In *Technical Report of the 2018 AAAI Spring Symposium Series, Symposium on Data-Efficient Reinforcement Learning* DERL 2018 Stanford, CA, USA March 2018

State Abstraction Synthesis for Discrete Models of Continuous Domains

Jacob Menashe and Peter Stone
{jmenashe,pstone}@cs.utexas.edu
The University of Texas at Austin
Austin, TX USA

Abstract

Reinforcement Learning (RL) is a paradigm for enabling autonomous learning wherein rewards are used to influence an agent's action choices in various states. As the number of states and actions available to an agent increases, so it becomes increasingly difficult for the agent to quickly learn the optimal action for any given state. One approach to mitigating the detrimental effects of large state spaces is to represent collections of states together as encompassing "abstract states".

State abstraction itself leads to a host of new challenges for an agent. One such challenge is that of automatically identifying new abstractions that balance generality and specificity; the agent must identify both the similarities and the differences between states that are relevant to its goals, while ignoring unnecessary details that would otherwise hinder the agent's progress. We call this problem of identifying useful abstract states the Abstraction Synthesis Problem (ASP).

State abstractions can provide a significant benefit to model-based agents by simplifying their models. T-UCT, a hierarchical model-learning algorithm for discrete, factored domains, is one such method that leverages state abstractions to quickly learn and control an agent's environment. Such abstractions play a pivotal role in the success of T-UCT; however, T-UCT's solution to ASP requires a fully discrete state space.

In this work we develop and compare enhancements to T-UCT that relax its assumption of discreteness. We focus on solving ASP in domains with multidimensional, continuous state factors, using only the T-UCT agent's limited experience histories and minimal knowledge of the domain's structure. Finally, we present a new abstraction synthesis algorithm, RCAST, and compare this algorithm to existing approaches in the literature. We provide the algorithmic details of RCAST and its subroutines, and we show that RCAST outperforms earlier approaches to ASP by enabling T-UCT to accumulate significantly greater total reward with minimal expert configuration and processing time.

1 Introduction

The efficiency with which an AI agent learns the dynamics of a domain is heavily influenced by that agent's internal representation of the world. This is especially true for complex, multi-faceted domains that are often found in the real world. For this reason, AI researchers are often forced to wrestle with the so-called *curse of dimensionality* wherein the complexity of a domain scales exponentially with the number of variables used to describe it.

A traditional solution to this problem is through the use of hierarchical layers of abstraction; rather than painstakingly learning about every individual state possible in some domain, an agent can instead group large numbers of states together and consider only this abstract representation during the learning process. While such representations are merely helpful for simpler domains, abstractions are essentially a requirement when applying any form of machine learning to domains over real-valued variables (such as one's position in continuous space).

The T-UCT algorithm (**?**) exemplifies the success of this approach by learning decision-tree-based models from scratch in hierarchically structured domains. However, T-UCT has no internal mechanism for modeling continuous state, and while it is designed for domains with factored state representations, it performs poorly on domains whose state factors span large value spaces. When faced with value spaces of infinite cardinality, T-UCT often performs worse than chance.

In this work we augment T-UCT with mechanisms for efficiently modeling actions and state transition dynamics in continuous, factored state spaces. We call this augmented version of T-UCT *Continuous* T-UCT (CT-UCT). We design our CT-UCT augmentations in such a way that a variety of competing abstraction synthesis algorithms can be "plugged in" and evaluated on a single hierarchical, continuous learning task, toward the ultimate goal of enabling a CT-UCT agent to maximize its total accumulated extrinsic reward.

In Section 2 we describe the necessary background for T-UCT and CT-UCT as well as the past research on abstraction synthesis. The primary challenge in developing CT-UCT is that of retrofitting T-UCT's discrete model-learning infrastructure with the necessary machinery for learning abstractions over continuous space; in Section 3 we present a novel abstraction synthesis algorithm, the *Recursive Cluster-based Abstraction Synthesis Technique* (RCAST), which achieves this feat of identifying abstractions that can be consumed by CT-UCT's modeling framework.

In Section 4, we compare RCAST with alternative algorithms from existing literature by plugging these algorithms into CT-UCT and evaluating their performance on a challenging HRL task. Finally in Section 5 we conclude and discuss

future work.

2 Background and Related Work

In this paper we focus our discussion on state abstraction synthesis in the context of Reinforcement Learning (RL). Thus we begin our discussion of background literature with a brief overview of model-based RL, and then proceed to cover the related work on state abstraction.

2.1 Model-based Reinforcement Learning

Model-based reinforcement learning is a branch of reinforcement learning in which an agent uses a model to predict the effects of its actions in the environment. In effect, while classical Reinforcement Learning is concerned with learning a value function dependent upon R, model-based reinforcement learning is additionally concerned with invoking (and possibly learning) an approximation of P. Often, the agent's model of P is used as an intermediate step toward improving the value function.

In our work we consider models based on Conditional Probability Trees (CPTs), which are a form of decision tree in which each internal node "splits" based on the values of a particular state factor F. Each branch from such an internal node denotes a value (or set of values) for F. (**?**) describe how such a model can encode the dynamics of a particular RL domain and be used to predict a state s_t from its predecessor s_{t-1} and an action a_t taken in s_{t-1} for some timestep t.

(**?**) describe the VISA algorithm, which uses CPTs and $\langle s_t, a_t, s_{t+1} \rangle$ histories to learn a model of its environment from scratch. (**?**) use T-UCT (based on the UCT algorithm (**?**)) with the CPT framework of (**?**) to improve learning performance and sample efficiency in comparison with VISA, however both T-UCT and VISA assume discrete MDPs. In our work, RCAST provides the key mechanism for relaxing this assumption of discreteness by producing discrete state abstractions over continuous value spaces.

2.2 State Abstraction Synthesis

Small, finite, and factored state spaces give rise to useful and intuitively defined state abstraction mechanisms. (**?**) propose a method for state abstraction in such factored state spaces through identification of so-called "irrelevant" factors. For instance, if the state space S has factors X and Y, then an abstract state might be a particular assignment $X = x_0$ with no assignment for Y. In this case, the abstract state space S' consists of $|X|$ abstract states each encompassing $|Y|$ primitive states. (**?**) use decision tree models of the state space toward a similar end, where each branch encodes an assignment of variables to values, and omitted variables represent those that are irrelevant for a particular action model.

Rather than defining abstractions in terms of critical values, there has been ample work on defining abstractions in terms of their relevance to "macro" actions using the options framework (**?**). "Bottleneck" options are one such example where the state space is divided into regions on either side of highly-traversed intermediate states ("bottleneck" states). The initiation and termination sets of such options each designate two distinct abstract states that can be used for planning in lieu of the primitive state space (**?**; **?**; **?**). Macro actions give rise to Hierarchical Reinforcement Learning (HRL), where a single macro action may consist of many sub-actions, and may itself comprise part of a more general macro action. State abstraction is often a central component of an HRL algorithm; T-UCT is no exception, as its entire model-learning framework is concerned with identifying dependencies between abstract states.

State abstractions can be more difficult to synthesize in domains with continuous-valued state variables. Due to the negligible likelihood of visiting a single real-valued state multiple times, an agent must instead attempt to visit the neighborhoods about such values for effective planning. Planning with neighborhoods raises the challenge of determining the appropriate size and shape of such neighborhoods. Option-based state abstraction extends naturally into continuous domains, however this does not relieve the aforementioned difficulty in identifying continuous neighborhoods. Such neighborhoods can be classified using a traditional supervised learning approach (**?**), but this relies on large numbers of sample trajectories and predefined classes used to label the samples.

Iterative Half-Space (IHS) Abstraction Synthesis Many alternative approaches to abstraction synthesis rely on iteratively dividing the state space into half-spaces using hyperplanes (**?**; **?**; **?**; **?**; **?**). Such iterative half-space (IHS) approaches identify optimal hyperplanes for splitting some space one at a time until all of the splits necessary to fully describe the space's dynamics have been identified. While this technique can achieve arbitrary levels of precision, it invariably results in creating unnecessary abstract states as a side-effect of the iterative halving process. Moreover, when multiple half-spaces are required for meaningful separation of datapoints, the initial splits must be performed with limited statistical indication of their relevance. Thus the algorithm must split aggressively in anticipation of high quality abstractions many iterations in the future, and at the same time split conservatively to avoid creating abstractions that are harmful to the learning process. The overall effect is that such algorithms tend to be either sample-inefficient or inaccurate.

(**?**) tackle the state abstraction problem with unidimensional continuous factors by hierarchically splitting continuous intervals into two parts at a time, but even this approach suffers from creating unnecessary abstract states and is poorly suited to continuous factors over multiple dimensions.

Our work improves upon that of **?** by both removing the need for creating unnecessary abstract states and enabling abstraction over continuous factors of arbitrary dimensionality. Section 3 provides a more detailed description of these differences.

kd-tree Discretization k-Dimensional Trees, originally described by (**?**) provide an effective means of partitioning continuous state with discrete and succinctly specifiable bounds to arbitrary levels of precision. Since each facet of a kd-tree is a hyperplane, any kd-tree is equal to some combination of half-space bounds, and thus the argument can be made that a kd-tree partitioning can be inferred via IHS abstraction synthesis. However, such inferrences may not

always be possible within reasonable bounds on processing time or computational complexity; an algorithm which can identify kd-tree partitions may therefore outperform an IHS algorithm in time-bound domains.

The Parti-Game Algorithm (**?**) is an example of how kd-tree-based abstractions can be beneficial in representing discrete decision boundaries over continuous-valued state spaces of arbitrary dimensionality. This algorithm is designed for deterministic goal-oriented RL problems where individual leaves are mapped to decisions; however, its core idea of representing decision boundaries with kd-trees shows promise in the more open-ended abstraction synthesis problem.

(**?**) extend the application of kd-trees to more traditional RL problems with the Variable Resolution Model-Free Function Approximation Algorithm. Here **?** shows that kd-trees can be used to approximate action dynamics over continuous domains without the need for deterministic state transitions or predefined goal states.

While kd-trees are used extensively in past work for the purpose of modeling dynamics or representing decision trees, we know of no other work where kd-trees are applied to the Abstraction Synthesis Problem in the manner we describe below. In Section 3 we introduce our own solution to the abstraction synthesis problem, where we apply kd-trees to the task of partitioning continuous, multidimensional state abstractions.

3 The RCAST Algorithm

This section introduces RCAST, one of the primary contributions of this work and the key to enabling CT-UCT to scale to continuous state spaces. We will first visually depict RCAST on a hypothetical dataset in Section 3.1, and then describe an implementation of RCAST in Section 3.2.

At each timestep $t-1$ in an MDP, an RL agent chooses an action a to take in state s_t, and then experiences the resulting state s_{t+1}. Thus, an agent which keeps track of these $\langle s_t, a_t, s_{t+1} \rangle$ tuples can analyze them to predict future experiences. A key feature of an RL model is the ability to predict s_{t+1} from s_t and a, and in a factored domain an agent may wish to specifically predict the value of some output factor F_o in s_{t+1} given s_t and the operative action a. If F_o is causally related to some other factor F_i, then the value of F_i in s may be of particular relevance to predicting F_o.

The CPT framework used by VISA (**?**) and T-UCT (**?**) enables such factor-specific predictions. A CPT allows an agent to predict the value of F_o *given* the action a and the value of F_i, but both of these algorithms assume that F_i and F_o take on discrete values, and that an agent can keep track of *all possible values* of F_i when predicting how a will alter F_o. CT-UCT relaxes this assumption and allows an agent to discretely model F_i and F_o even when they take on continuous and multidimensional values by invoking an abstraction synthesizer, namely RCAST; RCAST's role is thus to identify useful abstractions over F_i's value space, so that they may be used for branching decision trees in the CPT framework of (**?**).

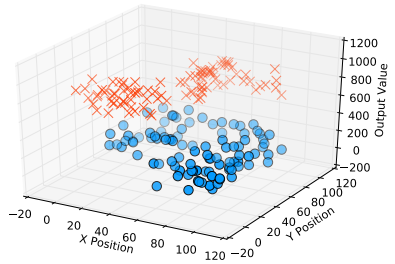

Figure 3.1: A hypothetical dataset D consisting of inputs on the xy plane and output on the z axis.

3.1 Visual Example

Before we describe the algorithmic details of RCAST we will begin by visually depicting RCAST using a hypothetical experience history H of $\langle s_t, a_t, s_{t+1} \rangle$ tuples for an RCAST agent in some domain. Assume that we are interested in understanding whether some state factor F_o depends on another F_i. For ease of visualization let us assume that $\dim(F_o) = 1$ and $\dim(F_i) = 2$.

Before analyzing the interaction between these two factors we first project H into an $\mathbb{R}^n$ subspace where $n = \dim(F_i) + \dim(F_o) = 3$. This projection $P : (S \times A \times S) \rightarrow \mathbb{R}^3$ maps $\langle s_t, a_t, s_{t+1} \rangle$ to (x, y, z) where (x, y) is the value of factor F_i in state s_{t-1} and z is the value of factor F_o in state s_t. We denote the image $P(H) = \{(x, y, z)\}$ as D.

In Figure 3.1 we see a scatter plot representation of D. Identifying a dependence relationship from F_i to F_o is therefore similar to the task of predicting z from (x, y). The shapes and colors used in Figure 3.1, as well as all figures in Section 3.1, are not available to the algorithm and are shown strictly for ease of visualization.

Figures 3.3 and 3.2 show the same dataset D restricted to F_i and F_o, respectively. From Figure 3.2 it is clear that two distinct classes of data exist; however, our goal is not to simply identify these classes, but also to use them for classifying tuples in F_i. Thus we wish to partition the plot in Figure 3.3 such that its projection into F_o's value space also partitions the data in Figure 3.2 according to the two obvious classes.

RCAST creates this partitioning by first clustering datapoints in the full $F_i + F_o$ value space (Figure 3.1) and then using these clusters to create a labeled kd-tree in the F_i value space (Figure 3.3). The labeled tree describes a partition of the F_i value space, enabling an agent to map from points in F_i to clusters in F_o.

Figure 3.4 shows the kd-tree T generated for D. In most areas the tree is only one layer deep, however the variety of points found in some regions of the value space necessitate secondary levels of refinement. Here we use a discretization factor of $\delta = 3$, resulting in $3^{\dim(F_i)} = 3^2 = 9$ subdivisions at each level, however we note that δ is configurable in the general case.

The fully labeled T in Figure 3.5 is a classifier that maps

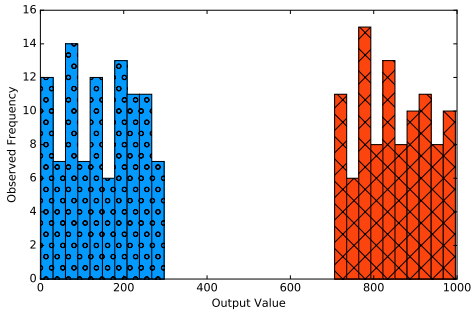

Figure 3.2: A histogram of the output (z) values in the dataset D from Figure 3.1.

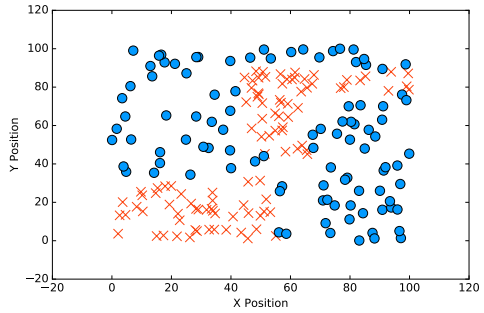

Figure 3.3: A view of D from Figure 3.1 projected onto the input (xy) plane.

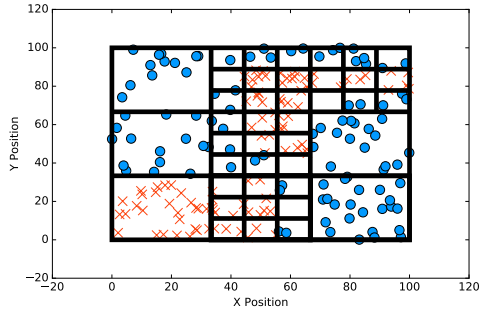

Figure 3.4: A kd-tree T which partitions the dataset D based on its classes of output (z) values.

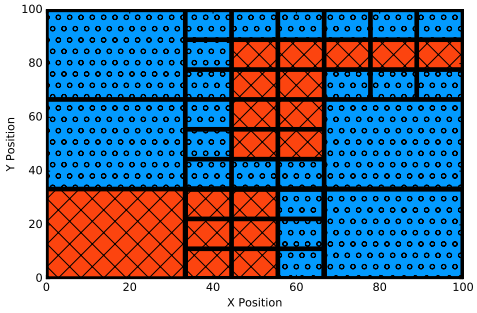

Figure 3.5: A kd-tree T with filled regions visually depicting the labels applied to its leaves.

F_i-value coordinates to labels. The union of the regions encompassed by the leaves of T for some label l can therefore be considered an abstract state over F_i which is relevant to predicting F_o. We can divide T according to these labels, creating a set of abstract meta-states which can then be integrated into a discrete model. Similar to the way in which singleton values can partition a tabular space for a discrete CPT model, these abstractions partition a continuous space for the same purpose (see (**?**)). In this way we are able to discretely model F_o's dependence upon F_i even though these factors describe multidimensional continuous values.

3.2 Algorithm Description

RCAST's purpose is to analyze observed dynamics in a given environment and identify key areas of the environment that exhibit similar dynamics. The areas identified by RCAST then inform model refinements which allow an agent to use its experiences to knowledgeably plan its actions.

Algorithm 3.1a provides pseudo-code to describe RCAST. Line 1 defines the basic inputs to the algorithm including an *input factor* F_i, an *output factor* F_o, a dataset D, and an orthotope Q describing the bounds of F_i. In calling this function we assume that changes in F_o depend on F_i when F_i's value falls within Q. RCAST analyzes D to identify the specifics of this relationship, returning a set of subspaces of Q which serve as abstractions over the value space of F_i.

In Line 2 we see that the use of the $\sqcap$ operator applied to D and Q. This operator restricts the dataset to those datapoints whose predecessor states' value assignments for F_i fall within the bounds of Q. Intuitively, this means that each state-action-state sequence "started" in Q. Line 3 then clusters this subset of points using Expectation-Maximization Clustering, which produces as many clusters as necessary to maximize the BIC score of successive Expectation Maximization iterations. We use Expectation Maximization over Gaussian models (implemented with OpenCV (**?**)).

Line 4 projects the clusters' datapoints into the value space Q of F_i so that Q can be partitioned in accordance with these clusters. Line 5 then hierarchically partitions Q according to C' using Algorithm 3.1b as the partitioning subroutine. Algorithm 3.1b produces a kd-tree with leaves labeled according to the clusters they encompass. In Line 6 this tree is

```
1: function RCAST(F_i, F_o, D, Q)
2:     D' ← D ⊓ Q
3:     C ← EM(D')
4:     C' ← F_i(C)
5:     T ← H-Part(Q, C')
6:     A ← SplitLabels(T)
7:     return A
8: end function
```

(a) An implementation of RCAST.

```
1: function H-PART(Q, C)
2:     T ← an empty kd-tree with label ∅
3:     C' ← {c ⊓ Q | c ∈ C, c ⊓ Q ≠ ∅}
4:     if |C'| = 1 then
5:         T.label ← c.label where c ∈ C'
6:     else if diameter(Q) ≤ 1 then
7:         T.label ←"multi"
8:     else
9:         d ← dimensionality of Q
10:        Q ← Partition Q into δ^d equal parts
11:        for q ∈ Q do
12:            T_q ← H-Part(q, C)
13:            T.children ← T.children ∪ {T_q}
14:        end for
15:    end if
16:    return T
17: end function
```

(b) H-Part: A hierarchical partitioning algorithm.

Algorithm 3.1: RCAST and its primary subroutine H-Part

partitioned according to its labeling, with each subtree containing all the leaves for a particular label. Since each of these leaves defines an orthotopic subspace of Q, each subtree is associated with a distinct subspace of Q, namely the union of its leaves' orthotopes. Each such union is an abstraction over the value space of F_i.

Algorithm 3.1b describes a general method for generating a kd-tree from a set of clusters C and an encompassing value space Q. In Line 3 the algorithm restricts its cluster set C to only those clusters with datapoints in Q. Next, H-Parttakes one of three actions. If D contains only one class of datapoint, the current subtree becomes a single leaf with its label taken from that class (Line 5). If D contains multiple classes of datapoints but Q is of minimal diameter[1] then T is designated as having the special label "multi" (Line 7). Otherwise, if the diameter of Q is large enough, the algoritm subdivides and recurses (Line 12). Ultimately the algorithm labels every leaf with either a class label or "multi". We use a static minimum diameter of 1 in this work.

In the context of CPTs, the partitions produced by Algorithm 3.1a are used to split leaves of the CPT. (**?**) show that a CPT leaf can be split along some input factor F_i' by creating one child leaf per value of that factor. In a sense, the discrete value space of F_i' is partitioned into singleton subspaces where each value of F_i comprises one subspace.

[1]Here the *diameter* of an orthotope O is defined as $\sup\{d(x,y)|x,y \in O\}$ where d is Euclidean distance.

In the context of a continuous and multidimensional F_i, singleton partitioning is not possible, so instead Algorithm 3.1b produces the aforementioned unions of orthotopes to represent arbitrary subspaces of F_i's value space, with each union corresponding to a single leaf added to the CPT.

Thus, we can refine a CPT model over a continuous, multidimensional F_i by partitioning its value space and then creating one new CPT leaf per label. In Section 4 we evaluate this approach empirically and compare against alternative abstraction synthesis methods.

4 Experiments

This section describes a set of experiments performed to evaluate the effectiveness of RCAST in comparison to both IHS and a deep regression network (DRN). While it would not be possible to evaluate RCAST against all the alternative approaches to abstraction synthesis described in Section 2, IHS and DRN are representative of the state of the art and serve as reasonable proxies for most other methods. Implementation details cannot be included due to space constraints.

4.1 The Continuous Taxi Domain (C-Taxi)

In our work we are interested in evaluating T-UCT and its derivatives on a HRL task over a factored, continuous state space. The Taxi domain (**?**) is a common choice for evaluating HRL algorithms, but lacks the property of having a continuous-valued state space. We therefore employ a modified version of Taxi in our work, which we refer to as Continuous Taxi (C-Taxi). Rather than the traditional 5-by-5 discrete grid, we use a 100-by-100 continuous grid containing multiple rectangular regions in which anywhere from 1 to 1000 passengers may be picked up by the Taxi agent. More formally, the state space of this domain is the set of 3-tuples $\langle x, y, k\rangle$ where x, y is the position of the agent in the grid and k is the number of passengers currently held by the agent. There are 6 actions including actions for movement in the four cardinal directions N, S, E, W as well as the pickup and dropoff actions P, D. When a movement action is executed the agent is transported a uniformly random distance between 5 and 10 units in the appropriate direction. When P is executed in a pickup region, a uniformly random number of between 500 and 1000 passengers is picked up. Dropoffs always unload all passengers.

The "goal" of the C-Taxi domain is to drop off passengers quickly; thus, the agent receives a reward of -1 for any movement actions, and a reward of 0 for pickups. When the agent executes the dropoff action D it receives a reward equal to the number of passengers that were dropped off.

A perfect abstraction over this domain (with respect to the pickup action and the vehicle's passenger count) would partition the grid into two distinct regions: one non-contiguous region matching the union of the pickup regions, and a second region perfectly complementing the first. When used as the basis for splitting a decision tree, such abstractions would allow an agent to reason with respect to perfect knowledge of where pickups occur, and where they don't.

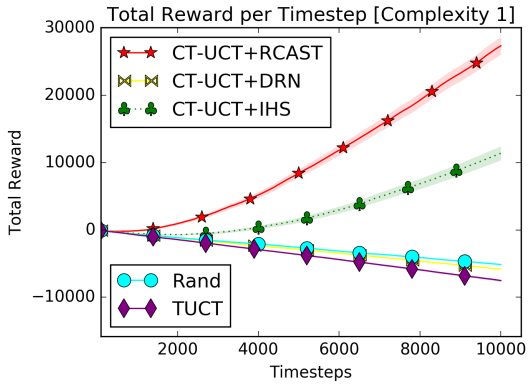

Figure 4.1: A comparison of the CT-UCT+IHS, CT-UCT+DRN, T-UCT, Random, and CT-UCT+RCAST algorithms over many C-Taxi domain instances.

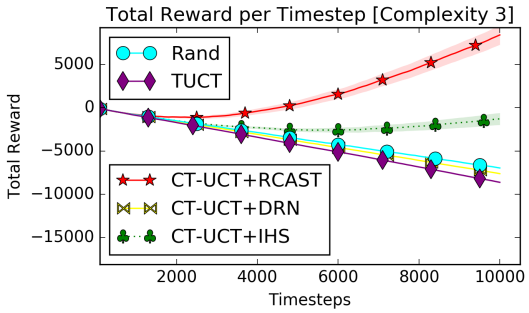

Figure 4.2: A comparison of the CT-UCT+IHS, CT-UCT+DRN, T-UCT, Random, and CT-UCT+RCAST algorithms over many C-Taxi domain instances.

4.2 Experiment Setup

In each experiment we evaluate a selection of HRL algorithms on the task of accumulating extrinsic reward in the C-Taxi domain. Since T-UCT is a model-learning algorithm that relies on extrinsic reward, we modify its internal exploration selection mechanism such that it decides between exploitation-oriented and exploration-oriented targets. During its target selection phase (see Section 3.2 of (**?**)), the T-UCT algorithm selects a target context based on its expectation of earned intrinsic reward. For the purpose of evaluating T-UCT on its ability to earn extrinsic reward, we modify this process such that targets are randomly selected based on expected *extrinsic* reward. This allows T-UCT and its derivatives to exploit their learned models with respect to the domain's reward signal. This modification is applied to all such algorithms in this work.

We present empirical results on total reward earned in the C-Taxi domain for the following algorithms: Random (uniformly random action selection), T-UCT[2], CT-UCT+IHS, CT-UCT+DRN, and CT-UCT+RCAST. The following figures show the average results over different levels of complexity. A domain of complexity n contains n pickup regions and n dropoff regions. We provide results for complexities 1, 2, and 3 below. Within each complexity level, we take the average of the results obtained over 30 different C-Taxi instances, each having differing random placements and sizes of pickup and dropoff regions. We then perform 10 evaluations per agent on each such instance, and record each agent's total processing time and accumulated reward every 100 timesteps.

4.3 Results

Experimental results are shown in Figures 4.1 and **??**. In both figures, "Complexity n" indicates that n pickup regions and n dropoff regions are generated for every instance of the C-Taxi domain. In both result sets, values are averaged over 30 domain instances with 10 trials per algorithm per domain instance. Shaded regions represent standard error.

In Figure 4.1 we see that RCAST significantly outperforms every other algorithm ($p \ll .001$). Figure **??** indicates that as complexity increases the performance gap only widens. These results show that RCAST is able to efficiently handle complex abstraction synthesis problems and allow for efficient exploration and exploitation in these domains.

[2]We note that T-UCT cannot model continuous state and so we use a simple tile coding over $3^{\dim(F)}$ uniform tiles that evenly divide the value space of each factor F.

5 Conclusion and Future Work

In this work we have described RCAST, a new method for synthesizing abstract states based on observed data. We have used RCAST as the core abstraction synthesis mechanism of CT-UCT, and thereby enabled T-UCT to produce state abstractions for continuous space and effectively incorporate these abstractions into its discrete decision tree model. Moreover, we have shown that RCAST is superior to alternative approaches to abstraction synthesis with respect to total accumulated extrinsic reward, and is competitive to alternatives with respect to time and configuration complexity.

Another interesting focus for future work lies in modifying the clustering subroutine which RCAST depends upon. There exists a vast array of EM alternatives the present literature which may be better suited to the problem of abstraction synthesis. For instance, EM is reliant upon Gaussian likelihood comparisons and is thus biased toward ellipsoid clusters; it may instead be advantageous to employ a hierarchical clustering algorithm that is better equipped to generate irregularly shaped clusters with a focus on contiguity.

Acknowledgements

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-1637736, IIS-1651089, IIS-1724157), Intel, Raytheon, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

Bradski, G. 2000. The opencv library. *Dr. Dobb's Journal of Software Tools*.

Fayyad, U., and Irani, K. 1993. Multi-interval discretization of continuous-valued attributes for classification learning.

Friedman, J. H.; Bentley, J. L.; and Finkel, R. A. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)* 3(3):209–226.

Hengst, B. 2002. Discovering hierarchy in reinforcement learning with HEXQ. In *ICML*, volume 2, 243–250.

Jong, N. K., and Stone, P. 2005. State abstraction discovery from irrelevant state variables. In *IJCAI*, volume 8, 752–757.

Jonsson, A., and Barto, A. G. 2001. Automated state abstraction for options using the u-tree algorithm. *Advances in neural information processing systems* 1054–1060.

Jonsson, A., and Barto, A. 2005. A causal approach to hierarchical decomposition of factored MDPs. In *Proceedings of the 22nd international conference on Machine learning*, 401–408. ACM.

Jonsson, A., and Barto, A. 2006. Causal graph based decomposition of factored MDPs. *Journal of Machine Learning Research* 7:2259–2301.

Jonsson, A., and Barto, A. 2007. Active learning of dynamic Bayesian networks in Markov decision processes. In *Abstraction, Reformulation, and Approximation*. Springer. 273–284.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*. Springer. 282–293.

Kohavi, R. 1996. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, volume 96, 202–207. Citeseer.

Konidaris, G., and Barto, A. G. 2009. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in neural information processing systems*, 1015–1023.

Liu, B.; Xia, Y.; and Yu, P. S. 2000. Clustering through decision tree construction. In *Proceedings of the ninth international conference on Information and knowledge management*, 20–29. ACM.

McGovern, A., and Barto, A. G. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density.

Menache, I.; Mannor, S.; and Shimkin, N. 2002. Q-cut: dynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning*, 295–306. Springer.

Menashe, J., and Stone, P. 2015. Monte Carlo Hierarchical Model Learning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 771–779.

Moore, A. W. 1994. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In *Advances in neural information processing systems*, 711–718.

Quinlan, J. R., et al. 1992. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, 343–348. Singapore.

Reynolds, S. I. 2000. Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 783–790. Morgan Kaufmann Publishers Inc.

Stolle, M., and Precup, D. 2002. Learning options in reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, 212–223. Springer.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1):181–211.

Vigorito, C. M., and Barto, A. G. 2010. Intrinsically motivated hierarchical skill learning in structured environments. *IEEE Transactions on Autonomous Mental Development* 2(2):132–143.