Active Learning of Hierarchical Policies from State-Action Trajectories

Mandana Hamidi, Prasad Tadepalli, Robby Goetschalckx, Alan Fern
School of EECS, Oregon State University, Corvallis, OR, USA
{ hamidi, tadepall, goetschr, afern}@eecs.oregonstate.edu

Abstract

While most work on trajectory mining is applied to predict movements of mobile users, in this paper we consider a more general problem of building behavior models of users from their state-action trajectories. We assume that the user behavior can be compactly modeled as a Probabilistic State-Dependent Grammar (PSDG) which represents a hierarchical policy. The key problem is that while the states and actions of the user are directly observed, his intentional structure is not. We propose to learn the user’s policy from a set of selected trajectories and intention queries at selected states in the trajectory. Our main contributions are an algorithm for learning hierarchical policies from state-action trajectories, and principled heuristics for selecting suitable trajectories and intention queries. Experiments in multiple domains show that our approach is effective and more sample-efficient than learning non-hierarchical policies.

1 Introduction

There has been much active research in mining the trajectories of mobile users to extract common patterns and behavior models (Zheng et al. 2009)(Yan et al. 2013)(Ying, Lee, and Tseng 2014). However much of this work is confined to location-specific trajectories and tied to the geography. In this paper, we consider more general kinds of trajectories, namely trajectories of state-action pairs, and propose algorithms for learning behavior models of users from such trajectories. Our notion of trajectories applies to geographical trajectories, e.g., the trajectory of a taxi that drives around in a city to pick up and drop off passengers. Importantly, it also applies to other tasks, e.g., the trajectory of states and actions of a user while he cooks a dish or plays a video game. Considering trajectories in this richer context enables us to develop algorithms that are more generally applicable and exploit more general kinds of structure than is present in geographical trajectories.

In this paper, we consider one such general structure, namely task hierarchies. Many of the tasks that people do exhibit a hierarchical intention structure, where the highest level tasks correspond to high level goals such as making lunch and the lowest level subtasks correspond to primitive actions such as picking up a spoon. The key problem in learning from trajectories is that only the primitive actions at the lowest level are observable, and the intermediate intention structure is not. This leads to significant ambiguity in interpretation. For example, understanding and predicting whether a user will open a drawer or an oven in a given state might be difficult. However, the task might be simpler if we knew that the user intends to heat something. Indeed, theory suggests that learning of hierarchical policies from state-action trajectories is fundamentally more challenging than learning flat policies. In particular, it has been shown that in the PAC-learning setting, learning flat policies from demonstrations in the form of decision lists can have polynomial complexity, while learning hierarchical policies is an NP-Hard problem (Khardon 1999). However, the same work showed that if demonstrations are annotated with the user’s intention structure, then learning hierarchical policies becomes tractable.

The latter result motivates the main idea of our approach, which is to ask queries to the user to efficiently acquire the intention structure and learn from the annotated data. In particular, we study the problem of learning the annotations via a two-level active learning process. The approach iteratively selects the “most useful” trajectory for which to request annotation (level one), and then intelligently queries the teacher about their intentions at specific times in order to acquire the intention annotation as efficiently as possible (level two).

Our contributions are as follows. We first show that Probabilistic State-dependent Grammars (PSDGs) (Pynadath and Wellman 2000) strictly generalize common prior representations such as MAXQ hierarchies (Dietterich 2000) to represent hierarchical policies. Second, we develop a new algorithm for learning PSDGs from sets of demonstrations and answers to intention queries, leveraging the relationship of PSDGs to probabilistic context-free grammars. Third, we introduce a novel two-level active learning approach for selecting demonstrations to annotate and then acquiring the intention annotations. For acquiring annotations we draw on ideas from Bayesian active learning. However, for trajectory selection, we demonstrate the surprising results that selecting demonstrations based on the common principle of maximum entropy can perform very poorly. We analyze this observation and develop a principled heuristic for selecting tra-
jectories that is significantly more effective. Our final contribution is to demonstrate the effectiveness of the learning algorithm and our active learning heuristics compared to competitors in three synthetic domains.

2 Problem Setup and Background

We consider a deterministic state space described by a set of states $S$, a set of primitive actions $A$, and a transition function $\delta(s,a)$ returning state arrived at after taking action $a$ in $s$. For simplicity of description we assume $\delta$ is deterministic, but note that our algorithms easily apply to stochastic domains as well. We assume that a set of task names $T$, and their associated termination conditions are known to the learner. Following the MAXQ hierarchical reinforcement learning framework (Dietterich 2000), we define a task hierarchy as a directed acyclic graph over $T \cup A$, where there is an edge between two tasks if the first task can call the second task as a subroutine. The primitive actions $A$ are the leaves of the hierarchy and can be executed directly in 1 step and change the state. The child nodes of a task are called its subtasks. We assume that all hierarchies have a special root task labeled $\text{Root}$. Some tasks have a small number of parameters which allows more efficient generalization. Some example task hierarchies (those used in the experiments in this paper) are shown in Figure 1. In contrast to the MAXQ framework, the task hierarchy is not known to the learner.

A task is executed by repeatedly calling one of its subtasks based on the current state until it terminates. When a subtask terminates, the control returns to its parent task. We define a deterministic hierarchical policy $\pi$ as a partial function from $T \times S$ to $T \cup A$ which is defined only for states that do not satisfy their termination conditions for any task and maps to (call) one of its subtasks. A trajectory or a demonstration $d$ of a policy $\pi$ starting in state $s$ is a sequence of alternating states and primitive actions generated by executing that policy from $s$ starting with the $\text{Root}$ task. We also assume that every task $t_i$ has a special symbol $t'_i$ that denotes its termination, which is included in the trajectory at the point the task terminates. More formally, a demonstration $d$ of a policy $\pi$ in a state $s$ can be defined as follows, where $;$ is used for concatenation, $t \gamma$ represents a task stack with $t$ as the topmost task and $\gamma$ is the rest of the stack to be executed.

$$d(s, \pi) = d(s, \pi, \text{Root})$$

$$d(s, \pi, t_1 \gamma) = t'_1 d(s, \pi, \gamma) \text{ if } t_1 \in T \text{ and } t_1 \text{ terminates in } s$$

$$d(s, \pi, t_1 \gamma) = d(s, \pi, \pi(s, t_1) t_1 \gamma) \text{ if } t_1 \in T \text{ and } t_1 \text{ does not terminate in } s$$

$$d(s, \pi, t_1 \gamma) = t_2 ; s' ; d(s', \gamma) \text{ where } t_2 \in A \text{ and } s' = \delta(s, t_1)$$

In this paper we consider the problem of learning a deterministic hierarchical policy and the associated task hierarchy from a set of demonstrations. The key problem in doing so is that the demonstrations only include the primitive actions and the termination symbols of the subtasks when they are completed. In particular, they do not indicate the starting points of various subtasks, which leaves significant ambiguity in understanding the demonstration. Indeed, it has been shown that the problem of inferring hierarchical policies is NP-hard when such annotations of subtasks are not given (Khardon 1999). This is true even when there is a single subtask whose policy is fully specified and it is known where the subtask ends in each trajectory. All the complexity comes from not knowing where the subtask begins. Hence we consider the problem of efficiently acquiring such annotations by asking a minimum number of queries to the expert.

3 Probabilistic State-Dependent Grammars

The above formulation suggests that a hierarchical policy can be viewed as a form of Push Down Automata (PDA), which can be automatically translated into a Context Free Grammar (CFG). Unfortunately such a CFG will have a set of variables that corresponds to $S \times T \times S$, which makes it prohibitively complex. Instead, we adapt an elegant alternative formalism called Probabilistic State-Dependent Grammar (PSDG) to represent the task hierarchies (Pynadath and Wellman 2000). A PSDG is a 4-tuple $(V, \Sigma, P, Z)$, where $V$ is a set of variables (represented by capital letters), $\Sigma$ is the terminal alphabet, $P$ is a set of production rules, and $Z$ is the start symbol. PSDGs generalize CFGs in that each production rule is of the form $s, t \gamma \to \gamma$, where $s$ is a state, $t$ is a variable, and $\gamma$ is a string over variables and terminal symbols. The above production rule is only applicable in state $s$, and reduces the variable $t$ to the sequence of variables and terminals described by $\gamma$.

It is desirable for a PSDG to be in Chomsky Normal Form (CNF) to facilitate parsing. We can represent the hierarchical policies in the MAXQ hierarchy as a PSDG in CNF as follows. For each task $t \in T$, we introduce a variable $V_t \in V$, and a terminal symbol $t'$ that represents the termination of $t$. For each primitive action $a \in A$ and state $s \in S$, we introduce a variable $V_{a,s} \in V$ and a terminal string $a; s$. $\text{Root}$ is the start symbol. Further, it suffices to restrict the rules to the following 3 types:

1. $s, t_i \to t_j t_i$, where $t_i$ is a non-primitive subtask of $t_i$
2. $s, t_i \to a_j ; \delta(s, a_j)$, where $a_j$ is a primitive action, and $\delta$ is the transition function
3. $s, t_i \to t'_i$ where $t'_i$ is a symbol representing the termination of task $t_i$.

The first rule represents the case where $t_i$ calls $t_j \in T$ when in state $s$ and returns the control back to $t_i$ after it is done. The second rule allows a primitive action in $A$ to be executed, changing the state as dictated by the transition function. The third rule is applicable if $s$ satisfies the termination test of $t_i$. In a deterministic hierarchical policy, there is a single rule of the form $s, t_i \to \gamma$ for each state-task pair $s, t_i$. A PSDG, on the other hand, allows multiple rules of the above form with right hand sides $r_1, \ldots, r_m$, and associates a probability distribution $p(|s, t_i)$ over the set $r_1, \ldots, r_m$. As is normally the case, we assume that a state is represented as a feature vector. Since only a small set of features is usually relevant to the choice of a subtask, we specify the above distributions more compactly in the form of $p(r_i | s_1, \ldots, s_k, t_i)$, where $s_1, \ldots, s_k$ are the only features which are relevant for choosing the subtask. The
set of productions of the PSDG which is equivalent to the task hierarchy of the taxi domain is shown below without the probability distributions that correspond to a specific hierarchical policy. It is easy to see that for any deterministic hierarchical policy and initial state, the corresponding deterministic PSDG, when started from the Root symbol and the same initial state, will derive the same demonstration. Table 1 represents the skeleton PSDG of taxi domain which does not include the state constraints and the result function.

However, the PSDG formalism is more general in that it allows us to represent stochastic hierarchical policies. In this work, we exploit this property and view the PSDG as representing a distribution over deterministic hierarchical policies. This allows us to adopt the efficient algorithms developed for probabilistic CFGs such as the inside-outside algorithm to learn PSDGs (Lari and Young 1990). It also allows us to efficiently compute some information-theoretic heuristics which are needed to guide active learning.

4 Active Learning of Hierarchical Policies

We now present our main algorithm for active learning of PSDGs from sets of demonstrations (see Algorithm 1), followed by a more detailed description of each algorithm component. The algorithm takes a set of task names, primitive actions, and trajectories as input (lines 1 & 2). It then constructs an initial uninform ed distribution of PSDGs by including every possible production in some MAXQ hierarchy with all production distributions initialized to be uniform (line 3). It then iterates over the following steps. First, the learner heuristically select a trajectory, trj, to annotate (line 5, details in Section 4.4). It then parses the trajectory using a version of the inside-outside algorithm and constructs a CYK table data structure, which compactly represents all possible parse trees of the trajectory and their probabilities (line 6, details in Section 4.1). Finally in lines 7 thru 21 (details in Section 4.2), it uses Bayesian active learning to ask queries about the intention structure of the trajectory. The information gathered from the queries is used to update the CYK-table (line 16). Whenever an unambiguous parse is found for a subtrajectory, it is used to generate concrete examples to learn a production rule (lines 17-20, details in Section 4.3). The rest of the section describes the different steps of the algorithm in more detail.

4.1 Parsing the Trajectories

We apply the inside-outside algorithm to parse a given trajectory using the current PSDG, which is assumed to be in CNF. The algorithm is based on dynamic programming and incrementally builds a parsing table called CYK table to compactly represent all possible parse trees of the trajectory and their probabilities (Lari and Young 1990). The algorithm recursively computes two different probabilities: 1) the inside probability, \( \alpha_{i,j}(B) \), which is the probability that \( B \) derives the subsequence of \( trj \) from \( i \) to \( j \), and 2) the outside probability, \( \beta_{i,j}(B) \), which is the probability that trajectory \( trj_1, ..., trj_{j-1}, B, trj_{j+1}, ..., trj_n \) can be derived from the start symbol \( S \). The current parameters of the distributions of the PDSG (which are uniform in the beginning) are used to compute these probabilities. Since all the intermediate states are observed, the state-dependence of the grammar does not pose any additional problems. Thanks to the context-free nature of PSDG, and the full observation of the
4.2 Query Selection via Bayesian Active Learning

We now consider selecting the best queries to ask in order to uncover the intention structure (parse tree) of a selected trajectory. Each query is relatively simple to answer and involves identifying whether a particular task was being pursued at a particular moment in the trajectory. In particular, each query highlights a certain trajectory state $s$ and asks if a particular task $B$ is part of the intention of the user in that state (i.e., part of the task stack). The answer is "yes" if $B$ is part of the true task stack at that point, and "no" otherwise.

The question now is how to select such queries in order to efficiently identify the parse tree of the trajectory. For this we follow the framework of Bayesian Active Learning (BAL) for query selection (Golovin, Krause, and Ray 2010), which considers the efficient identification of a target hypothesis among candidates via queries. We begin by generating a large set of $N$ hypothesis parse trees for the trajectory by sampling from the parse tree distribution induced by our current PSCFG estimate (line 7). BAL then attempts to heuristically select the most informative query for identifying the correct parse tree. After receiving the answer to each query the learner removes all parse trees or hypotheses that are not consistent with the answer and updates the CYK table appropriately. It also updates the CYK table to exclude all entries that are not consistent with the answer to the query. Querying continues until a single parse tree remains, which is taken to be the target. If all parse trees happen to be eliminated then we move on to select another trajectory, noting that information gathered during the querying process is still useful for learning the PSCFG. An alternative would be to sample a new set of $N$ trees based on the updated CYK table and continue querying.

It remains to specify the query selection heuristic. A standard BAL heuristic is to select the query that maximizes the expected information gain ($IG$) of the answer. Prior work (Golovin, Krause, and Ray 2010) has shown that this heuristic achieves near optimal worst case performance. It is straightforward to show that this heuristic is equal to the entropy of the answer distribution, denoted by $H(Answer)$. Thus, we select the question that has maximum entropy over its answer distribution (line 13).

4.3 Example Generation and Generalization

Recall that the answers to the queries are simultaneously used to update the CYK table as well as removing the inconsistent parse trees from the sample. We consider a parse of a trajectory segment between the indices $i$ and $j$ to be unambiguous if the inside and outside probabilities of some variable (task) $B$ for that segment are both 1. When that happens for some task $B$, and its subtasks, say $C$ and $B$, we create positive and negative training examples of the production rule for the pair $(s, B)$, where $s$ is the state at the trajectory index $i$ where $B$ was initiated. The positive examples are those that cover the correct children of $B$, namely $C$ and $B$, and the negative examples are those that were removed earlier through queries. To generalize these examples and ignore irrelevant features in the state, we employ a relational decision tree learning algorithm called TILDE (Blockeel and De Raedt 1998). TILDE uses a form of information gain heuristic over relational features and learns the probabilities for different right hand sides of production rules of PSDG. Ideally only the features of the state relevant for the correct choice of the subtask are tested by the tree. These probabilities are used to compute the $\alpha$s and $\beta$s during the future parsing steps.

4.4 Trajectory Selection Heuristics

In this section we focus on the problem of trajectory selection. On the face of it, this appears to be an instance of the standard active learning problem for structured prediction problems such as parsing and sequence labelling (Baldridge and Osborne 2003; Settles and Craven 2008). A popular heuristic for these problems is based on "uncertainty sampling," which can be interpreted in our context as picking the trajectory whose parse is most ambiguous. A natural measure of the ambiguity of the parse is the entropy of the parse tree distribution, which is called the "tree entropy" given by $TE(trj) = - \sum_{v \in V} p(v|trj) \times \log(p(v|trj))$, where $V$ is a set of all possible trees that our current model generates to parse $trj$. The probability of each $trj$ is $p(trj) = \sum_{v \in V} p(v, trj)$.
which is the sum over the probability of all possible trees, where \( p(v, trj) \) denotes the probability of generating \( trj \) via the parse tree \( v \). Both \( p(trj) \) and \( TE(trj) \) can be efficiently computed using the CYK table without enumerating all parse trees (Hwa 2004). Tree entropy is one of the first heuristics we considered and evaluated. Somewhat surprisingly we found that it does not work very well and is in fact worse than random sampling. One of the problems with tree entropy is that it tends to select long trajectories, as their parse trees are bigger and have a bigger chance of ambiguity. As a result they require more effort by the user to disambiguate. To compensate for this researchers have tried length-normalized tree entropy as another heuristic (Hwa 2004). We also evaluated this heuristic in our experiments.

Our third heuristic, cost-normalized information, is based on the observation that the tree entropy represents the amount of information in the correct parse tree, and hence can be viewed as a proxy for the number of binary queries we need to ask to identify the parse. Indeed, we have empirically verified that the number of queries needed grows linearly with the tree entropy. However it is not a good proxy for the information we gain from a labeled trajectory. Indeed, it would be best to use a trajectory with zero tree entropy, i.e., a trajectory with an unambiguous parse to learn, as it needs no queries at all. However, such trajectories might also be useless from the learning point of view, if they are already predicted by the current model. An ideal trajectory would be something the learner has only a low probability of generating by itself, but would require the least number of queries to learn from, given that trajectory.

This suggests that given the same cost, we should select the trajectories that are most informative or most surprising. The amount of surprise or novelty in an event \( E \) is nicely formalized by Shannon’s information function \( I(E) = -\log(p(E)) \). Thus we are led to the heuristic of maximizing cost-normalized information of the trajectory, which is approximated by \(-\log(p(trj))/TE(trj)\). Fortunately the probability of the trajectory is something we can easily compute from the CYK table.

### 5 Empirical Results and Discussion

In order to evaluate the performance of the proposed algorithm, we carried out two experiments employing three synthetic domains (Figure 1).

**Taxi Domain:** In this domain, there is a taxi that wishes to transport a passenger from a source location to a destination location within a \( 5 \times 5 \) grid. The passenger’s destination is restricted to one of four special locations on the grid denoted by \( R, G, B, Y \). The passenger’s location could be set to \( R, G, B, Y \) or in-taxi. The taxi problem is episodic. In each episode, the taxi starts in a randomly-chosen state and with a randomly-chosen amount of fuel (ranging from 5 to 12 units). The taxi must go to the passenger’s location, pick her up, go to the destination location, and drop her off. The episode ends when the passenger is deposited at the destination location. The actions that the agent can perform are to navigate in 4 directions, pickup, drop off, and fill up (Dietterich 2000).

**Stochastic Taxi Domain:** In the stochastic version of the Taxi domain, we assumed that whenever the taxi attempts to move, with probability 0.7 the next state is exactly the one that the selected action causes to move, with probability 0.1 the next state is not changed, and with probability 0.2 the next state is randomly selected.

**RTS Domain:** In this domain the agent is in a grid of size \( 7 \times 7 \), in which each cell may contains resources (i.e. gold, food), locations (e.g. enemy castles, granaries, banks), or dragons. The agent’s objective is to gather resources and kill enemies. To gather a resource it has to be collected and deposited to a designated location (e.g. gold in a bank, food in a granary). There are two locations for each resource and its storage. There are two kinds of enemies, red and blue. The agent has to kill the enemy dragon and destroy an enemy castle of the same color as the dragon. The goals and sub-goals are specified using a relational hierarchy. The episode ends when the user achieves the highest level goal. The actions that the user can perform are to navigate in 4 directions, open the 4 doors, pick up, put down and attack. (Natarajan et al. 2011).

In all three domains, we collected a set of training and test trajectories from a hand-coded hierarchical policy. The trajectories are stored with the corresponding hierarchical intention structures which are used to answer queries. The results are averaged over 10 runs of the system with the same hierarchical policy but with a different set of randomly generated training examples in each run.

**Experiment 1:** In this experiment, we evaluated the effect of the task hierarchy on learning policies. We compared the performance of our proposed algorithm, whose heuristic is cost-normalized information, with the flat policy learning algorithm, whose policies are learned using TILDE (Blockeel and De Raedt 1998). Figures 2 (a), (b), and (c) shows goal-success accuracy vs. the number of the training data in three domains. The performance metric, goal-success accuracy, is the percentage of the learned trajectories that reaches to the goal state using the learned policies and measured on the test set. The results show that in both deterministic and stochastic domains the proposed algorithm is able to learn the task faster and achieve a higher level of performance than flat policy learning algorithm. In stochastic Taxi domain in comparison to the deterministic Taxi domain, the learner needs more training data because the agent could potentially enter more states than in the deterministic case, and needs to learn how to act in those states.

**Experiment 2:** In this experiment, we compared the performance of our preferred heuristic, namely, maximizing cost-normalized information, \(-\log(p(trj))/TE(trj)\), with three other trajectory selection strategies for learning hierarchical policies: 1) Random selection, 2) Maximizing tree entropy, \( TE(trj) \), and 3) Maximizing length-normalized tree entropy, \( TE(trj)/length(trj) \). Each heuristic was trained on the same hierarchical policy and the same set of training examples. The learned model is evaluated on a test set generated from the same target policy as the training set. Our performance metric is accuracy of the test data that measures the degree to which the actions and the intentions of the target policy match those of the learned policy at different points of the
trajectory on a scale 0 to 1.

Figures 2(d), (e), and (f) show accuracy vs. the number of queries for each domain. Each point on the plot shows the performance after learning from one more trajectory than the previous point. The curve is flat between the trajectories because the performance does not change until a new trajectory is correctly parsed and generalized. Unlike the typical learning curves, the different points are not uniformly spaced because the number of queries used to learn each trajectory is usually different. One clear trend is that fewer queries are needed to disambiguate each successive trajectory. This is to be expected because it gets easier to infer the intentions behind actions, as more is learned about the target policy. Indeed, towards the end of the trial, many trajectories need only 1 or 2 queries to disambiguate them.

We first observe that selecting trajectories using the tree entropy performs poorly, even worse than random in all 3 domains. The reason is that the tree entropy does not take into account the cost of answering queries. Maximizing tree entropy encourages the selection of the most ambiguous trajectory, which maximizes the number of queries asked. Normalizing by length mitigates this effect somewhat and makes it perform closer to or better than the random strategy in the RTS domain. However, the random strategy still performs better in the Taxi domain. Our preferred heuristic, maximizing cost-normalized information, consistently performs better than all other heuristics in all 3 domains. The reason for this is that it correctly takes into account both the information gain from a trajectory and the cost of querying in uncovering this information. The shape of the different curves clearly suggests that it allows the learner to learn from more trajectories by spending fewer queries on each trajectory, and to reach better performance after the same number of queries as the other heuristics.

6 Summary

We examined the problem of actively learning deterministic hierarchical policies from trajectories generated by a hierarchical policy. We showed that a previously introduced plan representation language, namely PSDG, nicely captures hierarchical policies in the MAXQ framework and is strictly more general. We introduced a two-level active learning approach, where the top level employs the novel heuristic of cost-normalized information to select trajectories and the lower level applies Bayesian active learning over sampled parse trees. Experimental results on three benchmark problems indicate that our approach, which learns hierarchical policies in a query-efficient manner, can facilitate significant speedup in learning policies in comparison to the ordinary flat policy learning algorithm.
7 Acknowledgments
The authors acknowledge support of the ONR ATL program N00014-11-1-0106.

References


