Automatic Task Decomposition and State Abstraction from Demonstration

Luis C. Cobo
College of Engineering
Georgia Tech
Atlanta, GA 30332
luisca@gatech.edu

Charles L. Isbell Jr.
College of Computing
Georgia Tech
Atlanta, GA 30332
isbell@cc.gatech.edu

Andrea L. Thomaz
College of Computing
Georgia Tech
Atlanta, GA 30332
athomaz@cc.gatech.edu

ABSTRACT

Both Learning from Demonstration (LfD) and Reinforcement Learning (RL) are popular approaches for building decision-making agents. LfD applies supervised learning to a set of human demonstrations to infer and imitate the human policy, while RL uses only a reward signal and exploration to find an optimal policy. For complex tasks both of these techniques may be ineffective. LfD may require many more demonstrations than it is feasible to obtain, and RL can take an inadmissible amount of time to converge.

We present Automatic Decomposition and Abstraction from demonstration (ADA), an algorithm that uses mutual information measures over a set of human demonstrations to decompose a sequential decision process into several subtasks, finding state abstractions for each one of these subtasks. ADA then projects the human demonstrations into the abstracted state space to build a policy. This policy can later be improved using RL algorithms to surpass the performance of the human teacher. We find empirically that ADA can find satisficing policies for problems that are too complex to be solved with traditional LfD and RL algorithms. In particular, we show that we can use mutual information across state features to leverage human demonstrations to reduce the effects of the curse of dimensionality by finding subtasks and abstractions in sequential decision processes.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Human Factors.

Keywords

Reinforcement learning, learning from demonstration, task decomposition, state abstraction.

1. INTRODUCTION

As it is impractical to implement manually every possible skill an agent might need to flourish in a human environment, our research aims to enable autonomous agents to learn new skills from *non-expert* humans without intervention from an engineer or programer. Agents such as non-player characters in video games or robotic domestic assistants must be able to adapt their behavior and learn new skills from everyday people.

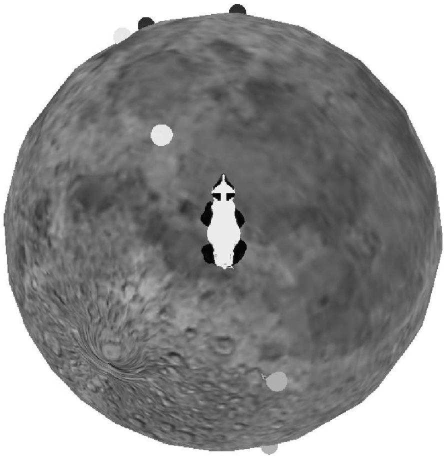

Figure 1: RainbowPanda domain. The agent must pick all balls, aiming at those that match the agent color at each moment. With 12 continuous state features, traditional RL does not converge in a reasonable amount of time. ADA finds a satisficing policy quickly by decomposing the problem into simpler subtasks.

Our approach derives a great deal from Learning from Demonstration (LfD) [3]. In the typical LfD scenario, an agent learns a mapping from states to actions using supervised learning techniques on a set of human demonstrations of a task. Such approaches assume that human demonstrations are positive examples of a near-optimal policy.

While LfD has proven successful [2], human demonstrations are expensive to obtain and, depending on the size of the state-action space, may require an impractical number of demonstrations. Additionally, small errors in imitating the human policy can compound and lead the agent to an unknown region of the state space. This large difference between training and testing state distributions can significantly impact performance.

In order to sidestep these shortcomings, we look for other kinds of information that human demonstrations can communicate to a learning agent. Instead of direct policy in-

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

formation, we use human demonstrations to learn a state abstraction and task decomposition to improve learning.

1.1 State Abstraction from Demonstration

We build on recent work in Abstraction from Demonstration [4] (AfD). In AfD, an agent uses human demonstrations on a task to figure which features of the state space are relevant for the task by measuring mutual information between each feature and the actions taken by the human teacher. Once the relevant features are identified, the agent builds an abstract state space in which the human policy can be expressed compactly, and then applies Reinforcement Learning (RL) algorithms to learn a policy in that abstract state space. If the original state space contains policy-invariant [9] features that can be ignored, this can lead to exponential speedups in skill learning, compared with traditional, raw-state space RL.

1.2 Task Decomposition from Abstraction

It is often the case that multipurpose agents have a high number of input signals of which only a subset are relevant for any specific task; however, using AfD does not help if all state features are relevant for the skill to be learned.

Our key insight is that there are often tasks where the entire state space is relevant for some part of the task, but the task can decomposed in subtasks so that for any given subtask there does exist an abstraction in which the policy can be expressed. For example, when we drive a car, we focus our attention almost completely on the car keys at the start and end of a drive, but completely ignore them for the rest of the trip. Our brain can receive simultaneously up to 11 million pieces of information, but it is estimated that at any given moment a person can be consciously aware of at most 40 of these [14].

Our goal is to infer this attentional focus, the particular task decomposition and state abstraction that the human demonstrator is using during their demonstration. We define a subtask as a region of the state space where only a subset of features is relevant, this subset being different from those of other subtasks. Thus we want a decomposition that maximizes our ability to apply AfD in each part.

1.3 Automatic Decomposition and Abstraction

In this paper we introduce *Automatic Task Decomposition and State Abstraction from Demonstration* (or Automatic Decomposition and Abstraction (ADA)), which uses human demonstrations to both decompose a skill into its subtasks and find independent state abstractions for each subtask. ADA can build more powerful abstractions than AfD, finding compact state space representations for more complex skills in which all state features are relevant at some point in time.

To determine which features are relevant to a particular subtask, we measure the mutual information between each feature of the state and the action taken by a human in a set of demonstrations. Once the state space is decomposed in different subtasks, the agent can learn and represent a compact policy by focusing only on the features that are relevant at each moment.

Fig.1 shows a simple example in our experimental domain. In this domain, an agent represented as a panda bear moves in an spherical surface. The agent can move forward and backwards and turn left and right. The overall task of the agent is to pick up all the balls, but at each moment it can only pick up balls of a specific color. With six balls, there are 12 continuous variables (relative angle and distance of each ball) and 1 discrete variable, the color the agent is currently allowed to pick up. In this 13-dimensional state space, traditional tabular RL takes an unreasonable amount of time to converge. Further, the complexity of the policy grows exponentially with the number of balls. As we shall see, with ADA, we can automatically decompose this problem into a set of subtasks, one per color, with each one needing to pay attention only to the closest ball of the target color. These 2-dimensional policies are easy to obtain, and the complexity of the global policy grows linearly with the number of balls.

After further situating our work in the next section, we describe in detail the ADA and ADA+RL algorithms and show that they can obtain good policies in problems where traditional RL and LfD algorithms offer poor performance.

2. RELATED WORK

Our work is at the intersection of different research lines, namely Learning from Demonstration, task decomposition, and state abstraction for RL.

LfD is a broad area of research, and several works explore how to combine demonstrations with traditional RL methods. Among these, using demonstrations or feedback to guide exploration [17, 12] or to learn a reward function [1] are complementary and could be combined with the method we propose. Other previous work uses demonstrations to extract task decompositions, like our method, but require a dynamic Bayesian network representation of the transitions and rewards models [16, 19], while our approach is model free.

There are also many approaches to task decomposition, but they usually require the user or the designer to explicitly specify the task structure [6, 10]. Others rely on heuristics that are adequate only for a very specific class of domains [8, 5, 18]. ADA is automatic and more general than these methods.

Regarding state abstraction in RL, prior work has used L_1 [13] and L_2 regularization [7], as well as selection from features that are based on Bellman error analysis [11]. While these approaches select features to represent a near-optimal value function, our work focuses on representing compactly a satisficing human policy, which is likely to be simpler and easier to learn than the optimal one. In the hierarchy for MDP state abstractions [15], ADA abstractions are in between a^*-irrelevance and π^*-irrelevance.

3. AUTOMATIC DECOMPOSITION AND ABSTRACTION

3.1 Preliminaries

We focus on sequential decision problems that can be expressed as Markov Decision Processes:

$$M = (S, A, P^a_{ss'}, R^a_s, \gamma),$$

where S is a finite state space, A a finite set of actions, $P^a_{ss'} = Pr(s'|s,a)$ is the transition model, $R^a_s = r(s,a)$ the reward function and $0 \leq \gamma \leq 1$ the discount factor. $F = \{F_1, \ldots, F_n\}$ is the set of features of the state space, so that

$S = \{F_1 \times \cdots \times F_n\}$ and a state $s \in S$ is an n-tuple of features $s = (f_1, f_2, \ldots, f_n)$.

Solving an MDP means finding a policy $\pi : S \to A$ mapping states to actions. The value or sum of discount rewards of taking action a in state s and then following a policy π is

$$Q^\pi(s, a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} Q^\pi(s', \pi(s')).$$

Most RL algorithms look for an optimal policy, *i.e.,* the policy that maximizes the sum of discounted reward,

$$\pi^*(s) = \arg\max_{a \in A} R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} Q^{\pi^*}(s', \pi^*(s'));$$

however, ADA will aim to find a *satisficing policy, i.e.,* a policy that is comparable in performance to that of the human teachers.

We utilize usual notation for set operations, including $|\cdot|$ for cardinality. We use $\|\cdot\|$ for the L_2 norm of a vector.

Human demonstrations are defined as a set of episodes, each one comprising a list of state action pairs

$$H = \{\{(s_1, a_1), (s_2, a_2), \ldots\}, \ldots\}, s_i \in S, a_i \in A.$$

Mutual information is a measure of the amount of *entropy* in one random variable that can be explained by the value of a different random variable,

$$I(X; Y) = H(X) - H(X|Y) = H(X) + H(Y) - H(X, Y),$$

and can be computed as

$$I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left(\frac{p(x, y)}{p_x(x) p_y(y)} \right), \qquad (1)$$

where $p(x, y)$ is the joint *probability density function* (pdf) of random variables X and Y and $p_x(x)$ and $p_y(x)$ the respective marginal pdfs of each random variable. Thus, we define:

$$\vec{mi}_E = (I(F_1; A), \ldots, I(F_n; A))$$

as a vector whose elements are the mutual information between each feature of the state space and the action taken by the human teacher, according to the samples of H in the region $E \subset S$. To compute $\vec{mi}_E$, we estimate the appropriate joint and marginal pdfs using the samples of H that fall in region E, and use Equation 1.

3.2 Overview

Given an MDP M and a set of human demonstrations H for a skill to be learned, ADA finds a policy in three conceptual steps and an optional fourth step:

1. **Problem decomposition.** Using H, partition the state space S in different subtasks $T = \{t_1, t_2, \ldots\}$, $\cup T = S$, $t_i \cap t_j = \emptyset$ if $i \neq j$.

2. **Subtask state abstraction.** Using H, determine, for each subtask $t_i \in T$ the relevant features $\hat{F}_i = \{F_{i_1}, F_{i_2}, \ldots\}, F_{i_j} \in F$. and build a projection from the original state space S to the abstract state space $\phi(s) = \{i, f_{i_1,s}, f_{i_2,s}, \ldots\} \in \hat{S}, s \in t_i$.

3. **Policy construction.** Build a stochastic policy $\pi(\hat{s})$.

4. **Policy improvement.** Use Reinforcement Learning to improve over the policy found in the previous step. We refer to our algorithm as ADA+RL when it includes this step.

For clarity, we list the first two steps as if they are sequential; however, they are interwoven and concurrent. The decomposition of the state space depends on the quality of the abstractions that can be found on different subspaces.

3.3 Problem decomposition

Definition 1. A set of **subtasks** $T = \{t_1, t_2, \ldots\}$ of an MDP M are a set of regions of the state space S such that:

- The set of all subtasks T form a partition of the original state space S, *i.e.,* $\cup T = S$ and $t_i \cap t_j = \emptyset$ if $i \neq j$.

- A subtask t_i is identified by having a local satisficing policy π_i that depends only on a subset of the available features. This subset is different from neighboring subtasks.

- The global policy $\pi(\hat{s})$, the combination of the policies of each subtask, is also satisficing.

While this definition is not the typical one for subtasks in a sequential decision problem, it turns out to be a useful one, particularly if we focus on human-like activities. For example, cooking an elaborate recipe requires multiple steps, and each of these steps will involve different ingredients and utensils. It is possible that two conceptually different subtasks may depend on the same features, but in our framework, and arguably in general, the computational benefits of separating them are not significant.

With ADA, we can identify these subtasks given a set of human demonstrations H with two requirements:

- There must be a sufficient number of samples min_{ss} from each subtask in the set of demonstrations H.

- The class of possible boundaries between subtasks $B = \{b_1, b_2, , \ldots\}, b_i \subset S$ must be defined. Each boundary divides the state space in two, b_i and $S - b_i$. ADA will be able to find subtasks that can be expressed as combinations of these boundaries.

The necessary number of samples is determined in the first step of the ADA algorithm. This minimum sample size is needed due to the metric we use to infer feature relevance. Mutual information is sensitive to the *limited sampling bias*, and will be overestimated if the number of samples considered is too low.

The decomposition algorithm is described in Algorithm 1. At each iteration of the while loop, we consider a subspace E, with $E = S$ in the first iteration. We then consider all valid boundaries. If there are none, then E itself is a subtask. If there are valid boundaries, we score them and choose the one with the highest score. We then split E according to the boundary, and add the two new subspaces to the list of state spaces to be evaluated, to be further decomposed if necessary.

The boundaries B can have any form that is useful for the domain. In our experiments we consider thresholds on features, *i.e.,* axis-aligned surfaces. Using these boundaries,

Algorithm 1 ADA problem decomposition.

```
Require: MDP $M = \left(S, A, P^a_{ss'}, R^a_s, \gamma\right)$, $S = \{F_1 \times \ldots \times F_n\}$,
  human demonstrations $H = \{\{(s_1, a_1), (s_2, a_2), \ldots\}, \ldots\}, s \in S, a \in A$, boundaries $B = \{b_1, b_2, \ldots\}, b_i \in S$, $\epsilon$.
  $min_{ss} \leftarrow min\_sample\_size(H, \epsilon)$
  $T \leftarrow \{\}$
  $\mathbb{S} \leftarrow \{S\}$
  while $\mathbb{S} \neq \emptyset$ do
    {pop removes the element from $\mathbb{S}$}
    $E \leftarrow \mathbb{S}.pop()$
    $B_E \leftarrow \{b \in B, valid\_split(b, E, min_{ss})\}$
    if $B_E = \emptyset$ then
      $T.push(E)$
    else
      $b_{best} \leftarrow \arg\max_{b \in B_E}(boundary\_score(b, E)))$
      $\mathbb{S}.push(b_{best} \cap E)$
      $\mathbb{S}.push((S - b_{best}) \cap E)$
    end if
  end while
  Return $T$
```

Algorithm 1 is just building a decision tree with a special split scoring function and stopping criteria.

In the next subsections we discuss the details of the split scoring function, the discriminator of valid boundaries, and the estimator of the minimum number of samples necessary. These contain the most interesting insights of ADA.

3.3.1 Boundary discriminator

Given a subspace $E \subset S$, m_{ss} and H, we consider a boundary $b \subset S$ to be valid if it meets three conditions:

1. There are enough samples in the set of human demonstration H to ensure we can measure mutual information with accuracy on both sides of the boundary, i.e.

$$|\{\{s, a\} \in H, s \in b \cap E\}| > min_{ss},$$
$$|\{\{s, a\} \in H, s \in (S - b) \cap E\}| > min_{ss}.$$

2. At least in one side of the boundary, either $b \cap E$ or $(S-b) \cap E$, it is possible to find a state abstraction, *i.e.,* some features are policy-invariant and can be ignored. We detail how we find these features in Section 3.4.

3. The state abstraction at both sides of the boundary is not the same.

This boundary discriminator works as the stopping criteria of the algorithm. When there are no more valid boundaries to be found, the decomposition step finishes.

3.3.2 Boundary scoring

The boundary scoring function determines the quality of b as a boundary between different subtasks within a region $E \in S$.

$$boundary\_score(b, E) = \left\| \frac{\vec{mi}_{b \cap E}}{\|\vec{mi}_{b \cap E}\|} - \frac{\vec{mi}_{(S-b) \cap E}}{\left\|\vec{mi}_{(S-b) \cap E}\right\|} \right\|. \qquad (2)$$

The score is thus the euclidean distance between the normalized mutual information vectors at both sides of the boundary.

We are therefore measuring the difference between the relative importance of each feature at both sides of the boundary. As we want to find subtasks that rely on different features, we choose the boundary that maximizes this difference (see Algorithm 1).

3.3.3 Minimum samples

Due to the *limited sampling bias*, mutual information is overestimated if it is measured in an insufficient number of samples. The minimum samples in ADA, given a set of demonstrations H and parameter $\epsilon \approx 0.1$ is

$$m_{ss} = \arg\min_n average\left(\frac{\vec{mi}_S - \vec{mi}_{S,n}}{\vec{mi}_S}\right) > \epsilon, \qquad (3)$$

where $mi_{S,n}$ is the mutual information vector on the original state space S, taking only a subset of n randomly chosen samples from all the samples in H. The subtraction and division are element-wise and the average function takes the average of the values of the resulting vector. Because of the variability of mutual information, it is necessary to evaluate Equation 3 several times for each possible n, each time with a different and independently chosen set of samples. Because of the limited sampling bias, the difference between $\vec{mi}_S$ and $\vec{mi}_{S,n}$ will grow as n decreases, and a binary search can be used to find m_{ss} efficiently.

3.4 Subtask state abstraction

Given a region of the state space $E \subset S$, we consider $\vec{mi}_E$ in order to estimate policy-irrelevant features. Even if a feature is completely irrelevant for the policy in a region of the space, its mutual information with the action will not be zero due to the limited sampling bias. Therefore, in ADA we group the values of $\vec{mi}_E$ in two clusters separated by the largest gap among the sorted values of the vector. If the value difference between any two features in different clusters is larger than the distance within a cluster, we consider we found a good abstraction that discards the features in the lower value cluster.

Note that this step occurs concurrently with the previous one, since the decomposition step needs to know in which regions of the state space there are good abstractions. Once these steps complete, we can build the projection function from the original function state space S to the abstract state space $\phi(s) = \{i, f_{i_1}, f_{i_2}, \ldots\} \in \hat{S}, s \in t_i$.

3.5 Policy construction

Once the task decomposition and state abstraction are completed and we have the projection function $\phi(s)$, we use the demonstrations H to build a stochastic policy that satisfies

$$P(\pi(\hat{s}) = a_i) = \frac{\left|\{\{s, a_i\} \in H, \phi(s) = \hat{s^*}\}\right|}{\left|\{\{s, a\} \in H, \phi(s) = \hat{s^*}, a \in A\}\right|}, \qquad (4)$$

where $\hat{s^*}$ equals to $\hat{s}$ if $|\{\{s, a\} \in H, \phi(s) = \hat{s}, a \in A\}\}| > 0$. Otherwise, $\hat{s^*}$ equals to the nearest neighbor of $\hat{s}$ for which the denominator in Eq. 4 is not zero.

To compute the policy we project the state of each sample of H into the abstracted space and make a normalized histogram of each action. This concludes the basic ADA algorithm.

3.6 Policy improvement

ADA+RL adds another step, policy improvement, in which we use Reinforcement Learning techniques to find the optimal policy that can be represented in the abstract state space $\hat{S}$. Unlike traditional LfD techniques, ADA was designed so that the resulting policy can be easily improved given additional experience. In this way, we can obtain a better policy than that of the human teacher.

Given the kind of abstraction that ADA performs, bootstrapping methods such as Sarsa or Q-learning are not guaranteed to converge in the abstract state space[15]. As such, we can use either Monte Carlo methods or direct policy search. In Section 4 we choose to use policy search because it is fast and performs well given the compact state space that ADA generates.

4. EXPERIMENTAL RESULTS

4.1 Domains

To test the ADA and ADA+RL algorithms, we used two different domains, PandaSequential and RainbowPanda. We implemented a 3D game interface, shown in Figure 1, to capture human demonstrations. In both games, an agent (a panda bear) runs on a spherical surface collecting a series of colored balls. In PandaSequential, the agent must pick balls of different colors in a specific, fixed order, while in RainbowPanda, the agent is tinted with the color of the balls it is allowed to pick at any given moment. The color of the agent in RainbowPanda changes when the agent picks the last ball that matches the current color and may also change, with a small fixed probability, at any time step. With both domains, the initial position of the balls is assigned randomly at the start of each episode.

The state features are the distance and angle to each ball, relative to the agent. In RainbowPanda there are two balls of each color, so there are separate features for the closest ball and further ball from the agent. Distance and angle are measured in radians, and when a ball is not present (it has already been picked up), both features take a value outside of their normal range. RainbowPanda also has a discrete feature that contains the color of the agent, which is the color of the balls that the agent is allowed to pick up.

On both domains, the actions are move forward, backward, rotate right, rotate left, and no operation. The agents move backwards at a fourth of the speed they can move forward. The balls are picked up just by touching them. If the touched ball is the correct one to pick, the agent receives a positive reward and the ball disappears. Nothing happens if the agent touches a different ball. To compute the discounted reward, we use a discount factor $\gamma < 1$. The games were played at 20 frames per second, and this was also the rate at which the state was updated in the screen and an action was taken.

Both domains have similar interfaces, but their subtask structure is quite different. In PandaSequential, the subtasks have a fixed order, while in RainbowPanda there is no fixed order and every subtask may appear many times in the same episode. Additionally, in the first domain the current subtask is determined by the presence or absence of the balls (continuous variables) while in the second domain the current subtask is encoded in the color of the agent (a discrete variable).

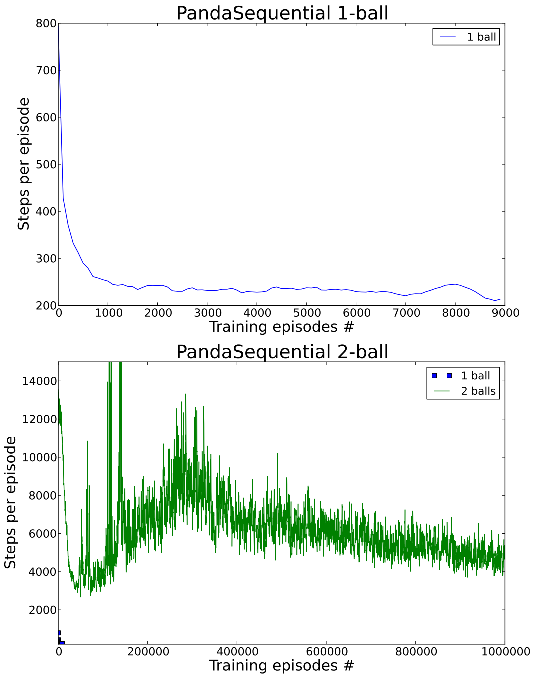

Figure 2: Results using Sarsa(λ) algorithm on a simplified version of the domain with only one or two balls. 1 extra ball decreases performance 2 orders of magnitude, even if the algorithm is left training 2k times longer.

To compare LfD, AfD and ADA, we captured a set of human demonstrations. We obtained 400 episodes for the PandaSequential domain (about 2 hours of gameplay) and 200 and 300 episodes, from different individuals, for the RainbowPanda domain (1.5h and 2.25h, respectively).

4.2 Results

Using the domains described above, we compared ADA with Reinforcement Learning using Sarsa(λ), Learning from Demonstration using a C4.5 decision tree, and Abstraction from Demonstration. We first discuss RL and LfD, then our algorithm, and finally we compare the abstractions that our algorithm and AfD found.

4.3 Reinforcement Learning using Sarsa

For Sarsa, we discretized the continuous values in 64 bins. The results were very poor in these domains because the dimensionality of the problems is so high. The simpler domain, PandaSequential, still has 64^6 possible states, for a total of about 343 billion Q-values. To ensure our implementation of the algorithm was correct and find its limits, we tested it with simplified versions of the game, with only one and two balls. The results are shown in Figure 2. We can see that Sarsa performs reasonably well for the case of only one ball (4096 states), but no longer so for the two-ball game (16.8 million states), even if leaving the algorithm running for 8 days in a modern computer, this is, 2000 times

Table 1: LfD results over 10K episodes, using a C4.5 decision tree. Avg. steps computed only over successful episodes.

Domain/Player	Episodes	Success rate	Avg. steps
Sequential	100	0.28%	241.21
	200	0.44%	227.79
	400	0.82%	242.83
Rainbow/A	100	8.1%	1543.84
	200	2.76%	1650.37
Rainbow/B	100	0.27%	1518.70
	200	0.27%	1994.92
	300	0.73%	1901.51

Table 2: Comparison of human performance, ADA, and ADA+RL, measured in number of steps to task completion averaged over 10 thousand episodes.

Domain/Player	Episodes	Human	ADA	ADA+RL
Sequential	100	322.67	360.75	267.11
	200	318.71	360.11	
	400	311.30	335.92	
Rainbow/A	100	525.33	-	-
	200	504.71	626.27	466.59
Rainbow/B	100	540.03	596.46	
	200	536.43	593.45	
	300	533.56	583.34	

longer than it took for the 1-ball policy to converge. The policy performance keeps improving, but at an extremely slow rate. Therefore, Sarsa is not an effective option for these domains.

4.4 Learning from Demonstration using C4.5

To compare with the performance of traditional LfD techniques, we trained a C4.5 decision tree with the demonstrations captured from human players, in a purely supervised learning fashion. We can see in Table 1 that LfD also performed poorly. The best result we obtained, using all available demonstrations, was a policy that would reach the goal state in less than 3% of the episodes[1].

4.5 Automatic Decomposition and Abstraction from Demonstration

To use ADA in our domains, we discretized the continuous values in 64 bins (same as for RL/Sarsa), used $\epsilon = 0.1$ and considered as candidate boundaries every possible threshold on every feature of the domain. ADA was much more effective than the other methods on both domains, and led to near-optimal polices that succeeded on every episode. Table 2 shows that we obtained policies comparable to those of the human teacher. Note that even though the average number of steps is slightly higher than for the LfD policy in the Sequential domain, this is averaged over all episodes, while the number for LfD is only averaged over the small percentage of episodes that LfD is able to resolve.

The success of ADA, compared with LfD and Sarsa, is due to its finding the right decomposition of the domains. For both domains, the algorithm builds an abstraction that focuses only on the angle with respect to the agent of the next ball to be picked up. Which ball is the target ball depends on what balls are present for the Sequential domain, and on what is the current color the agent is targeting for the Rainbow domain. The algorithm was able to identify the right boundary on each domain. It was a surprise that only the angle, and not the distance to the ball, was necessary, but it is easy to see that a satisficing policy can be found using only the angle: rotate until the ball is in front of the agent and then go forward. In fact, this was what the human players were doing, except in the rare case where the ball to pick up was right behind the agent; since the agent moves faster forward than backward it was usually not worth moving backwards.

Only one case in Table 2 did not produce the abstraction described above. Rainbow/B-100 episodes did not find any abstraction. This was due to m_{ss} being higher than a third of the total number of samples, therefore it could not find any of the 3 subtasks, one per color and roughly of the same size, that were found in the other cases. We tested a lower value for ϵ and in that case the usual abstraction was found.

In the same table we can see results for $ADA + RL$, applying policy search on top of the policy found by ADA. The abstraction built by ADA may prevent boot-strapping algorithms such as Sarsa or Q-learning from converging, but with only 192 states in the abstraction and a good starting policy, we can use direct policy search methods. We could obtain good results by just iteratively changing the policy of each state and evaluating the effect in performance using roll-outs.

Notice that, using this additional policy improvement step, we can find policies that are better than those demonstrated by the human teachers. The policies found were better than those demonstrated in three ways. First, the preferred action for states that were rarely visited was sometimes incorrect in the ADA policy because there were not enough samples in the demonstrations. ADA+RL could find the best action for these uncommon states. Second, human players would make the agent turn to face the target ball and then move forward when the relative angle to the ball was less than 15 degrees. ADA+RL found it was more efficient to turn until the angle to the ball was less than 3 degrees and only then move forward. Third, ADA policies assign some probability to each action depending how often it is taken in the demonstrations for a particular state. ADA+RL can identify which actions were not appropriate for the state and never execute them even if they appear in the demonstrations, maybe because of distractions or errors from the teacher. In short, the policy found by ADA+RL was a *more precise* and *less noisy* version of the policy derived directly from the demonstrations.

4.6 Abstraction from Demonstration

Finally, we tried AfD in the domains, using the abstraction algorithm described in Section 3.4 for the whole state space. In the Sequential domain, AfD would identify as the only useful feature the position of the first ball. This abstraction leads to a policy that can find the first ball quickly but can only perform a random walk to find the other two balls. The large difference in mutual information between each ball position and the action is due to the fact that while

[1] It should not be a surprise that sometimes, with more samples, the number of average steps on successful episodes increases. This is due to the policy being able to deal with more difficult episodes (remember that the initial placement of the balls is random) that require more steps to complete.

the first ball is present, its position is significant for the policy; however, the second ball is significant for the policy only half of the time it is present, and the third ball only a third of the time it is present.

Regarding AfD for the Rainbow domain, because the active color at each moment is chosen at random, the mutual information measures between each ball relative position and the action are similar. In this case, AfD is able to identify the true relevant features, *i.e.,* the relative position to the closest ball of each color. Due to the nature of AfD abstraction, we could not use bootstrapping algorithms such as Sarsa, and $64^3 = 262144$ states are too many for our naive policy search, so we tried to obtain a policy using Monte Carlo methods. Unfortunately, these are known to be much slower to converge than Sarsa and, even after experimenting with various exploration parameters, we could not reach a policy better than a random walk.

We can thus conclude that for complex domains that can be decomposed in different subtasks, ADA can find policies better than those demonstrated by humans, while traditional LfD, RL and AfD cannot find policies significantly better than a random walk.

4.7 Discussion

There are several advantages of ADA over traditional LfD. ADA can obtain much better performance from a small set of samples, while LfD often needs more samples than it is practical to obtain. In fact, for our two domains, we did not have the resources to collect a number of demonstrations large enough to obtain reasonable performance with LfD. Additionally, even with an arbitrarily large number of demonstrations, it is likely that ADA+RL can obtain policies better than those demonstrated and thus beat LfD, whose policy performance is limited by the quality of the demonstrations used.

Regarding RL techniques, it may seem unfair to compare those with ADA, as RL does not use the human demonstrations that are necessary for ADA. Yet, in the domains considered, even if we account for the time and cost of acquiring the human demonstrations, ADA still outperforms RL. We have seen in Section 4 that even for the simpler version of the Sequential domain with 2 balls, we still do not have a reasonable policy after a week. Using ADA, just with half an hour of human demonstrations and less than ten minutes of computing, we obtain a satisficing policy.

ADA is successful in these domains because it can find different state abstractions for different regions of the state space. Abstraction for Demonstration (AfD), a previous technique combining RL and LfD, only finds a single abstraction for the whole domain, and therefore it does not help much in the domains considered since all balls are relevant at some point during the task. AfD could make a small difference in performance by ignoring the distance to the balls, but the complexity of an AfD policy would still grow exponentially with the number of balls in the domain, while the complexity of the ADA policy grows linearly with the number of balls in the domain.

Obviously ADA cannot help if there is no possible decomposition of the domain and every feature is important at every moment; however we conjecture that such complex policies are rare, especially among tasks that can be demonstrated by humans. Humans have a limited capacity of attention and accomplish complex tasks by dividing them in manageable pieces. Therefore, if we have a set of demonstrations of a complex task, it is likely there are task decompositions to be found.

One “unfair” advantage of ADA over the other methods is that we must provide it with a set B of candidate boundaries, which is after all a form of domain information. In principle we could chose as boundary every possible subset of S, but this would be computationally intractable, so we must explicitly choose the candidate boundaries. This is a small price to pay for the performance gains of the algorithm. As a default choice, axis-aligned boundaries, i.e., thresholds in a single feature, are a compact class that works well across a diverse range of domains. They would work for the Taxi domain, which is the typical example of task decomposition in RL, using as boundary whether the passenger has been picked up or not yet. If the features are learned from low-level sensing information using unsupervised feature learning techniques, it is likely that one of the generated features will provide adequate thresholds. Additionally, many learning algorithms have similar kinds of bias; *e.g.,* decision trees also consider only thresholds in a single feature, just like ADA in our experimental setup.

A real limitation of ADA is that it does not consider second-order mutual information relationships, and these can be relevant. For example we can imagine a domain where the action to take depends on whether two independent random variables have the same value. The mutual information between each variable and the action might be 0, but the mutual information between both variables and the action would account for all the entropy of the action. We have decided to use only first-order mutual information because we believe it is enough to obtain a good decomposition of a wide range of problems and because the number of samples needed to get an accurate estimate of higher-order relationships is much larger. However, if a large number of demonstrations is available, ADA can be easily extended to use these additional mutual information measures.

One additional advantage of ADA, is that it can be used as part of a larger system of **transfer learning**. Once an autonomous agent learns a new skill and the subtasks it decomposes to, the subtask policies can be useful for other skills that may be decomposed in a similar way. An agent might, *e.g.,* as part of the policy improvement step for a specific subtask, try policies of previously learned subtasks that have the same abstraction, maybe after comparing policies and determining that the subtasks are similar. Demonstrating the utility of ADA for transfer learning is an important area of future work.

5. CONCLUSIONS

We have introduced Automatic Task Decomposition and State Abstraction from demonstration (ADA), an algorithm that leverages a small set of human demonstrations to decompose a skill in different subtasks, find abstractions for each of these subtasks, and build a compact satisficing policy for the skill. We have shown experimentally that, with a small number of demonstrations, ADA can easily find a policy for problems that are intractable using traditional RL and LfD techniques. Furthermore, we have shown that, given the structure of the policy that ADA finds, it can be improved to obtain a policy that outperforms the human teachers.

With this work we show that mutual information can be used to extract useful domain knowledge of a sequential decision process from a set of human demonstrations. In the future, we plan to build upon this technique and combine it with function approximation and transfer learning.

6. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under Grant No. 0812116 and Obra Social “la Caixa”.

7. REFERENCES

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. *International Conference on Machine Learning*, page 1, 2004.

[2] R. Aler, O. Garcia, and J. Valls. Correcting and improving imitation models of humans for robosoccer agents. In *IEEE Congress on Evolutionary Computation*, volume 3, 2005.

[3] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[4] L. C. Cobo, Z. Peng, C. L. Isbell, and A. L. Thomaz. Automatic Abstraction from Demonstration. *International Joint Conference on Artificial Intelligence*, pages 1243–1248, 2011.

[5] O. Şimşek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. *International Conference on Machine Learning*, page 95, 2004.

[6] T. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *International Conference in Machine Learning*, pages 118–126, 1998.

[7] A. Farahmand, M. Ghavamzadeh, C. Szepesvari, and S. Mannor. Regularized policy iteration. *Advances in Neural Information Processing Systems*, 21:441–448, 2009.

[8] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *International Conference on Machine Learning*, pages 234–250, 2002.

[9] N. K. Jong and P. Stone. State Abstraction Discovery from Irrelevant State Variables. *International Joint Conference on Artificial Intelligence*, (August):752–757, 2005.

[10] A. Jonsson and A. Barto. Automated state abstraction for options using the U-tree algorithm. *Advances in Neural Information Processing Systems*, pages 1054–1060, 2001.

[11] P. W. Keller, S. Mannor, and D. Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. *International Conference on Machine Learning*, pages 449–456, 2006.

[12] W. Knox and P. Stone. Combining Manual Feedback with Subsequent MDP Reward Signals for Reinforcement Learning. *Annual International Conference on Autonomous Agents and Multiagent Systems*, pages 10–14, 2010.

[13] J. Z. Kolter and A. Y. Ng. Regularization and feature selection in least-squares temporal difference learning. *International Conference on Machine Learning*, 94305:1–8, 2009.

[14] J. Kristeva. *Strangers to ourselves*. European Perspectives: A Series In Social Thought And Cultural Criticism. Columbia University Press, 1991.

[15] L. Li, T. J. Walsh, and M. L. Littman. Towards a Unified Theory of State Abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.

[16] N. Mehta, M. Wynkoop, S. Ray, P. Tadepalli, and T. Dietterich. Automatic induction of MAXQ hierarchies. In *NIPS Workshop: Hierarchical Organization of Behavior*, pages 1–5, 2007.

[17] W. Smart and L. Pack Kaelbling. Effective reinforcement learning for mobile robots. *IEEE International Conference on Robotics and Automation*, pages 3404–3410, 2002.

[18] M. Stolle and D. Precup. Learning Options in Reinforcement Learning. *Abstraction, Reformulation, and Approximation*, pages 212–223, 2002.

[19] P. Zang, P. Zhou, D. Minnen, and C. Isbell. Discovering options from example trajectories. *International Conference on Machine Learning*, pages 1217–1224, 2009.