) to task \( v \)) given an action set \( A \) and domain \( \Upsilon \). We wish to find a new action set \( A' = A \cup M \), such that:

\[
E_{\Upsilon-1}[B_{A'}(\tau_{\pi^*}, v)] \leq E_{\Upsilon-1}[B_A(\tau_{\pi^*}, v)].
\]

The reasoning underlying this objective is that finding a set of macros that leads to compressed trajectory representations implies that these macro-actions are encountered frequently as part of optimal policies in the domain. The action patterns emerging from such trajectories allow us to determine which action sequences do occur often in optimal policies, and which ones do not. Identifying macros that maximize trajectory representation compression, therefore, allows us to capture the underlying action patterns in optimal policies \( \pi^* \) from which trajectories were sampled; these can then be used by the agent to bias exploration when tackling novel tasks from the same domain. To achieve the goal above, we propose to minimize the following objective function:

\[
J(A') = E_{\Upsilon-1}[B_{A'}(\tau_{\pi^*}, v)] + f_e(A').
\]

The first term in Equation (1) seeks to minimize the expected number of bits needed to encode trajectories sampled from their corresponding policies \( \pi^* \), while the second term is a regularizer on the dimensionality of the extended action space; this dimensionality depends on the particular encoder \( e \) used to compressed trajectories by re-writing them using a particular extended action set. On one hand, if the extended action set \( A' \) were to become too large, it could include useful macros but could make learning more challenging, since there would be too many actions whose utilities need to be estimated. On the other hand, if \( A' \) is too small, the agent would forfeit the opportunity of deploying macros that encode useful recurring behaviors.

In practice, it may be infeasible or too expensive to find the set \( A' \) that minimizes this expression, since the agent can only sample tasks from \( \Upsilon \). We can, however, approximate the solution to \( J(A') \) by sampling tasks and optimizing the following surrogate objective:

\[
\hat{J}(A') = \frac{1}{|T|} \sum_{\tau \in T} (B_{A'}(\tau_{\pi^*}, v)) + f_e(A').
\]

where \( T \) is a set of sampled tasks drawn from the domain. Next, we show how to use this formulation to obtain useful macro-actions from sample trajectories by using two different compression algorithms: Huffman Coding and LZW.

3.1 Compression via Huffman Coding

Huffman coding [6] is a compression algorithm that assigns variable-length codes to each symbol in its codebook. This technique requires every possible symbol and their respective probabilities to be known in advance, which is commonly done as a pre-processing step. Assuming that all macros have a fixed length \( l \), we first identify all action sequences of length \( l \) that occur in the set of sampled trajectories and compute their respective probabilities (i.e., the frequency with which they occur in the sampled trajectories). For example, given a trajectory \( \tau = \{a_1, a_2, \ldots, a_k\} \) and maximum macro length \( l = 2 \) we identify candidate macros \( m_1 = \{a_1, a_2\} \), \( m_2 = \{a_2, a_3\} \), \( m_3 = \{a_1, a_2\} \); their respective probabilities are \( p_1 = 0.5, p_2 = 0.25, p_3 = 0.25 \). Equipped with this data, we use Huffman coding as follows. Let \( M = \{m_1, m_2, \ldots, m_n\} \) be the set of available symbols corresponding to macros and \( P = \{p_1, p_2, \ldots, p_n\} \) be a set of probabilities, where \( p_i \) is the probability of macro \( m_i \) appearing in sampled trajectories for tasks taken from \( \Upsilon \). Based on \( M \) and \( P \), we create a codebook \( C(M, P) \) that generates binary encodings \( c_i \) for each macro \( m_i \); that is, a string of 0s and 1s that uniquely identifies a macro \( m_i \); Huffman Coding ensures that the more frequent a macro is in the sampled trajectories, the shorter its encoding will be. In order to keep the extended action set from including all possible length-1 macros, we may wish to keep only the \( n \) most frequently occurring ones—i.e., the \( n \) macros with shorter binary encodings. For added flexibility, we also extend the codebook to include the primitive action set \( A \) and ensure (by construction) that their corresponding codes are longer than those used to represent macros. This is done to represent our preference for using macros over primitives. A given trajectory \( \tau_{\pi} \) (drawn from a policy \( \pi \) for task \( v \)) can then be re-expressed as some sequence of primitives and macros \( \tau_{\pi} = \{c_i^1, c_i^2, \ldots, c_i^C\} \), where \( c_i^j \) is the codeword of the action or macro performed at time-step \( i \). This minimum-length representation of a given trajectory has length \( B_{A'}(\tau_{\pi}, v) \) and can be identified via a simple dynamic programming algorithm, which we omit here due to space constraints.

Note that when using this approach, we may wish to regularize \( A' \) so as not to include highly unlikely macros—i.e., macros with large code lengths. Let \( c_{\text{max}} \) be the length of the longest code in \( M \) and let \( f_e(A') = \lambda c_{\text{max}} \), where \( \lambda \) is a regularization parameter; our objective is now given by:

\[
\hat{J}(A') = \frac{1}{|T|} \sum_{\tau \in T} (B_{A'}(\tau_{\pi^*}, v)) + \lambda c_{\text{max}}.
\]

Assuming an upper bound on \( n \) (the number of macros we wish to obtain) and on \( l \) (the length of those macros), we can identify the particular values of \( n \) and \( l \) that minimize \( \hat{J}(A') \) by executing an iterative search for values of \( n = \{1, \ldots, n_{\text{max}}\} \) and \( l = \{l_{\text{min}}, l_{\text{min}}+1, \ldots, l_{\text{max}}\} \). This allows for recurring macros (and the probability distribution with which they occur as part of optimal solutions to related tasks) to be identified. However, a pre-processing step, which builds a candidate codebook for each possible length \( l \)
and estimates the corresponding macros’ probabilities, is required and can be computationally expensive:

**Theorem 3.1.** Let \( l_{\text{min}} \) and \( l_{\text{max}} \) be the minimum and maximum allowed length for a macro, \( |T| \) the number of sampled trajectories, and \( S = \max_k (|r^k| - l_{\text{min}}) \), where \( |r^k| \) is the length of the \( k^{th} \) sampled trajectory. Let \( l_D = l_{\text{max}} - l_{\text{min}}. \) The pre-processing step has complexity \( O(l_D |T| S). \)

**Proof.** Proof given in appendix A.

Furthermore, we can show an upper bound on the number of bits required to represent sample trajectories if using Huffman Coding—which is only one possible compression scheme, but not necessarily the one that achieves the true maximum compression according to the proposed objective:

**Theorem 3.2.** Assume a set of \( m \) macros, \( m > 1, \) and corresponding probabilities \( \{p_1, \ldots, p_m\}, \) where \( p_i \leq p_{i+1} \) for \( i = 1, \ldots, m - 1. \) If a codebook constructed via Huffman Coding is used, the number of bits \( B_k \) needed to represent a set of \( |T| \) sampled trajectories is upper bounded by:

\[
c \times \min \left( \log \frac{\rho + 1}{\rho p_1 + p_2}, \log \frac{m - 1}{m} \right)
\]

where \( c = \left\lceil \frac{|T|}{\rho} \right\rceil \) and \( \rho = \frac{1 + \sqrt{5}}{2}. \)

**Proof.** The proof follows trivially from [3].

Huffman Coding constructs a codebook by analyzing (during an offline pre-processing step) all candidate macros of length \( l \) that occur in the sampled data. This results in the construction of binary representations that take into account the frequencies of each macro in the entire data set, but which is (as discussed above) computationally expensive. To address this problem we show, in the next section, how to use an on-line compression: LZW.

### 3.2 Compression via LZW

LZW [24] is another compression technique that can be used to identify binary encodings for each macro and primitive action in order to compress trajectories. However, unlike Huffman coding, it does not require pre-processing the data to build codes for each symbol. LZW assumes that the same number of bits will be used to represent all symbols in its codebook and incrementally builds it as it processes a given message, string, or set of trajectories. For example, in the case of encoding the English language, it is standard practice to set the limit in the number of bits per symbol to 8; LZW will then be able to represent/store 256 symbols in its codebook. The first symbols to be included in the codebook would be, in this case, characters \( a \) through \( z. \) The remaining of the codebook is then built incrementally, as new character combinations are found in the message, string, or trajectory. For instance, if the symbol \( a \) is already in the codebook and the subsequence \( ae \) is found in the input, the symbol \( ae \) is added to the codebook. In this sense, LZW is a greedy method: it always tries to create new, longer symbols that match the patterns observed in the data. This process continues until no more symbols can be represented by the allotted number of bits. If we consider primitive actions in trajectories analogous to single characters in an alphabet, we can easily extend this compression method to identify a codebook containing macros. We first define a limit, \( b_{\text{limit}} \), in the number of bits we allow the codebook to have, and initialize the codebook with primitive actions. As we compress a set of sampled trajectories with LZW, macro actions that recur will naturally be identified and be added to the codebook. As it is the case with Huffman coding, we are also interested in computing \( P \), the set of probabilities (frequency of occurrence) for a given set of macros \( M \); this will be used later when using the identified macros to construct an exploration strategy. \( P \) can be estimated directly by counting (in the sampled trajectories) the number of matches that each macro in the codebook has and normalizing those values into a valid probability distribution.

As before, we may wish to regularize \( A' \) so as to keep it from becoming too large; this can be achieved by penalizing the objective function based on \( b_{\text{limit}} \). Let the regularization term be given by \( f_{\text{el}}(A') = \lambda b_{\text{limit}}, \) where \( \lambda \) is the regularization parameter; our objective now becomes:

\[
f(A') = \frac{1}{|T|} \sum_{\tau \in T} (B_A(\tau, v)) + \lambda b_{\text{limit}}. \tag{4}
\]

Since LZW does not require an expensive pre-processing step, it is possible to efficiently iterate over different values \( b_{\text{limit}} \) in order to find the value of \( b_{\text{limit}} \) that minimizes this above objective. The particular number of bits needed to encode a given set of \( |T| \) sampled trajectories can be upper bounded as by the following theorem:

**Theorem 3.3.** Given \(|A|\) primitive actions and the number \( N \) of symbols stored in the codebook constructed by LZW, the total number of bits \( B_k \) needed to represent \( |T| \) sampled trajectories is upper bounded by:

\[
\left( \sum_{k=1}^{|T|} |r^k| - \sum_{j=1}^{i-1} |A|^j \right) \times b_{\text{limit}}
\]

where \( i = \lfloor \log |A| \rfloor - N \frac{|A| - 1}{|T|}. \)

**Proof.** The proof is given in Appendix A.

A downside of using LZW to identify recurring macros is that the macros that incrementally populate the codebook depend on the order in which trajectories are processed. Unlike Huffman coding, therefore, and depending on the maximum size that the codebook can be, it is possible that some frequently-occurring macros will be excluded from the extended action set.

### 4 GUIDING EXPLORATION VIA RECURRING MACROS

In the previous sections we showed how to use different compression methods to approximately solve the proposed minimization objective. The proposed methods, however, implicitly assumed the existence of a process for determining whether two particular sequences of actions (or macros) were equal. This was needed, for instance, for LZW to check if the current set of symbols in the input was a match with any of the existing macros in its codebook. It was also needed to count how many times a given sequence of actions/macro occurred within sampled trajectories for estimating its probability. In this section we discuss how such comparisons can be...
done in both discrete and continuous action spaces, and then propose an exploration strategy that leverages the macros identified by our method.

4.1 Macros for Discrete-Action Task Exploration
In the case of discrete action spaces, equivalence between macros can be easily established. Let $k_1$ and $k_2$ be the length of macros $m_1$ and $m_2$, respectively. $m_1$ and $m_2$ are equivalent iff $k_1 = k_2$ and $m_{1,t} = m_{2,t}$, for $t = 1, \ldots, k_1$, where $m_{1,t}$ and $m_{2,t}$ refer to the actions taken at time-step $t$ in macro $m_1$ and $m_2$, respectively. We can use this equivalence relation along with the methods presented in the previous section whenever a comparison between two macros was needed in order to identify a given set of recurring macros, $M$, and its corresponding probability distribution, $P$.

4.2 Macros for Continuous-Action Task Exploration
Continuous action spaces present unique challenges. Unlike discrete action spaces, their action spaces are infinite (e.g., contain an infinite number of primitive actions); for this reason, it is unlikely that any two sequences of actions will be identical. To deal with this situation, we could discretize such a space, but this would raise a new question: what discretization resolution should be selected? If the resolution is too coarse, the agent might not be able to execute particular actions that are part of an optimal policy. On the other hand, if the resolution is too fine-grained, it becomes unlikely that any two sequences will repeat in subsequent executions of a policy, which implies that identifying recurring action sequences becomes nontrivial. In this setting, we propose to check for the equivalence between two continuous-action trajectories by measuring the distance between them according to the Dynamic Time Warping (DTW) method. DTW [2] is an algorithm developed for measuring the similarity between two continuous signals by measuring the distance between them. We assume that the continuous-action trajectories being compared are Repeating continuous signals that vary with time. It produces a mapping from one signal to the other, as well as a distance estimate entailed by such a mapping. In particular, given two macros $m_1$ and $m_2$ of length $k$, we use DTW to define the following equivalence relation:

$$
\begin{align*}
(m_1 = m_2 &: dtw(m_1, m_2) < \alpha \\
(m_1 \neq m_2 &: \text{otherwise})
\end{align*}
$$

where $dtw(m_1, m_2)$ is the mapping distance between $m_1$ and $m_2$, as given by DTW, and $\alpha$ is an environment-dependent similarity threshold. We assume that the continuous-action trajectories being analyzed are sampled in time according to some fixed frequency $dt$ and are stored in a vector $r$ whose $i$-th element is the continuous action executed at the $i$-th time step. When using DTW to identify recurring continuous macros of length $l$ via Huffman Coding or LZW, we interpret $l$ as the desired time duration of the macros; these macros then correspond to action subsequences formed by $\frac{l}{dt}$ contiguous elements in $r$. This implies that candidate continuous macros can be extracted from $r$ and computationally represented as finite vectors of continuous actions. Whenever two candidate macros are compared and deemed equivalent according to the DTW criterion, they are clustered and the mean of the trajectories in the cluster is used to represent the macro itself. For example, assume there are two clusters of trajectories deemed equivalent, $c_1$ and $c_2$, each of which initially contains only one macro: $c_1 = \{m_1\}$ and $c_2 = \{m_2\}$.

Assume that the representative macro associated with a cluster $c_i$ is the mean between the trajectories in that cluster, denoted by $m_i$. When analyzing whether a new candidate macro $m_3$ is equivalent to existing ones representing clusters, three things can happen:

1. $dtw(m_3, m_1) < dtw(m_3, m_2) < \alpha$: In this case we update $c_1 = \{m_1, m_3\}$, and $m_1 = m_1 + m_3$;
2. $dtw(m_3, m_2) < dtw(m_3, m_1) < \alpha$: In this case we update $c_2 = \{m_2, m_3\}$, and $m_2 = m_2 + m_3$;
3. $dtw(m_3, m_1) > \alpha$ and $dtw(m_3, m_2) > \alpha$: In this case we create a new cluster $c_3 = \{m_3\}$, and $m_3 = m_3$.

4.3 Modified $\epsilon$-greedy Exploration with Macros
We now propose a simple way of using the identified set of macros $M$ (and corresponding probability distributions $P$) to design a modified $\epsilon$-greedy exploration strategy:

$$
\pi_{\exp}(s) = \begin{cases} 
\arg\max_{a \in A'} Q(s, a) & \text{w.p. } (1 - \epsilon) \\
\frac{m}{\pi \sim P} & \text{otherwise}
\end{cases}
$$

That is, with probability $1 - \epsilon$ the agent behaves greedy with respect to the extended action set $A'$ and with probability $\epsilon$ it draws a macro $m$ according to the probability distribution $P$. Selecting actions in this manner biases exploration towards macros that occurred often in similar tasks; these macros are selected according to the estimated probability distribution with which they were part of optimal solutions to tasks sampled from the domain. The use of recurring macros to bias exploration allows the agent to more easily reach states that would be unlikely to be reached by random or unguided exploration, and that are often encountered when solving tasks drawn from the domain.

5 EXPERIMENTS
We carried out two sets of experiments (in discrete and in continuous action spaces) in order to evaluate the benefits of using the macros identified by our method. In the discrete case, we evaluated our method in a navigation task involving maps taken from the video game Dragon’s Age, whereas in the continuous case we used mountain car [15]. Sample trajectories from optimal policies were collected by using Q-Learning [23] and Deterministic Policy Gradient (DPG) [18] to solve different tasks drawn from each corresponding domain. Our proposed macro-based exploration strategy was tested using both Huffman Coding and LZW on novel tasks against an agent with no macros and an agent with a set of macros randomly defined and sampled uniformly during exploration. Note that our method could, if necessary, be coupled with orthogonal exploration strategies that do not make use of knowledge gained when previously solving related tasks, such as techniques that rely on reward bonuses.
5.1 The Dragon’s Age Domain

For this set of experiments, we trained an agent to find optimal policies on 20 different mazes corresponding to challenging environments drawn from the Dragon’s Age video game [19]. These mazes have sizes varying between 400 and approximately 30,000 states. The agent receives a reward of -1 at each time-step and a reward of +10 once it reaches a predefined goal. The test set (mazes where the identified macros will be evaluated) was composed of another 10 different environments with goal locations and starting points placed at random. The agent has at its disposal four different primitive actions: moving right (r), down (d), left (l) and up (u). The starting and goal locations are placed at random on these maps. We used tabular Q-learning to acquire policies for each task instance. In this experiment, Huffman Coding identified 11 macros of length 3, while LZW filled a 4-bit codebook containing 16 macros with lengths varying between 1 and 5. Table 1 shows the four most commonly-occurring macros identified by Huffman Coding (H) and LZW (L), and also their marginal probabilities. In this scenario, it is easy to interpret what the extracted macros achieve when used as exploration biases: they discourage the agent from repeatedly moving back and forth between a small number of states. Figure 3 shows a performance comparison for 4 different selected test environments (tasks) over 500 episodes. We compared the performance of an agent equipped only with primitive actions (black), an agent with 9 randomly-defined macros (green), an agent with macros extracted via Huffman Coding (red), and an agent with macros extracted via LZW (blue). The mean performance over the entire test set is shown in Figure 1.

5.2 Mountain Car Domain

In these experiments, we used the mountain car problem to define a family of related continuous-action tasks and evaluate our macro-identification method on such a setting. The state variables in this problem are given by a 4 dimensional vector representing the current position and velocity of the car in the x and y axis. The action space is limited to one-dimensional real values between \([a_{\text{min}}, a_{\text{max}}]\) representing a range of accelerations the agent is able to produce, where \(a_{\text{min}} < 0\) and \(a_{\text{max}} > 0\). The agent receives a reward of -1 at each time step and a reward of +100 if it reaches the goal position at the top of the mountain. To create different learning tasks, we defined several variations of the basic mountain car problem; these consist in changing the goal position, the maximum velocity in the positive and negative x-axis, and the maximum acceleration that the agent is capable of producing. The agent was trained for 200 episodes in 6 training tasks, and another 8 task variations where construct as testing tasks to evaluate the learned macros. In this setting, we obtained 7 macros of length 5 by using Huffman Coding and 16 macros of lengths ranging from 2 to 9 via LZW. Figure 5 shows the three most frequently-occurring macros identified by each method.

6 CONCLUSION

We have introduced a data-driven method for identifying recurring macros (frequently-occurring behaviors) observed as part of the solutions to tasks drawn from a family of related RL problems. Such macros were then used to define an exploration strategy for accelerating learning on novel tasks. Our method is based on identifying macros that allow for maximal compression of a set of sampled...
trajectories drawn from optimal policies for related tasks. A property of macros that allow for such a compression is that they correspond to recurring action patterns, and thus capture structure in the action sequences underlying the solutions to tasks from the domain. We formulated this problem as an optimization one and developed a sample-based approximation to its solution. We performed a series of experiments demonstrating the usefulness of the action patterns discovered by the method, and provided evidence of the benefits obtained by leveraging previous experiences to bias exploration on novel tasks, compared to using task-agnostic exploration methods. As future work, we would like to extend our method to closed-loop options; this could be achieved by identifying regions in the state space where stochastic policies for different tasks share similar action probabilities, or by optimizing an exploration policy directly (i.e., one that in expectation leads to rapid acquisition of near-optimal policies on tasks drawn from a common distribution or domain). In this work we focused on how an agent should behave when exploring by exploiting prior knowledge about related tasks. A natural follow-up question is when should the agent explore and whether we can design an experience-based approach to determining this as well. Finally, we believe that exploration methods that leverage previous experiences on related tasks may be useful in traditional multitask learning problems, in which policies for different tasks are transferred or reused.

REFERENCES

[1] Andrew G. Barto and Sridhar Mahadevan. 2003. Recent Advances in Hierarchical Reinforcement Learning. Discrete Event Dynamic Systems 13, 4 (2003), 341–379.

[2] Donald J. Berndt and James Clifford. 1994. Using Dynamic Time Warping to Find Patterns in Time Series. In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (AAAIWS’94). AAAI Press, 359–370.
(a) Macros extracted through Huffman Coding. Probabilities are shown inset in the legend.

(b) Macros extracted via LZW. Probabilities are shown inset in the legend.

**Figure 5: Sample continuous-action macros identified via Huffman Coding and LZW (mountain car domain).**

[3] Michael Buro. 1993. On the Maximum Length of Huffman Codes. *Inform. Process. Lett.* 45 (1993), 219–223.

[4] Justin Fu, John D. Co-Reyes, and Sergey Levine. 2017. EX2: Exploration with Exemplar Models for Deep Reinforcement Learning. *CoRR* abs/1703.01260 (2017). http://arxiv.org/abs/1703.01260

[5] Abhishek Gupta, Céline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. 2017. Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning. In *proceedings of the International Conference on Learning Representations (ICLR-2017)*.

[6] David A. Huffman. 1952. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the Institute of Radio Engineers* 40, 9 (September 1952), 1098–1101.

[7] George Konidaris and Andrew Barto. 2006. Autonomous shaping: knowledge transfer in reinforcement learning. In *proceedings of the 23rd International Conference on Machine Learning (ICML-2006)*. 489–496.

[8] George Konidaris and Andrew Barto. 2007. Building portable options: Skill transfer in reinforcement learning. In *proceedings of the 20th International Joint Conference on Artificial Intelligence*. 895–900. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.121.9726

[9] George Konidaris and Andrew G. Barto. 2007. Building Portable Options: Skill Transfer in Reinforcement Learning. In *proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*. 895–900.

[10] Ramnandan Krishnamurthy, Aravind S. Lakshminarayanan, Peyush Kumar, and Balaraman Ravindran. 2016. Hierarchical Reinforcement Learning using Spatio-Temporal Abstractions and Deep Neural Networks. *CoRR* (2016).

[11] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* (2015).

[12] Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. 2017. A Laplacian Framework for Option Discovery in Reinforcement Learning. *CoRR* (2017).

[13] Sridhar Mahadevan. 2005. Proto-value Functions: Developmental Reinforcement Learning. In *proceedings of the 22nd International Conference on Machine Learning (ICML-2005)*. ACM, 553–560.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. (2013).

[15] Andrew William Moore. 1990. Efficient memory-based learning for robot control. Technical Report. University of Cambridge, Computer Laboratory.

[16] Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. 2017. Actor-Mimic: Deep Multitask And Transfer Reinforcement Learning. In *proceedings of the International Conference on Learning Representations (ICLR-2017)*.

[17] Doina Precup. 2000. Temporal Abstraction in Reinforcement Learning. Ph.D. Dissertation. (2000).

[18] David Silver, G"{u}"{u}z Le"{v}er, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-2014)*, Tony Jebara and Eric P. Xing (Eds.). JMLR Workshop and Conference Proceedings, 387–395.

[19] N. Starrevant. 2012. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games* 4, 2 (2012), 144 – 148.

[20] Richard S. Sutton, Doina Precup, and Satinder P. Singh. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence* 112, 1-2 (1999), 181–211.

[21] Matthew E. Taylor and Peter Stone. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research* 10 (Dec. 2009), 1633–1685.

[22] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. In *Machine Learning*. 279–292.

[23] T. A. Welch. 1984. A Technique for High-Performance Data Compression. *Computer* 17, 6 (June 1984), 8–19. https://doi.org/10.1109/MC.1984.1659158
A APPENDIX

A.1 Proof of Theorem 3.1

Given a specific macro length \( l \), identifying all length-\( l \) action subsequences in a particular trajectory \( r \) with length \( |r| \) takes \(|r| - l \) steps. Let \( r^{\text{max}} \) be the longest trajectory in our sampled data. Since there are \(|T| \) trajectories and given that \(|r^k| \leq |r^{\text{max}}| \) for \( k = 1, \ldots, |T| \), identifying all candidate macros of length \( l \) that occur in the sampled data takes \( \sum_k^{|T|} |r^k| - l = O(|T||r^{\text{max}}| - l) \) steps. The quantity \(|r| - l \) is at its maximum when \( l = t_{\min} \) and the process of constructing codebooks for different candidate macro lengths \( l \) needs to be repeated \( t_\Delta = t_{\max} - t_{\min} \) times. Therefore, the complete pre-processing process has complexity \( O(t_\Delta|T|) \).

A.2 Proof of Theorem 3.3

To prove the upper bound on the number of bits used by LZW to encode a set of \(|T| \) trajectories, we first define a lower bound on the maximum length of a macro represented by a codebook of \( N \) symbols and a primitive action set of size \(|A|\). Assume, as an example, that \( A = \{a_1, a_2\} \) and \( N = 2 \). In this case, the longest macro considering all possible codebooks of length \( N = 2 \) would be at least of length 1, and the codebook \( C \) would be given by \( C = \{a_1, a_2\} \). If \( N = 6 \), a lower bound on the largest possible macro over all possible codebooks of such size would be of length 4; and the codebook would be given by \( C = \{a_1, a_2, \{a_1, a_1\}, \{a_1, a_2\}, \{a_2, a_1\}, \{a_2, a_2\}\} \).

In general, determining such a lower bound on the maximum length macro given a codebook of \( N \) symbols can be posed an optimization objective:

\[
\max_i \quad i + 1
\]

\[
\text{s. t.} \quad N - \sum_{j=1}^i |A|^j \geq 0
\]

First, note that the summation in the constraint functions above corresponds to a geometric series whose value is given by \( \sum_j^i |A|^j = \frac{|A|(1 - |A|^i)}{1 - |A|} \). Also note that the boundary of the constraint (i.e., the smallest value of \( i \) that does not violate it) can be found when \( N - \sum_j^i |A|^j = 0 \). Combining these two we obtain:

\[
0 = N - \frac{|A|(1 - |A|^i)}{1 - |A|}
\]

\[
|A|^i = 1 - N \frac{1 - |A|}{|A|}
\]

\[
i = \log_{|A|} 1 - N \frac{1 - |A|}{|A|}
\]

Since the maximization requires returning the largest value of \( i \) that does not violate the constraint, plus one, we take the ceiling of the quantity above:

\[
i = \left\lceil \log_{|A|} 1 - N \frac{1 - |A|}{|A|} \right\rceil
\]

Note that in the worst case (in terms of the number of bits needed to encode trajectories) we could use macros of length greater than 1 just once, and encode the remaining parts of the trajectories using only primitive actions. The total number of symbols used, in this case, would then be given by:

\[
\sum_k^{|T|} |r^k| - \sum_{j=1}^{i-1} |A|^j
\]

where \(|T| \) is the number of trajectories sampled and \(|r^k| \) is the number of primitive actions in the \( k^{\text{th}} \) trajectory.

Given that all symbols in the codebook have a binary encoding of length \( b_{\text{limit}} \), the total number of bits, \( B_t \), needed to encode all \(|T| \) sampled trajectories is upper bounded by:

\[
\left( \sum_k^{|T|} |r^k| - \sum_{j=1}^{i-1} |A|^j \right) \times b_{\text{limit}}
\]

where \( i = \lceil \log_{|A|} 1 - N \frac{1 - |A|}{|A|} \rceil \).