Using Uniform State Abstractions For Reward Shaping With Reinforcement Learning

John Burden
Department of Computer Science, University of York
York, UK
jjb531@york.ac.uk

Daniel Kudenko
Department of Computer Science, University of York
York, UK
National Research University Higher School of Economics
Saint Petersburg, Russia
JetBrains Research
Saint Petersburg, Russia
daniel.kudenko@york.ac.uk

ABSTRACT

Traditional reinforcement learning methods typically take a very long time to learn satisfactory policies for most complex problems. This can be alleviated with the introduction of knowledge based reward shaping, in which a domain expert provides knowledge to the agent in order to guide it towards better performance. The main drawback to this is the cost of encoding such knowledge, or the availability of domain experts. In this paper a method is proposed to relieve the need for some of the knowledge required by automating the generation of Abstract Markov Decision Processes (AMDPs) using uniform state abstractions. This paper then explores the effectiveness and efficiency of different resolutions of states abstractions and finds that many of these fare well even when compared to hand-crafted AMDPs for variations of the Flag Collection domain.

KEYWORDS

Reinforcement Learning, Reward Shaping, Uniform Abstraction

1 INTRODUCTION

Reinforcement Learning (RL) has shown itself to be a successful machine learning paradigm, learning to complete tasks principally through interaction with an environment and receiving feedback. This feedback is used to alter the behaviour of the agent towards receiving an increased level of positive feedback. Like many machine learning techniques, RL suffers from the 'Curse of Dimensionality' [1]. This causes there to be exponentially many state combinations for each state-dimension, and crucially limits the use of traditional RL to small scale domains.

One method of overcoming this limitation is to impart domain knowledge to the agent in order to guide it towards more rewarding behaviour. This can be achieved by the process of Reward Shaping [9], where extrinsic reward is given to the agent in a manner corresponding to domain knowledge in addition to the normal intrinsic feedback it receives from the environment. If the extrinsic reward is defined in a specific manner, then the ultimate behaviour learned by the agent will not change [8].

This, however, assumes that domain knowledge is readily available and encoded for use by the agent. Such knowledge is often expensive to encode or impractical to obtain. Some approaches to automatically generate the shaping function have been made [6], but these require analysis of the state-space in order to find sub-goals for the agent. This is a costly process.

Solving an abstract version of the original task, has proven to be useful for finding shaping functions [3][5][7]. However, this still requires domain knowledge, albeit less, in the form of appropriate abstractions of states, transitions and rewards.

The aim of this paper is to present a conceptually simple, yet surprisingly effective method for constructing spatial abstractions of environments with a spatial structuring. The agent can then solve these in order to yield a useful reward shaping function. This can then improve the overall performance of the agent. Although the environments used are limited to ones with spatial structure, this method should be generalisable to temporal abstractions also.

The rest of the paper will be organised as follows: First, the relevant background knowledge and terminology is given. Then the proposed method is detailed. The domains that the method was tested against are then shown. The results of these experiments are displayed and analysed. Finally, future avenues for this research are considered.

2 BACKGROUND

2.1 Reinforcement Learning

Reinforcement Learning consists of interaction between an agent and environment [10]. The agent observes how the environment responds to the agent's actions and updates its behaviour accordingly. The response of the environment is typically a numerical reward that depends on the propriety of the action made by the agent for the current state of the environment at that time. Typically, this interaction is modelled by a Markov Decision Process (MDP).

An MDP is a tuple (S, A, R, P) where S is a set of states in which the environment can inhibit, A is the associated action space detailing available actions for each state, $R(s, a, s')$ is the immediate reward the agent receives after transitioning from state s in the environment to s' via action a. $P(s, a, s') = \mathcal{P}(s' \mid s, a)$ — that is, the probability of transitioning to state s' when in state s and action a is performed. A policy π for an MDP is a function mapping states to the action for the agent to perform in that state. The goal of solving an MDP is to find a policy that maximises the expected cumulative reward (often called the *return*). The policy achieving this is referred to as optimal.

Iterative approaches are often used to find the optimal policy. Algorithms based on these approaches often utilize temporal-difference (TD) updates, which update values of states $V(s)$ or state-action pairs $Q(s, a)$, based on estimates of these value functions at different times.

One fundamental TD algorithm is called Q-Learning [10], and after each transition $s \xrightarrow{a} s'$, updates the Q values:

$$Q(s, a) = Q(s, a) + \alpha(R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

Here α is the learning rate, a measure indicating how to weight new information Q-values against old, γ is the discount factor, that weights regard for future reward as opposed to immediate.

When selecting actions, it is important to balance both "exploration and exploitation" [10]. This is necessary in order for the agent to avoid becoming trapped within locally optimum policies. To achieve this balance, the agent, with probability ϵ, selects an available action at random. With probability $1 - \epsilon$ the agent selects the action a, when in state s, that maximises the value of $Q(s, a)$.

In order to speed up the learning process, the agent can make multiple updates for each observation, following the trajectory of state-action pairs of which the agent has recently directly experienced. Each of these state-action pairs is recorded, and each of these pairs has an associated value referred to as its eligibility. This eligibility value is multiplied by the the temporal difference to create the new update value. This is done for all pairs in the trajectory. After an action is taken, the current state-action pair has its eligibility set to 1, and the other state-action pairs in the trajectory have their eligibility multiplied by some parameter $\lambda \leq 1$. The addition to Q-Learning speeds up propagation of the TD values through the state-action space. Q-Learning using these eligibility traces is referred to as $Q(\lambda)$-Learning [10].

Despite the speed up in the learning process given by $Q(\lambda)$-Learning, the learning process is still intractably slow for many complex problems. This is due to the aforementioned Curse of Dimensionality. In order to overcome this issue, some form of generalisation or conveyance of knowledge to the agent is required. This motivates the creation of new models for RL to use in order to utilise these these approaches.

2.2 Abstract Markov Decision Processes

An Abstract Markov Decision Process (AMDP) is, much like an MDP, a tuple (T, A, R, P). T is a set of *abstract* states. A is a set of abstract actions, corresponding to chains of primitive actions in the MDP. R is an abstract reward function, with $R(t, a, t')$ denoting the reward given when an agent moves from abstract state t to t' using abstract action a. Finally, P is the transition probability function, working in the same way as for an MDP. There exists also an abstraction function Z such that for $s \in S$ of the associated MDP $Z(s) = t$ denotes that state s is encapsulated by abstract state t. AMDPs are useful generalisations of large MDPs that reduce large, intricate problems to smaller, simpler ones.

2.3 Reward Shaping

One form of giving knowledge to the agent is reward shaping. Reward shaping consists of giving the agent some additional reward $F(s, a, s')$ that represents prior knowledge. This additional knowledge intuitively is used to steer the agent in the right direction. With this additional reward the Q-Learning update becomes:

$$Q(s, a) = Q(s, a) + \alpha(R(s, a, s') + F(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

It has been shown that if this additional reward is given as the difference of the potential of the two states, that is,

$$F(s, a, s') = \gamma\phi(s') - \phi(s)$$

then the optimal policy will not change as a result of this extra reward [8]. Without this potential based reward shaping — using just arbitrary extra reward — the optimal policy can indeed change [9]. The main concern with reward shaping from a practical perspective is the origin of the potential function ϕ. Defining some function that gives an indication of the "quality" of a transition is often time consuming or infeasible, due to the sheer number of such transitions.

2.4 Abstract Reward Shaping With AMDPs

One method of inducing a potential function with comparatively little external input is to solve an abstract version of the problem using an appropriate AMDP [3]. If this is done using value iteration, then there will be a value $V(t)$, for each abstract state $t \in T$ of the AMDP. For each state $s \in S$ of the MDP, the potential function for reward shaping can then be set to $\phi(s) = \omega V(Z(t))$, for some ω used to tune the weighting given to this external reward, and the state abstraction function Z.

3 DOMAIN EXPERTS AND KNOWLEDGE REVISION

Domain experts can be used in order to construct appropriate AMDPs for specific environments [3]. This involves defining an abstraction function Z mapping low-level states to abstract states, as well as transition and reward functions for the AMDP. This hand-crafted AMDP can then be easily solved using value iteration before the learning process begins. Then it can be used for reward shaping in the manner given in section 2.4.

Whilst this approach certainly requires less input than defining ϕ over every state, it can still be considerable work for the domain expert. Moreover, the suitable state abstractions may even be only partially known, or not known at all.

Domain experts are not infallible and will potentially make mistakes. Inaccuracies in reward shaping can lead to a slower convergence rate [2]. This has been addressed using knowledge revision for AMDPs [4]. In this approach, the abstract agent updates its transition probabilities to reflect its experiences. During each episode, the agent records which abstract transitions $t \xrightarrow{a} t'$ are used. At the end of the episode, the agent applies the following update rule to each transition in the AMDP. The AMDP is then resolved.

$$P(t, a, t') = \begin{cases} P(t, a, t') + \alpha(1 - P(t, a, t')), if\, t \xrightarrow{a} t' \text{occurred} \\ P(t, a, t') + \alpha(0 - P(t, a, t')), \text{otherwise} \end{cases}$$

Principally, doing this causes states that are not visited to have a lower probability attached to their associated transitions. States may not be visited due to incorrect domain knowledge or due to poor expected long-term reward. This yields lower values for $V(s)$,

which in turn reduces the probability that AMDP will attempt to utilize these states for shaping. The downside to this approach is that resolving the AMDP can be very costly, depending on its size.

4 EXPERIMENTAL DOMAIN

The environment that will be used to evaluate various abstractions for RL is the Flag Collection domain.

The Flag Collection domain is an augmentation of the navigational Gridworld environment. The agent is tasked with traversing a grid of cells, locating and picking up flags and taking them to the goal. The states of the environment are the grid cells, as well as which flags have been picked up so far. The agent can move in the four cardinal directions one cell at a time. In this environment the agent is given reward zero after almost every transition. The exceptions to this are when it reaches the goal, at which point the agent receives a reward proportional to number of flags collected (1000 times the number of flags collected). The agent also receives a smaller reward of 100 when it picks up a flag. The agent's task is made more difficult by impassable walls, spread throughout the domain. Further, the domain does not "roll", the agent cannot move out of the area and reappear on the other side. Episodes within the flag domain only terminate when the agent reaches the goal state.

The Flag Collection environment has intuitive abstractions. Abstract states are in the form of 'rooms', which are defined as collections of adjacent cells. Rooms are often formed naturally by walls within the environment, which limit movement. Abstract transitions are simply transitions between rooms instead of individual cells.

The Flag Collection domain was also selected for its scalability — it can be arbitrarily large with many flags. It also has been extensively used to evaluate RL algorithms and therefore is a solid choice for comparing the results here against benchmarks.

Different factors of this domain may markedly change the difficulty for an agent to complete its task. These factors may include parameters such as the connectivity of the rooms making up the environment, the presence of wide open spaces, long corridors, the total number of states, among others. In order to assess the utility of any RL method it seems prudent to test the method on multiple Flag Collection domains, each expressing a factor that may affect performance. As such, six variations on the domain were evaluated in this paper. These variations are shown in figure 1.

It is worth mentioning that not all environments naturally have such a spatial structure like the Flag Collection domain. In Section 7.1.1 it is described how this method could be extended to deal with temporal abstractions.

These variations are additionally described in Table 1.

5 METHOD

Bestowing to the agent knowledge from a domain expert, whilst having been shown to be helpful, is sometimes intractable or impractical. This is often due to lack of knowledgeable domain experts or the expense of encoding the appropriate knowledge. The method presented here attempts to recover some of the benefits of providing such knowledge, without the associated costs or time sinks. This is achieved using a simple uniform partitioning of the environment into abstract states and combining this with AMDP-based reward shaping.

For the purposes of this paper, a 'tiling' of an environment will refer to a partition of the environment into uniform shapes of equal size. The size of these tilings will be denoted by how many constituent concrete states make up each individual tile. For example, if a two dimensional environment is tiled using a 3×3 tiling, then each tile is a square containing 3^2 cells.

In the aid of reducing confusion over nomenclature, when referring to the distinct method of 'Tile Coding' [10], the full name shall be used in order to distinguish it from the method presented here.

To begin the method presented by this paper, take an environment and tile it using an appropriate sized tiling — the appropriate size will often depend on time available and the size of the environment — it can be considered a hyper-parameter of the system. This tiling partition is then to be used to construct the corresponding AMDP in order to be used for AMDP reward shaping. The state abstraction function will be given as the mapping from concrete state to tile. The abstract actions available in each state are moving to adjacent tiles in the abstract state space.

The abstract reward is defined in such a way to represent a simplified version of the reward, essentially representing a manner of measuring the "success" of the agent. Abstract rewards are usually quite sparse. For instance, there may be zero reward entirely until the end of an episode, where a single evaluation of performance is given, say for a number of tasks completed. In order for this to be realisable, the abstract agent may need to be told in which abstract states certain tasks can be completed. This, unfortunately, is provided by external domain knowledge. However this was always present in hand labelled abstractions used in previous work [9]. There is still significantly less information given to the agent, as the abstract states or transitions in this method are determined automatically. From needing domain knowledge of the abstract states, available transitions and abstract rewards, this method reduces the required knowledge to just abstract rewards.

Once this AMDP is constructed, value iteration is used to calculate $V(s)$ for each abstract state. Then, as is typical for AMDP reward shaping, the extrinsic reward is defined as

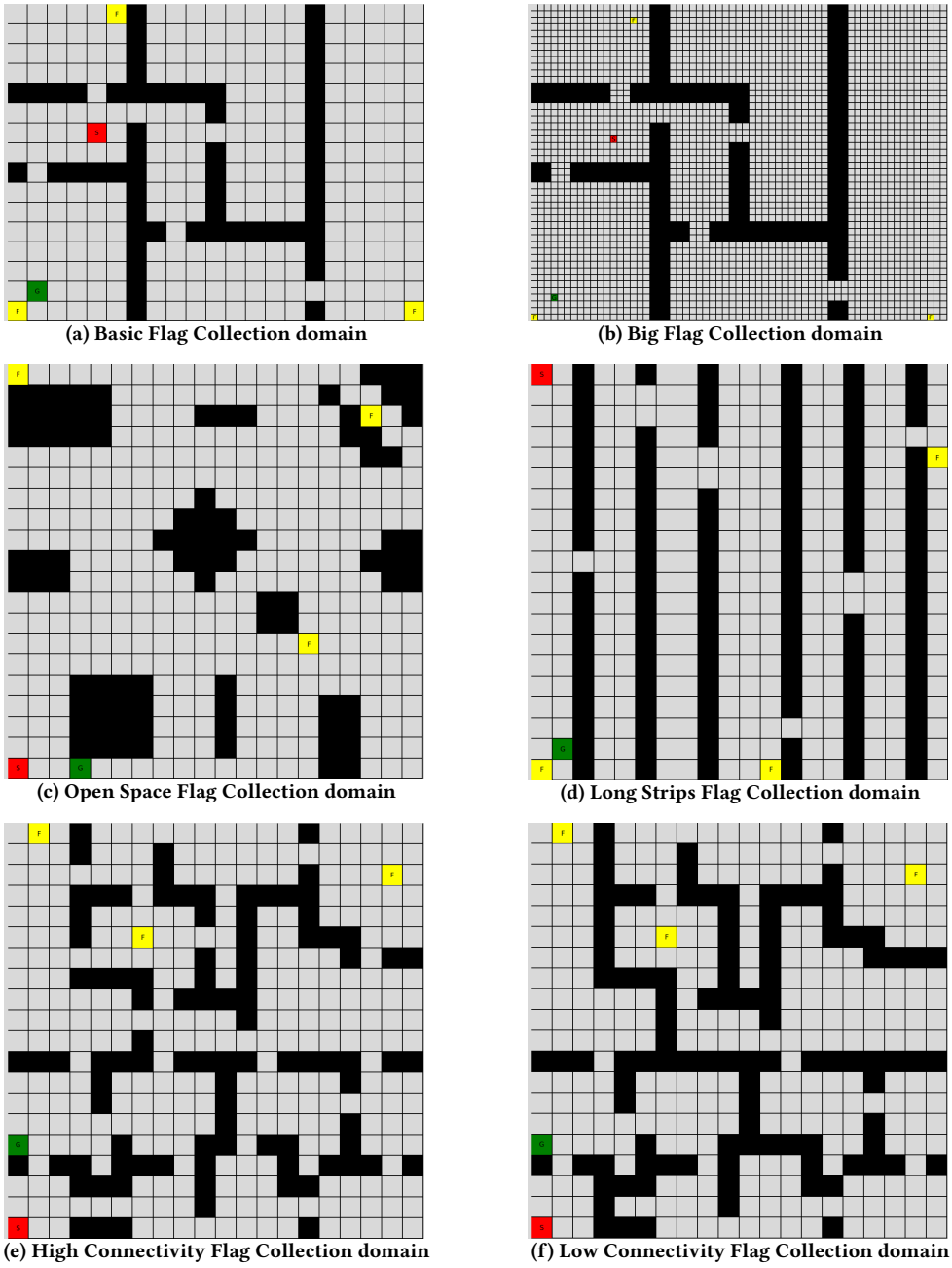
$$F(s, a, s') = \gamma\phi(s') - \phi(s) = \gamma\omega V(Z(s')) - \omega V(Z(s))$$

where Z is the state abstraction function.

5.1 Difference From Tile Coding

It is important to distinguish the method proposed by this paper from Tile Coding. Tile Coding is a technique for RL to help the agent to generalise states, especially for continuous domains [10]. Tile Coding does this by having a number of 'Tilings' that are larger than the state space. Each tiling consists of a number of 'active tiles' which partition the state space. All of the tilings are offset from each other by a known amount. This possibly changes which active tiles each state is part of for each tiling. Each state now belongs to exactly one active tile for each tiling. This leads to a feature vector representation mapping local groups of states to the same vector. These vectors are then used for function approximation RL methods, where the agent learns over the Tile-space.

Whilst Tile Coding seems similar to the method proposed at first glance, there are a few important differences to note.



(a) Basic Flag Collection domain

(b) Big Flag Collection domain

(c) Open Space Flag Collection domain

(d) Long Strips Flag Collection domain

(e) High Connectivity Flag Collection domain

(f) Low Connectivity Flag Collection domain

Figure 1: Graphic representation of each variation on the Flag Collecting domain. The agent begins at the red cell marked 'S' and must traverse to the green cell marked 'G', whilst moving through yellow flag cells marked 'F'.

First is the fact that due to the offset of each Tiling in Tile Coding, updates to one specific feature vector can affect surrounding feature vectors also, namely the ones that share other active tiles. This is not possible in the proposed method, since only one Tiling is used, and therefore no active tiles are shared. Whilst this may seem like a disadvantage for the proposed method, it does allow this method to view the active tiles as abstract states in an AMDP. This allows the solving of this AMDP and use in reward shaping. This is not possible with Tile Coding whilst retaining the sharing of active tiles in feature vectors.

Secondly, and more importantly, is that when Tile Coding is used with Function Approximation RL methods, the agent loses the ability to distinguish between states mapping to the same feature vector. Indeed, this is the whole point, as it reduces the different number of state-action pairs to learn values for. When the actions of the agent are limited, this can hinder the ability of the agent to

Name	Description
Basic	This is the basic environment for the Flag Collection domain. This has been used to test many RL algorithms previously and serves as a standard benchmark [2][4].
Big	The 'Big' variation is the same as the the basic one, except every cell in the basic environment becomes a set of 3x3 cells. This increase in size will test the scalability of the method.
Open	This variation is mostly open space with a few obstacles to traverse around. This variation was selected to see how the agent handles a lot of wide open space and many action choices available.
Strips	The 'Strips' variation is composed of rooms of long, thin strips. These were chosen to directly contradict the assumptions made by the upcoming method that abstractions are uniformly square. This was done in order to see if the agent is able to adapt to environments that are very different from their abstract representation.
High Connectivity	The 'High Connectivity' environment is full of rooms that are very interconnected. This fits with the abstract agents assumption that it can always transition from one room to the next.
Low Connectivity	The 'Low Connectivity' variation uses the same room layout as the 'High Connectivity' variation, except now the agent cannot transition between most rooms. More walls are present to block off its movement. Both the high and low connectivity versions were selected in order to compare how the agent copes when transitions the abstract agent uses for shaping do not exist.

Table 1: Descriptions of the variations on the Flag Collection domain that were selected.

navigate the state space successfully, as before it may have chained together different actions within one group of states that map to the same feature vector. Using Tile Coding, it cannot do this, and in each feature vector, the agent can perform only one action. In environments that require high levels of precision — such as the Flag Collection domain — this can lead to the agent becoming unable to complete the task. This disadvantage is not shared with the proposed method, the agent never loses the ability to distinguish between different states, regardless of which tile these states map to under the tiling. This allows the agent to still achieve precision even when actions are limited.

Both of these differences together make Tile Coding not suitable for environments in which actions are limited and precision is required - a flaw not shared by the proposed method.

6 EXPERIMENTAL RESULTS

The results from using the proposed method in Section 5 are now given. Each variation of the Flag Collection environment was completed by the agent ten times. The results shown are the mean values of each attempt. The uniform tiling was performed using tiles of varying sizes depending on the variation size — but typically using tiles from size 3×3 up to 10×10. The agent also completed each variation using a 'True' hand-labelled abstraction to represent perfect knowledge, as well as an agent that uses no reward shaping — just the standard $Q(\lambda)$ algorithm.

Each algorithm used the same parameters, with the exception of the number of episodes the agent was tested on — this was tuned for each agent to show the differences in convergence times easier to visualise. The parameters used were $\alpha = 0.1$, $\lambda = 0.9$, $\gamma = 0.99$, $\omega = 20$, and $\epsilon = 0.5$ initially and linearly decaying to 0.05. These parameters were all found empirically to give strong results over a wide range of environment variations and abstractions.

In Figure 3a there is a clear trend showing that many of the uniform tilings actually can compete with hand-labelled examples, and sometimes even converge quicker. This is actually quite intuitive, as using smaller tilings over a state-space will typically yield more transitions between abstract states. This means that there are more

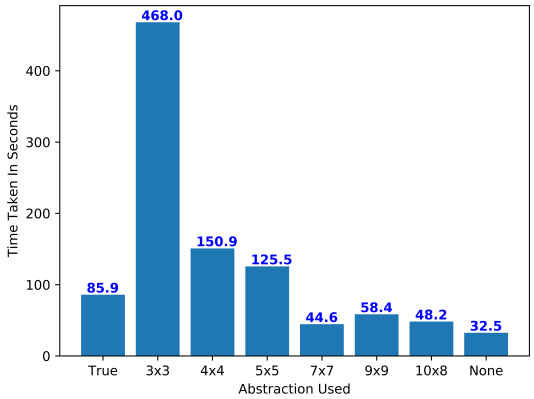


Figure 2: Time taken to solve each abstraction and simulate 10,000 episodes for the basic Flag Collection domain

states in which reward shaping is fully utilised, as $\phi(s) = \phi(s')$ if both $s, s' \in t$ for abstract state t. The extrinsic reward is then equal to $\gamma\phi(s') - \phi(s) = (\gamma - 1)\phi(s)$. In the case of the these experiments, the extrinsic reward then becomes $-0.01 \times \phi(s)$. The point here is that having more state transitions with a high extrinsic reward will give the agent more guidance.

Whilst using smaller tilings may improve the number of episodes required to converge to a near-optimal policy, solving the AMDP created becomes harder the more abstract states it has. In Figure 2 the time taken to solve each abstraction and simulate 10000 episodes is shown. This figure shows the vast increase in time required for smaller tilings. Many of the smaller tilings take much longer than the hand labelled abstraction. However, the 5x5 tiling actually takes a time comparable to this hand labelled abstraction. The 5x5 tiling and hand labelled abstraction both perform similarly. This is a very positive result due to the difference in knowledge given to each agent and abstraction. The larger tilings take much less time, and although they perform worse than the hand labelled

abstraction, they do not perform particularly badly. In fact, every tiling outperformed the standard $Q(\lambda)$ algorithm.

Throughout the rest of the results in Figure 3, this trend continues. It is worth noting that many of these variations of the flag collection problem required markedly fewer episodes in order for most of the abstractions to converge upon the optimal policy. A lot of the uniform abstractions converge quicker than the hand-labelled abstraction.

Figure 3b appears to get stuck in a local optimum and the agent's policy doesn't converge on a final flag. This is likely to do with the initial parameters, the size of the domain and also the accuracy required to achieve reward. The abstract states are also much larger in comparison to individual states — meaning that navigating to an abstract state is not as useful. It is worth noting that this largeness also applies to the hand-crafted abstraction.

7 CONCLUSION

This paper has shown that in many variations of a commonly tested RL domain, that uniform abstractions of a state space can often compete with hand crafted domain knowledge. Previous work used domain knowledge to produce a state abstraction function, an abstract transition function and abstract reward function. The results from this paper suggest that the state abstraction function and abstract transition function are not a hugely important factor for convergence of the agent, and that just grouping states which are "close by" in the state-space is enough.

7.1 Future Work

There are innumerable ways in which this idea can be extended. Some of the more promising ones are addressed briefly here.

7.1.1 Temporal Abstractions. The abstractions created for the Flag Collection domain used in this paper were of only a spatial nature. The utility of uniform abstractions should also apply temporally. By this, it is meant that instead of creating abstract states using their spatial position in the environment, the agent creates abstractions based on time. There is a remarkable similarity between abstractions in both space and time. In the Flag Collection domain (and most other MDP-based environments) a single action is assumed to take unit time. This means that a temporal abstraction of a fixed time can be viewed as a path of fixed length through the statespace of the MDP. Uniform spatial abstractions could then be constructed over these areas in the same way. What remains to be seen is how well the method of uniform abstractions maps over to an temporal domain, due to the fact that the environment dynamics and topology will no longer necessarily be limited to physical space.

7.1.2 Hierarchies of Abstraction. When using the method proposed in this paper and the state space of the environment gets even larger, there are currently two options. The first is the keep the tile dimensions the same, but this will drastically increase the time taken to solve the associated AMDP. The second option is to increase the tile size. This will resolve difficulty of solving the AMDP. However, doing this will reduce the amount of shaping given to the agent, as more states are considered to be in each tile. This will likely slow down learning.

A possible solution to this, would be to introduce a 'hierarchy' of abstraction. The principal idea here being to repeat the tiling process on the constructed AMDP. The tiling process may be repeated an arbitrary amount creating a "hierarchy" of AMDPs, with the higher-level AMDPs informing and shaping the lower levels. It is not clear if this would be worth doing, or if the advantages would diminish for each level of abstraction added. Doing this could potentially allow for solving much larger problems, without as large sacrifices as the previously stated other options.

7.1.3 Removing Flag Knowledge. One of the remaining details that is provided to the agent by a domain expert is which abstract state each flag lies within. Certainly, as the abstract states become smaller, this assumption becomes harder to justify. Removing this characteristic would be ideal from a perspective of automation. Knowledge Revision for AMDPs [4] allows a learning agent to update erroneous knowledge in the AMDP. Building on this, if the flag collection action is always considered to "collect" the appropriate flag for every state of the AMDP, then the knowledge revision technique could be employed to devalue the "collect" action in incorrect abstract states. Doing so would reduce the value of these non-existent transitions, causing the agent to ignore them. This would potentially enable the agent to create effective abstractions with essentially zero domain knowledge. What is uncertain, is how quickly the revision techniques would remove all of the erroneous knowledge, and whether this would impact the viability of this method for large scale applications.

REFERENCES

[1] Richard Bellman. 2010. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA.

[2] Kyriakos Efthymiadis, Sam Devlin, and Daniel Kudenko. 2016. Overcoming incorrect knowledge in plan-based reward shaping. *The Knowledge Engineering Review* 31, 1 (2 2016), 31–43. https://doi.org/10.1017/S026988891500017X

[3] Kyriakos Efthymiadis and Daniel Kudenko. 2014. A comparison of plan-based and abstract MDP reward shaping. *Connection Science* 26, 1 (2014), 85–99.

[4] Kyriakos Efthymiadis and Daniel Kudenko. 2015. Knowledge Revision for Reinforcement Learning with Abstract MDPs. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS '15)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 763–770. http://dl.acm.org/citation.cfm?id=2772879.2773251

[5] M. Grzes and D. Kudenko. 2008. Plan-based reward shaping for reinforcement learning. In *2008 4th International IEEE Conference Intelligent Systems*, Vol. 2. 10–22–10–29. https://doi.org/10.1109/IS.2008.4670492

[6] M. Marashi, A. Khalilian, and M. E. Shiri. 2012. Automatic reward shaping in Reinforcement Learning using graph analysis. In *2012 2nd International eConference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 111–116. https://doi.org/10.1109/ICCKE.2012.6395362

[7] Bhaskara Marthi. 2007. Automatic Shaping and Decomposition of Reward Functions. In *Proceedings of the 24th International Conference on Machine Learning (ICML '07)*. ACM, New York, NY, USA, 601–608. https://doi.org/10.1145/1273496.1273572

[8] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML '99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 278–287. http://dl.acm.org/citation.cfm?id=645528.657613

[9] Jette Randløv and Preben Alstrøm. 1998. Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 463–471. http://dl.acm.org/citation.cfm?id=645527.757766

[10] Richard S. Sutton and Andrew G. Barto. 2017. *Introduction to Reinforcement Learning* (2 ed.). MIT Press, Cambridge, MA, USA.