Lifelong Transfer Learning with an Option Hierarchy

Majd Hawasly Subramanian Ramamoorthy
School of Informatics, University of Edinburgh, United Kingdom, EH8 9AB.
Email: M.Hawasly@ed.ac.uk Email: S.Ramamoorthy@ed.ac.uk

***Abstract*— Many applications require autonomous agents to achieve quick responses to task instances drawn from a rich family of qualitatively-related tasks. We address the setting where the tasks share a state-action space and have the same qualitative objective but differ in dynamics. We adopt a transfer learning approach where common structure in previously-learnt policies, in the form of shared subtasks, is exploited to accelerate learning in subsequent ones. We use a probabilistic mixture model to describe regions in state space which are common to successful trajectories in different instances. Then, we extract policy fragments from previously-learnt policies that are specialised to these regions. These policy fragments are *options*, whose initiation and termination sets are automatically extracted from data by the mixture model. In novel task instances, these options are used in an SMDP learning process and option learning repeats over the resulting policy library. The utility of this method is demonstrated through experiments in a standard navigation environment and then in the RoboCup simulated soccer domain with opponent teams of different skill.**

I. Introduction

A. Lifelong Transfer Learning

In this work we propose a mechanism for transfer learning that supports an agent in a lifelong learning process. Here, a *lifelong* agent is one that is introduced continually to new problem instances in a domain. *Transfer* refers to the use of knowledge acquired in past instances to boost the ability to solve new instances. Given that the learning process is continual, it is essential to manage the acquired knowledge in a way that enables computationally tractable, quick use.

Then, by *lifelong transfer learning* we mean the agent's ability to continually learn, manage experience, and reuse it online in new instances drawn from a family of related tasks in the domain. Our formulation is related to many general models of transfer but our applications focus also implies differences. For instance, we assume that task instances can vary in fairly general ways, including arbitrary changes to opponent strategies, although we do not explicitly consider the problem of mapping between different state-action spaces. Our focus is on scenarios where the agent must respond *quickly* to tasks, making it infeasible to learn policies from scratch. While many interesting and practically useful domains require this kind of lifelong learning, we focus attention on a couple of domains where it is possible to experimentally induce substantial variability and evaluate the performance of the algorithm. One such domain is simulated robotic soccer [2].

In this paper, we consider transfer of reinforcement learning policies based on a continual policy abstraction process. Policy abstraction aims to exploit the structure in a policy/set of policies to accelerate planning/learning. We argue that many interesting tasks require key behavioural components, or subtasks, that are shared between instances but are *neutral to the variability in the instances*, and that capturing these would improve the agent's performance in a novel instance. We seek to automatically extract a set of such subtasks from the agent's experience, and continually refine the hierarchy as new instances are solved.

To do this, we employ a probabilistic unsupervised learning method to automatically identify a generalised concept of *state trajectory bottlenecks*, capturing interesting contiguous regions in state space that frequently show up in task instances. We combine that with a process of policy reuse which populates these regions with the exact same behaviours learnt in the previous instances, creating pieces of skill. The options framework is used to represent these abstract skills and the induced hierarchy. The process is iterated, uncovering core skills that are generally useful in solving multiple instances of the task.

B. Related Work

Learning to act in a set of related tasks by leveraging the experience gained in a small subset is a branch of Transfer Learning in reinforcement learning (see [3] for a review). The set of source tasks and the novel target task may differ in rewards, dynamics, or state-action space. Some successful methods rely on an explicit observable parametrisation of the task space (e.g. [4], [5]), while others assume a known distribution of variability (e.g. [6]), or approximate it from previously-seen source tasks (e.g. [7], [8]). Bayesian priors can be defined from these estimates (e.g. [9], [10], [11]). A different approach is to find an alternative task space in which the related tasks are effectively the same [12].

Policy reuse is a specific approach to transfer that deals with task families. With a mechanism to select the most similar previous instance to any given novel instance, learnt policies can be used *as is* [13], or used to bias the exploration towards accelerated learning [14].

Exploiting policy abstraction for transfer purposes has seen some success. Hierarchical policies were seen originally as a way to control complexity in reinforcement learning, and

* A preliminary version of this work appeared in [1].

a number of methods to organise and learn a hierarchical policy for a task were proposed in what became known as Hierarchical Reinforcement Learning (HRL) [15]. Later, the transfer potential of options [16] was examined. For example, [12] introduces portable options which are abstract actions defined not in the problem-space but rather in a reduced state space (the agent-space) to be transferred to any problem that shares that reduced representation. 'Skills' in [17] are extracted and chained to construct skill trees from expert demonstrations. Finally, generalisation for parametrisable tasks is achieved by the discovery of smooth low-dimensional spaces where their policies lie [5].

Learning the abstraction is essential in a lifelong learning framework. One way to abstract policies of a task is to discover essential subgoals, and a commonly used notion for this is that of a *bottleneck*: a landmark state which successful trajectories tend to go through, but not the unsuccessful ones. Bottleneck discovery has been approached in many ways, including state visitation frequencies [18], [19], graph-theoretic properties of the transition graph (e.g. Max-flow/Min-cut [20], betweenness [21]), among others. Many methods require discrete settings or an upfront complete knowledge of the task, but in [17] options are discovered automatically in a continuous space for a single task.

C. Unsupervised learning for lifelong transfer learning

Our approach depends on automatically extracting an option hierarchy from experience, with continual refinement. The concept of a bottleneck as a subgoal is a key concept in option discovery. However, assumptions such as that of discrete state spaces which are used in metrics like state visitation frequencies, or the sort of complete knowledge needed to compute centrality or betweenness, render the available methods unsuitable for continuous and otherwise complex domains. In the soccer example, we would anticipate that concepts of 'game play', rather than isolated salient states, are the key determinants of the ability to score. In [17], continuous 'target functions' are identified first, then options are learnt to achieve them. Alternately, we start by discovering domains of potential options from traces of previous successful trials. Using an EM (Expectation-Maximisation) procedure, we find a generative mixture model of the states of those good traces, then use the support of the components to define the option boundaries. In our example, this corresponds to identifying contiguous situations of play that are common in successful trials, and creating skills around them.

In HRL, policies are specifically learnt to achieve the discovered subgoals. We employ a process of policy reuse to populate the discovered option domains with action policies. That is, we 'borrow' from a previous instance a part of its learnt policy and use it *as is* in the new option, replicating policies in qualitatively similar situations.

D. Setup

We consider tasks that can be modelled as discrete-time Markov decision processes (MDPs). An MDP m is the tuple (S, A, T, R), where S is a bounded (possibly infinite) state space; A is a bounded (possibly infinite) action space; $T : S \times A \times S \to [0, 1]$ is the dynamics; and $R : S \times A \times S \to \mathbf{R}$ is the reward process encoding the goal of the task. A (Markov) policy for an MDP is a stochastic mapping from states to actions, $\pi : S \times A \to [0, 1]$, and the optimal policy π^* is the policy that maximises expected cumulative reward.

We consider episodic, goal-oriented tasks, in which termination occurs either by satisfying a *goal function* $g : S \to \{0, 1\}$, or when the episode elapses. The agent faces a sequence of instances of its task, each slightly different. We capture this variability by a set of MDPs $\mathcal{M}$ with a common objective, sharing the state-action space $S \times A$ and the reward R, but allowing for different dynamics T for each $m_i \in \mathcal{M}$. We do *not* assume that the model of variability in the task is known to the agent beforehand.

The framework of options [16] defines generic temporally-extended actions $\langle I, \pi, \beta \rangle$: $I \subseteq S$ is the initiation state set where the option can be invoked, π is a (Markov or semi-Markov) policy to follow when the option is invoked, and β is a probability distribution over the state space encoding the termination condition of the option.

Next, we describe our approach in detail. Then, we discuss extensions to improve efficiency and scalability (Section III). Finally, we demonstrate the merits of the framework in two domains: navigation in a rooms environment, and an attack drill in simulated robotic soccer (Section IV).

II. ILPSS: Incremental Learning of Policy Space Structure

Figure 1 gives a snapshot of the components of the framework and its operation, involving the following key steps:

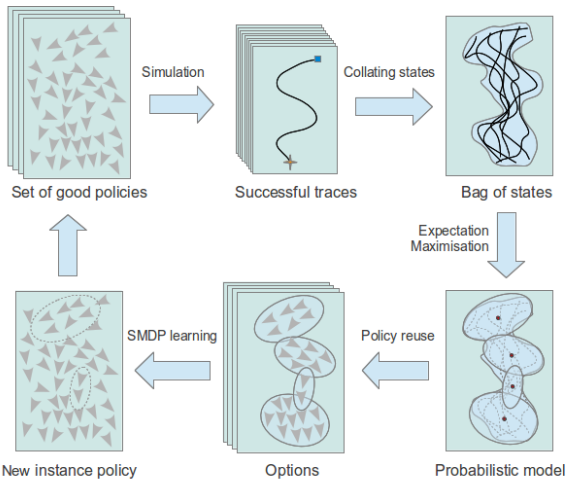

Fig. 1. A high level overview of ILPSS.

1) Starting from a collection of n policies $\pi_0, \pi_1, \ldots, \pi_n$ of some sample instances of the task $m_0, m_1, \ldots, m_n \in \mathcal{M}$, we sample a set of complete episode-long state traces $\tau = \{s_0, \ldots, s_{|\tau|-1}\}$, $s_j \in S$.
2) We label the trajectories with respect to their success $(+)$ or failure $(-)$ in reaching the goal. An episode is suc-

cessful if it terminates at the goal. Only the states in successful traces of instance m_i, $\boldsymbol{\tau}_i^+ = \bigcup_l \{\tau_l \mid g(s_{|\tau_l|-1}) = 1\}$, are used to make the state dataset $\mathcal{D} = \bigcup_{i=1}^n \boldsymbol{\tau}_i^+$.

3) To identify the contiguous regions in state space that show up frequently in successful trials we search for a probabilistic mixture model of $\mathcal{D}$. The combination of such a model with policy reuse is what defines ILPSS. In more concrete terms, we want to find the parameters θ of a model that maximises the log likelihood

$$\log P[\mathcal{D};\theta] = \sum_i \log P[\boldsymbol{\tau}_i^+|\theta] = \sum_i \sum_\tau \log P[\tau_i^+;\theta] \quad (1)$$

assuming independence of traces of an instance given the model, and independence of traces of different instances. That is, the model summarises all the correlations. Here, we ignore time and make the traces as *bags of states*,

$$\log P[\tau;\theta] = \sum_{s\in\tau} \log P[s|\theta] \quad (2)$$

We use to model $P[s|\theta]$ a mixture of $|K|$ multivariate Gaussian kernels, each represented by its mean μ_k and covariance matrix Σ_k in S,

$$\log P[s|\theta] = \log \sum_k p_k \mathcal{N}(s;\mu_k,\Sigma_k) \quad (3)$$

For our choice of model, the parameters are the weights $\{p_k\}$, means $\{\mu_k\}$ and covariances $\{\Sigma_k\}$, $k = 1\ldots|K|$. We allow $\{p_k\}$ to be *instance-specific*, i.e. each task instance i has it own weighting of the components ${p_k}^i$, while $\{\mu_k\}$ and $\{\Sigma_k\}$ are *instance-independent*, i.e. all the task instances share the exact same components. That is, we push toward finding a common set of mixture components across the different instance policies, regardless of their relative significance in the various instances. This shared structure is what we are after for lifelong transfer.

The mixture components define kernels in the state space, $\mathcal{K}_k(.) = \mathcal{N}(.;\mu_k,\Sigma_k)$. These are an important component of our representation of the policy space in that we assign policy fragments to the state regions defined by them, $S^k = \{s \in S : \mathcal{K}_k(s) > \Psi\}$ for some cutoff probability Ψ. For a specific kernel, we borrow a policy fragment from a learnt policy of a previous instance π_i by restricting it to the kernel's state space region: $\pi_i^k : S^k \times A \to [0,1]$. The scale (or weight) of a component k in a task instance i (captured by ${p_k}^i$) is ignored as it is irrelevant from a structure-learning point of view.

To fit the model, we set the desired number of kernels $|K|$ and use an adapted EM algorithm [22] that takes our structural constraint into account.

4) A discovered kernel k can spawn a set of policy fragments $\{\pi_0^k, \pi_1^k, \ldots, \pi_n^k\}$, each borrowed from a different policy. These will give a set of options $\mathcal{O}_k$, the domain and the termination conditions of which are both the PDF defined by the kernel. That is, an option from $\mathcal{O}_k$ is allowed to start at a particular state s with a probability equal to the normalised kernel function $\bar{\mathcal{K}}_k(s) = \mathcal{K}_k(s)/\max \mathcal{K}_k(.)$, and will continue stochastically with the same probability, terminating with probability $1 - \bar{\mathcal{K}}_k(s)$. That is, we are reusing policies of previous tasks in controlled regions of state space where they fared well in experience.

5) Adding the options to the action repertoire, the agent is presented with another task instance m_{n+1}, and a policy π_{n+1} is learnt. If the discovered options are not useful in solving the new instance, SMDP learning will fall back to learning using the primitive actions.

6) Adding π_{n+1} to the policy collection, the exact same procedure is repeated (traces, bag of states, probabilistic model, then options) in a lifelong learning process.

Algorithm 1 ILPSS: Incremental Learning of Policy Space Structure

Require: input policies $\pi_0, \ldots, \pi_n$, number of kernels $|K|$, cut-off probability Ψ.

Initialise the repertoire $\mathbf{O} \leftarrow \phi$.
for every new instance m_{n+1} **do**
  Generate state traces $\{\tau\}$ from input policies through simulation or runtime recording.
  Label and extract successful traces $\{\tau^+\}$.
  Create a 'bag of states' dataset $\mathcal{D} = \bigcup_{i=1}^n \{\tau_i^+\}$.
  Fit a probabilistic mixture model to $\mathcal{D}$, generating a set of kernels $K = \{\mathcal{K}_k(.)\}_1^{|K|}$ in state space.
  Extract state regions from the kernels, $S^k = \{s \in S : \mathcal{K}_k(s) > \Psi\}$, for $k = 1, \ldots, |K|$.
  Restrict input policies to kernel state regions, $\pi_i^k : S^k \times A \to [0,1]$, for all k and i.
  Create a set of options $\mathcal{O} = \bigcup_{k=1}^{|K|} \mathcal{O}_k$ with $o_i^k = \langle \bar{\mathcal{K}}_k, 1 - \bar{\mathcal{K}}_k, \pi_i^k \rangle \in \mathcal{O}_k$.
  $\mathbf{O} \leftarrow \mathbf{O} \cup \mathcal{O}$.
  Learn a policy π_{n+1} for the new instance with $\mathbf{O}$.
  Add π_{n+1} to the input policies.
  $n \leftarrow n + 1$.
end for
return Option set $\mathbf{O}$.

III. Scaling ILPSS

A. Managing experience

To control the number of maintained options, a process of temporal discounting (*forgetting*) is employed. The options that do not get used often enough will become less likely to be used afterwards and more likely to be forgotten. On the other hand, options that are used often will persist, and may develop and generalise through reuse inside future options.

One way to do this is by shrinking the support of the option in state space, making it less likely to be chosen later and concentrating it on a denser domain in state space. After crossing a threshold on effective option size Υ, the options can be pruned out of $\mathbf{O}$. In our implementation, we achieve forgetting by multiplying the covariance of the distribution Σ^o with a scalar $\xi < 1$ in every new instance in which the option is not used.

The procedure *Option Discounting*, that can be called after every ILPSS learning trial, is detailed in Algorithm 2.

Algorithm 2 Option Discounting

Require: options $\mathbf{O}$, newly-learnt policy π, forgetting parameter ξ, effective kernel size Υ.

1: **for** every option $o \in \mathbf{O}$ not used in π **do**
2: $\quad \Sigma^o \leftarrow \xi\Sigma^o$.
3: $\quad$ Update the option domain S^o.
4: $\quad$ **if** $\int \mathbb{1}_{S^o}(s)\, ds < \Upsilon$, with $\mathbb{1}_{S^o}(.)$ being an indicator function, **then**
5: $\quad\quad \mathbf{O} \leftarrow \mathbf{O}/o$.
6: $\quad$ **end if**
7: **end for**
8: **return** Option set $\mathbf{O}$.

B. Trace sampling

Density estimation methods in general, including EM, tend to be computationally expensive, especially with the increase in the size of the input dataset. In ILPSS, the dataset represents states that are used in successful trials more often as the agent is experiencing more instances. To control the size we employ two down-sampling measures. First, we choose a subset of previous instances to be included in $\mathcal{D}$ after learning a new instance rather than using them all. The choice is based on similarity to the newly-learnt instance. This will still uncover commonalities in the included instances and encourage generalisation and reuse. If similarity cannot be measured, instances can be sampled randomly subject to the risk of generating less 'meaningful' options.

The second measure is to sample states from the chosen traces, up to the desired dataset size. This depends on the continuity of the state space and the probabilistic fitting method to overcome the representation gaps.

IV. Empirical Results

A. Rooms environment

The aim of this experiment is to test the lifelong learning aspect of the proposed framework by visualising the set of skills that can be produced and maintained after experiencing many instances of a specific domain. This task, in a discrete domain, requires an agent to navigate in a rooms environment from some *random* initial position to a *random* goal position. The domain is a 2D grid world of size 25×25 with walls and exits. After completing each trial, the agent uses ILPSS to generate a set of options that are used in the next trials.

In this experiment, $|K|$ is set to 4, and the forgetting ξ is 0.8. The agent receives -1 for each time step, and 10 for reaching the goal. After experiencing 25 instances of the task, the agent ends up with the skills shown in Fig. 2.

In Fig. 3 we show a heat map of the visitation frequency of states from traces, which is one metric to identify bottlenecks in discrete domains. The white dots on the figure are the means of ILPSS kernels extracted using the same traces. What this shows is that the framework is able to approximate the discrete bottlenecks.

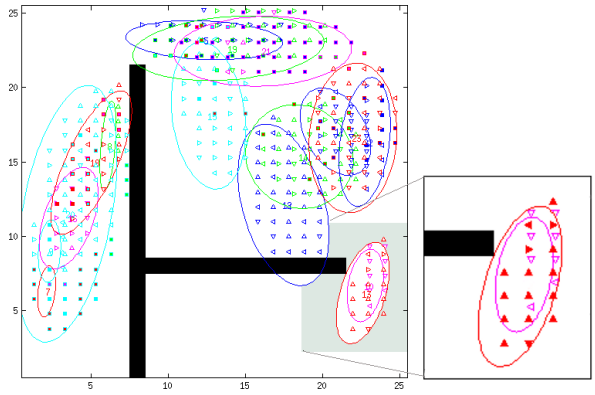

Fig. 2. ILPSS in the rooms environment. LEFT: acquired options after solving 25 instances of the task. The black lines are walls, and each ellipse defines an option (70% support of a kernel). The arrows show option policies, which move the agent through different rooms and exits. RIGHT: two options that appear in the highlighted bottom-right corner in the domain, zoomed-in. They allow passing in and out of the room through the exit.

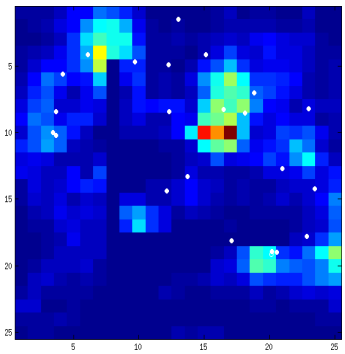

Fig. 3. The visitation frequency of the rooms environment after 30 random instances. The darker the colour, the less-visited the state. The superimposed white dots show the means of ILPSS kernels learnt using the same instances.

B. RoboCup simulated robotic soccer

The aim of this experiment is to test the transfer capability of the proposed framework in a continuous domain. The task models a training drill for 2 vs. 2 players, in which the team with ball possession tries to cross a specific line in the field *with the ball*. The task is episodic, starting with the agents in set positions, and terminating either successfully when the goal is achieved, or otherwise if the adversaries intercept the ball, kick it out of the training region (35m×35m), or the episode elapses (100 time steps). The experiment is conducted using RoboCup 2D Simulation League Soccer Server [23] and the Keepaway extension [2]. The setup is shown in Fig. 4.

The state space is defined using 9 continuous state features describing the position and orientation of the agent with respect to other agents and the goal line. The action space comprises 3 basic options: holding the ball, passing to a team mate and dribbling toward the goal. The action set is enriched later with the discovered options.

The learning agent is the striker with the ball, while others use hand-tuned stochastic behaviours. The variability in defenders' behaviour is manifested by different tendencies

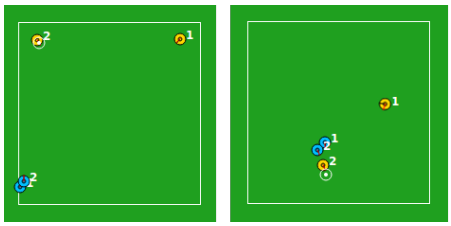

Fig. 4. The experiment setup. LEFT: the attacking team starts from the top corners, the defending team starts from the bottom. RIGHT: the task of the attackers is to reach the bottom line with the ball, with the middle point having the highest reward, while the defenders are trying to prevent that.

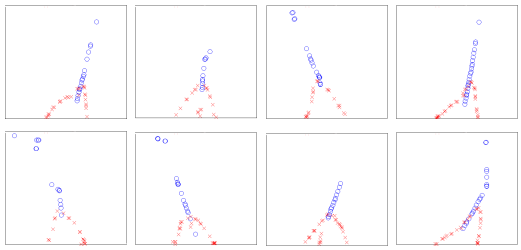

Fig. 6. Traces through time from ILPSS options in the simulated soccer domain. The blue circles are locations of the learning agent in the field moving toward the goal line. The red crosses are for the two adversaries.

toward the ball and the goal line. In the experiment, four opponent teams are used: the first two are only concerned with the ball, having different coordination protocols between the players; the third type has one opponent which intercepts the ball while the other protects the goal line; the fourth sees both opponents protecting the goal with a small probability of going to the ball, making it different and tougher to beat.

The opponents are presented to the learning agents sequentially, with ILPSS running after each to generate 5 options. After experiencing the first three opponents and acquiring 15 options, the performance against the fourth is compared to learning the same task from scratch in Fig. 5.

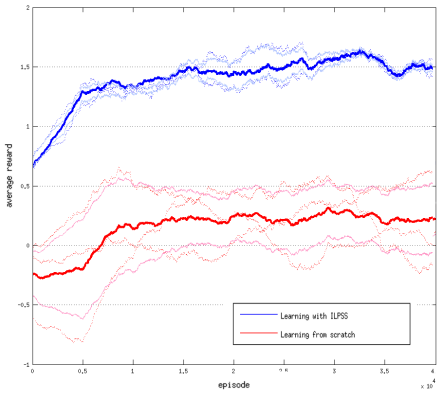

Fig. 5. Average reward against the fourth opponent team using ILPSS options of three other instances (BLUE), compared to learning anew (RED).

As the results show, using the set of options discovered by ILPSS gives a head start in performance. This shows that ILPSS is able to produce useful abstractions in the policy space allowing faster convergence in novel instances. We show in Fig. 6 example traces generated by the most used options. It appears that the agent developed a behaviour to approach the adversaries slowly closer to the middle before beating them to the goal line at a short range, which seems to be a sensible approach considering the opponents' conservative behaviour.

V. Discussion

ILPSS is built around fitting a mixture model to the states of successful trajectories in a task family to uncover useful commonalities. This assumes that task instances do share common structure, complete traces can be generated, and a probabilistic model can be fit to the resulting state dataset. In a very high-dimensional, complex domain that does not satisfy all these assumptions ILPSS may need extensions. The specific techniques and models used in this paper (e.g. EM and GMMs) are not essential to the operation of ILPSS; these can be replaced with other state-of-the-art tools to better tackle high dimensionality. Combining this framework with dimensionality reduction method is also a topic of future work.

The intuition behind the incremental, lifelong nature of the framework is in 'biasing' the evolved hierarchy toward finding common components. Reusing the previously-learnt policies in options while learning a new instance encourages the agent to generalise and evolve useful, stable behaviours, while allowing it to fall back to complete policy search in extreme cases. We argue that this is the kind of structure needed to face a novel instance of a task. However, a biased experience may cause a bias in abstraction, delaying the convergence into a meaningful skill set. This is analogous to negative transfer in transfer learning, and similar solutions may be needed.

Traditional bottleneck discovery methods work best for single tasks with small, discrete state spaces for which complete interaction graphs can be built or sufficient visitation information can be collected. The aim of these methods is to locate potential subgoals, leaving the policies to be subsequently learnt. On the other hand, ILPSS generalises that concept of a bottleneck into a continuous, probabilistic model in state space, making it more appropriate in large and continuous domains. Also, it immediately discovers where the *existing* policies might be useful, avoiding the two-step process of subgoal discovery followed by policy learning. ILPSS may appear close in spirit to skill trees of [17] but the assumptions are different, with ILPSS being devised to extract common structure in many instances of a task, rather than backchaining to solve a specific task.

An earlier method that defines skills with probabilistic domains is the SKILLS algorithm [24] where the aim is to find a set of macro-actions for a single task with a minimum description length. The policy of a skill is learnt in a way that balances performance loss with compactness gains. ILPSS policies are chosen to support an extensive family of tasks, while achieving compactness through option discounting.

The work in [25] investigates the structure in the space

of value functions for optimal policies of related tasks. They employ a mixture model as the generative process of value functions, with components representing the discontinuity in value due to inherent properties of the task/domain (e.g., location of walls in a room environment). They use the model to accelerate learning in new instances by augmenting the state space with the discovered high-level features. ILPSS defines structure in policy space *implicitly* via a probabilistic model in state space with reusable policy fragments. The components in state space do not necessarily feature smooth value functions but rather a useful and consistent policy. ILPSS only requires sample traces compared to complete explicit representations (e.g. value functions or transition graphs), making it usable even before learning converges, especially in big worlds.

Using Gaussian kernels for option domains may not always be the optimal choice as they suggest symmetry across all dimensions, but they stand as an intuitive choice from a computational point of view. More work is needed on defining option boundary in state space (e.g. a mixture model per option). Some previous work used probabilistic models in HRL. [26] defines a more-complicated graphical model for both state and policy, trained from sample trajectories using EM, but constrained to a single task.

VI. Conclusion

We introduce a framework for learning and refining a structural description of the space of policies for a set of qualitatively-related task instances. The aim of ILPSS is to enable an agent to react quickly in novel instances of the same task. We employ a principled probabilistic method to decompose the state space, and relate the learnt abstraction with policy fragments through policy reuse. The resulting structure is maintained using a set of temporally-extended options. We note that learning continually is essential for extracting useful decompositions in policy space.

ILPSS does not require explicit representation of the space of policies, and it does not rely on the optimality of the input policies, allowing it to scale well. Only a set of trajectories of successful trials and the policies that generated them are needed to enable the agent to produce a rough and quick solution to a novel instance, then refined by learning. Testing this framework on larger problems to understand its scalability is a topic of future work, as well as understanding the effects of state abstraction on the learnt action abstraction.

ACKNOWLEDGMENT

This work has taken place in the Robust Autonomy and Decisions group in the School of Informatics, University of Edinburgh. Research of RAD is supported by the UK Engineering and Physical Sciences Research Council (grant number EP/H012338/1) and the European Commission (TOMSY Grant Agreement 270436, under FP7-ICT-2009.2.1 Call 6).

References

[1] M. Hawasly and S. Ramamoorthy, "Lifelong learning of structure in the space of policies," in *Lifelong Machine Learning, AAAI Spring Symposium Series*, 2013.

[2] P. Stone, R. Sutton, and G. Kuhlmann, "Reinforcement learning for robocup soccer keepaway," *Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, 2005.

[3] M. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *The Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[4] N. Mehta, S. Natarajan, P. Tadepalli, and A. Fern, "Transfer in variable-reward hierarchical reinforcement learning," *Machine Learning*, vol. 73, no. 3, pp. 289–312, 2008.

[5] B. D. Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," in *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ser. ICML '12, J. Langford and J. Pineau, Eds. New York, NY, USA: Omnipress, July 2012, pp. 1679–1686.

[6] T. Perkins, D. Precup, *et al.*, "Using options for knowledge transfer in reinforcement learning," *University of Massachusetts, Amherst, MA, USA, Tech. Rep*, 1999.

[7] F. Tanaka and M. Yamamura, "Multitask reinforcement learning on the distribution of mdps," in *Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on*, vol. 3. IEEE, 2003, pp. 1108–1113.

[8] M. Snel and S. Whiteson, "Multi-task reinforcement learning: shaping and feature selection," *Recent Advances in Reinforcement Learning*, pp. 237–248, 2012.

[9] F. Sunmola and J. Wyatt, "Model transfer for markov decision tasks via parameter matching," in *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, 2006.

[10] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-task reinforcement learning: a hierarchical bayesian approach," in *Proceedings of the 24th intl. conference on Machine learning*. ACM, 2007, pp. 1015–1022.

[11] A. Wilson, A. Fern, and P. Tadepalli, "Transfer learning in sequential decision problems: A hierarchical bayesian approach," in *ICML 2011 Unsupervised and Transfer Learning Workshop. JMLR W&CP*, 2012.

[12] G. Konidaris and A. Barto, "Building portable options: Skill transfer in reinforcement learning," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, vol. 2, 2007, pp. 895–900.

[13] S. Ramamoorthy, M. H. Mahmud, M. Hawasly, and B. Rosman, "Clustering markov decision processes for continual transfer," School of Informatics, University of Edinburgh, Tech. Rep., 2013.

[14] F. Fernández and M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 2006, pp. 720–727.

[15] A. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.

[16] R. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[17] G. Konidaris, S. Kuindersma, A. Barto, and R. Grupen, "Constructing skill trees for reinforcement learning agents from demonstration trajectories," *Advances in neural information processing systems*, vol. 23, pp. 1162–1170, 2010.

[18] M. Stolle and D. Precup, "Learning options in reinforcement learning," *Lecture Notes in Computer Science*, vol. 2371, pp. 212–223, 2002.

[19] Ö. Şimşek and A. Barto, "Using relative novelty to identify useful temporal abstractions in reinforcement learning," in *Proceedings of the 21st intl. conference on Machine learning*. ACM, 2004, p. 95.

[20] I. Menache, S. Mannor, and N. Shimkin, "Q-cut: dynamic discovery of sub-goals in reinforcement learning," *Machine Learning: ECML 2002*, pp. 187–195, 2002.

[21] Ö. Simşek and A. Barto, "Skill characterization based on betweenness," in *In Advances in Neural Information Processing Systems*, 2009.

[22] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38, 1977.

[23] I. Noda and H. Matsubara, "Soccer server and researches on multi-agent systems," in *Proceedings of IROS Workshop on RoboCup*, 1996.

[24] S. Thrun, A. Schwartz, *et al.*, "Finding structure in reinforcement learning," *Advances in neural information processing systems*, pp. 385–392, 1995.

[25] D. Foster and P. Dayan, "Structure in the space of value functions," *Machine Learning*, vol. 49, no. 2, pp. 325–346, 2002.

[26] V. Manfredi and S. Mahadevan, "Hierarchical reinforcement learning using graphical models," in *Proceedings of the ICML05 Workshop on Rich Representations for Reinforcement Learning*, 2005, pp. 39–44.