Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Active Imitation Learning of Hierarchical Policies

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

In this paper, we study the problem of imitation learning of hierarchical policies from demonstrations. The main difficulty in learning hierarchical policies by imitation is that the high level intention structure of the policy, which is often critical for understanding the demonstration, is unobserved. We formulate this problem as active learning of Probabilistic State-Dependent Grammars (PSDGs) from demonstrations. Given a set of expert demonstrations, our approach learns a hierarchical policy by actively selecting demonstrations and using queries to explicate their intentional structure at selected points. Our contributions include a new algorithm for imitation learning of hierarchical policies and principled heuristics for the selection of demonstrations and queries. Experimental results in five different domains exhibit successful learning using fewer queries than a variety of alternatives.

1 Introduction

Hierarchical structuring of policies and procedures is a common way to cope with the intractability of sequential decision making for both people and machines. Almost any complex task from cooking to arithmetic is taught by hierarchically decomposing it to simpler tasks. Indeed, hierarchical task structures are known to improve the efficiency of automated planning and [Nau *et al.*, 2003] reinforcement learning [Dietterich, 2000] in many complex domains.

In hierarchical imitation learning, we seek to imitate an agent who has an effective hierarchical policy for a sequential decision making domain. One reason to do this is the ubiquity and naturalness of hierarchical policies relative to the non-hierarchical (flat) policies. Another virtue of hierarchical policies is their transferability to other related domains. A third, perhaps more practical reason, is their utility in learning from humans and tutoring other humans. The tutoring application suggests that the policy needs to be constrained by the human vocabulary of task names for machines to communicate naturally with people [Byrne and Russon, 1998; Whiten *et al.*, 2006; Koechlin and Jubault, 2006].

Recent research in reinforcement learning and hierarchical planning has considered the problem of automatically discovering hierarchical structure by analyzing the dynamics and interactions of state variables during execution [Hengst, 2002; Jonsson and Barto, 2006], by finding bottlenecks in the state space [Mcgovern and Barto, 2001], and by doing a causal analysis based on known action models [Mehta *et al.*, 2008; Hogg *et al.*, 2009; Nejati *et al.*, 2006]. However, by being autonomous, these approaches have the problem of discovering unnatural hierarchies, which may be difficult to interpret and communicate to people.

In this paper, we study the problem of learning policies with hierarchical structure from demonstrations of a teacher whose policy is structured hierarchically, with natural applications to problems such as tutoring arithmetic, cooking, and furniture assembly. A key challenge in this problem is that the demonstrations do not reveal the hierarchical task structure of the teacher. Rather, only ground states and teacher actions are directly observable. This can lead to significant ambiguity in the demonstration data from the perspective of learning the teacher policy. For example, it might only be clear in hindsight why the teacher opened a drawer in a given state, and might require substantial reasoning.

Indeed, theory suggests that imitation learning of hierarchical policies in the form of decision lists is NP-Hard [Khardon, 1999]. One approach for learning hierarchical policies is to apply algorithms such as Expectation-Maximization (EM) which work by iteratively guessing the intentional structure and learning the policy Our extensive experiments in this direction showed that the hierarchies produced by these methods are often poor local optima and are typically inconsistent with the true intentional structure of the teacher. Further, even if the correct intention structure was learned, in order to communicate about that structure, a mapping to human-understandable terms would need to be learned. Thus, in this paper we follow a different learning approach where we directly work with a human to efficiently uncover the intention structure of demonstrations.

Our approach is motivated by the work of Khardon (1999) which showed that if demonstrations are annotated with the teacher's intention structure, then learning hierarchical policies becomes tractable. An analogous lesson from the tutoring literature is that learning can be made efficient if the examples illustrate one simple new concept at a time [VanLehn, 1987]. However, carefully ordering the examples to satisfy this constraint or fully annotating every trajectory with its in-

tention structure are too prohibitive. We address both of these problems by making the example selection and the task annotation *active* and *incremental*.

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 efficiently 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, the natural heuristic of selecting maximum entropy trajectory can perform poorly. We analyze this observation and develop a principled heuristic for selecting trajectories 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 five domains, including a domain relevant to arithmetic tutoring.

2 Problem Setup and Background

We consider a rewardless Markov Decision Process (MDP) defined by a a set of states S, a set of primitive actions A, and a transition function $\mathcal{P}(s'|s,a)$ which represents the distribution of next states s' given the state s and action a. 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 $T \bigcup A$, 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 leafs 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 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 (e).

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 π as a partial function from $T \times S$ to $T \bigcup A$ which is defined only for states that do not satisfy their termination conditions for any task and maps to one of its subtasks. A trajectory or a demonstration d of a policy π starting in state s is a sequence of alternating states and primitive actions generated by executing that policy from s starting with the 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.[1]

[1]When these termination symbols are not given, they can be inferred from the task definitions (which are known) and by asking additional queries to the user to rule out accidental task fulfillment. We ignore this complication in this paper.

More formally, a demonstration d of a policy π in a state s can be defined as follows, where ; is used for concatenation, $t_i\gamma$ represents a task stack with t_i as the topmost task and γ is the rest of the stack to be executed.

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

$d(s,\pi,t_i\gamma) = t_i' d(s,\pi,\gamma)$ if $t_i \in T$ and t_i terminates in s

$d(s,\pi,t_i\gamma) = d(s,\pi,\pi(s,t_i)t_i\gamma)$ if $t_i \in T$ and t_i does not terminate in s

$d(s,\pi,t_i\gamma) = t_i; s'; d(s',\gamma)$ where $t_i \in A$ and $s' \sim \mathcal{P}(.|s,a)$

In this paper we consider the problem of learning a deterministic hierarchical policy from a set of demonstrations. We assume that the different task definitions are known but the task hierarchy is not. The key problem in learning 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 was 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 subtasks begins. In this paper, 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 $\langle V, \Sigma, P, Z\rangle$, where V is a set of variables (represented by capital letters), Σ 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 $s, t_i \rightarrow \gamma$, where s is a state, t_i is a variable, and γ is a string over variables and terminal symbols. The above production rule is only applicable in state s, and reduces the variable t_i to the sequence of variables and terminals described by γ.

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 $V_a \in V$ and a terminal string $a; s$. Root is the start symbol. Further, it suffices to restrict the rules to the following 3 types:

1. $s, t_i \rightarrow t_j\ t_i$, where t_j is a non-primitive subtask of t_i
2. $s, t_i \rightarrow a_j; \delta(s, a_j)$, where a_j is a primitive action, and δ is the transition function
3. $s, t_i \rightarrow t_i'$ 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 $s, t_i \rightarrow \gamma$ 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. Table 1 represents the skeleton PSDG equivalent to the task hierarchy of the Taxi domain (see Figure 1(e)) without the probability distributions that correspond to a specific policy. It is easy to see that for any deterministic hierarchical policy, initial state and action dynamics, the corresponding deterministic PSDG, when started from the Root symbol and the same initial state, will derive the same distribution of demonstrations.

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 adapt the efficient algorithms developed for probabilistic CFGs such as the inside-outside algorithm to learn PSDGs [Lari and Young, 1990].[2] It also allows us to efficiently compute some information-theoretic heuristics which are needed to guide active learning.

Root → Get, Root	Up → up	Goto(L) → Left, Goto(L)
Root → Put, Root	Root → root′	Goto(L) → Right, Goto(L)
Root → Refuel, Root	Put → put′	Goto(L) → Up, Goto(L)
Dropoff → dropoff	Left → left	Goto(L) → Down, Goto(L)
Get → Goto(L), Get	Goto(L) → goto′	Refuel → refuel′
Get → Pickup, Get	Get → get′	Put → Goto(L), Put
Put → Dropoff, Put	Down → down	Right → right
Refuel → Goto(L), Refuel		Pickup → pickup
Refuel → Fillup, Refuel		Fillup → fillup

Table 1: An Example of PSDG skeleton of Taxi Domain without the state information. Primitive actions start with small letters and the task names start with capitals.

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 uninformed distribution of PSDGs by including every possible production in some MAXQ hierarchy with all production

[2]This view is not exactly correct in that a distribution over deterministic grammars requires us to first choose a deterministic grammar and use it to parse all parts of the trajectory, as opposed to making a different choice at each point. However, in practice this is perfectly adequate as we rarely encounter the same production distribution twice while parsing the same trajectory.

Algorithm 1: Learning Hierarchical Policies
1: Input: A set of trajectories Trjs, tasks T, actions A
2: Output: Hierarchical policies.

3: Initialize the PSDG
4: **loop**
5: trj = **SelectTrajectory**(Trjs)
6: Generate CYK table for trj
7: Sample N parse trees for trj from CYK table
8: $L \leftarrow \emptyset$ // a set of answered queries
9: **while** there is not a unique parse tree **do**
10: **for each** query $q \in Q$= $\{(s_i, task_j)_{i=1}^{|\text{trj}|}\}_{j=1}^{|T|}$
11: Compute H(Answer)
12: **end for**
13: newQuery = $argmax_{q \in Q}\{H(\text{Answer})\}$
14: answer = **AskQuery**(newQuery)
15: $L \leftarrow (L \bigcup \text{answer})$
16: **UpdateCYKTable**(L)
17: **if** there is a unique parse (L) for a sub-trajectory
18: **GenerateExamples**(CYK, L)
19: **run** TILDE
20: **end if**
21: **end while**
22: **end loop**

distributions initialized to be uniform (line 3). It then iterates over the following steps. First, the learner heuristically selects 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 through 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 sub-trajectory, 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 all segments of the trajectory. For each segment of the trajectory trj and indices i, j, and each variable B, it 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 $trj_1, ..., trj_{i-1}, B, trj_{j+1}, ..., trj_n$ 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 intermediate states, the α's and β's allow us to efficiently compute various other probabilities. For example, the probability that a trajectory (of length n) is generated starting from the Root task is $\alpha_{1:n}(\text{Root})$. See [Lari and Young, 1990] for details.

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. 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 *et al.*, 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 PSDG 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 PSDG.

It remains to specify the query selection heuristic. A standard heuristic inspired by generalized binary search is to select the query that maximizes the expected information gain (IG) of the answer. Prior work has shown that this heuristic achieves near optimal worst case performance [Dasgupta, 2004]. It is straightforward to show that this heuristic is equal to the entropy of the answer distribution, denoted by $H(\text{Answer})$. 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 hands 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 αs and βs 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 labeling [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 $TE(trj|M)$ can be efficiently computed using the CYK table without enumerating all parse trees [Hwa, 2004].

Tree entropy (TE) 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 (LNTE) as another heuristic [Hwa, 2004]. We also evaluated this heuristic in our experiments.

Our third heuristic, cost-normalized information (CNI), 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

5.1 Domains

We consider five domains for evaluating our active learning approach. The hierarchies for each domain are depicted in Figure 1.

Subtraction Domain: This domain is motivated by applications to automated tutoring systems, which raises the problem of allowing human teachers to easily specify hierarchical tasks without necessarily understanding the semantics and syntax of a formal hierarchy description language. There are two multi-digit numbers $A = A_n, ..., A_1$ and $B = B_m, ..., B_1$, where A > B, and the goal is to induce the standard "pencil and paper" hierarchical procedure for subtracting B from A (similar to [VanLehn, 1987]). This procedure is shown in Figure 1 and involves primitive actions such as crossing numbers out, overwriting numbers, and higher-level tasks such as borrowing. The state consists of the two digits being subtracted annotated by the result of all prior primitive actions. The trajectories used for training involve a number of digits ranging from 1 to 15 as in [VanLehn, 1987].

Kitchen Domain: This stochastic version of the kitchen domain originally described in [Natarajan *et al.*, 2011], models the process of preparing a meal involving a main dish and one or more side dishes. The primitive actions involve steps such as pouring, fetching ingredients, and baking. Each training trajectory specifies the main meal and side dishes and ends when they have been successfully prepared.

Assembly Domain: This domain models a furniture assembly problem, which is another domain where a computer assistant would need knowledge of a hierarchical assembly procedure. The agent's objective is to assemble cabinets, which contain one or more shelves. There are two types of shelves: shelves with a door and shelves with drawers. The agent starts by assembling shelves and connecting them together. For assembling each shelf, it first assembles a frame and a door or a drawer, then it connects the door or drawer to the frame. Each door, drawer and frame have different pieces, such as a handle, left side, right side, etc. Each frame has a connector for connecting doors or drawers. The actions are to add each side of the frame, add the door connector, add the door handle, connect shelves, etc.

Taxi Domain: This is a stochastic version of a domain that is a commonly used benchmark from hierarchical reinforcement learning, described in [Dietterich, 2000]. We use the more complex version that involves refueling the taxi. In each episode, the taxi starts in a randomly chosen location and with a randomly chosen amount of fuel (ranging from 5 to 12 units). The episode ends when the passenger is successfully transported. The stochastic dynamics dictates that with certain probability the taxi fails to move.

RTS Domain: This stochastic version of the RTS domain originally described in [Natarajan *et al.*, 2011], models a real-time strategy (RTS) game where an agent must move around a map collecting resources and attacking enemies.

5.2 Experiments

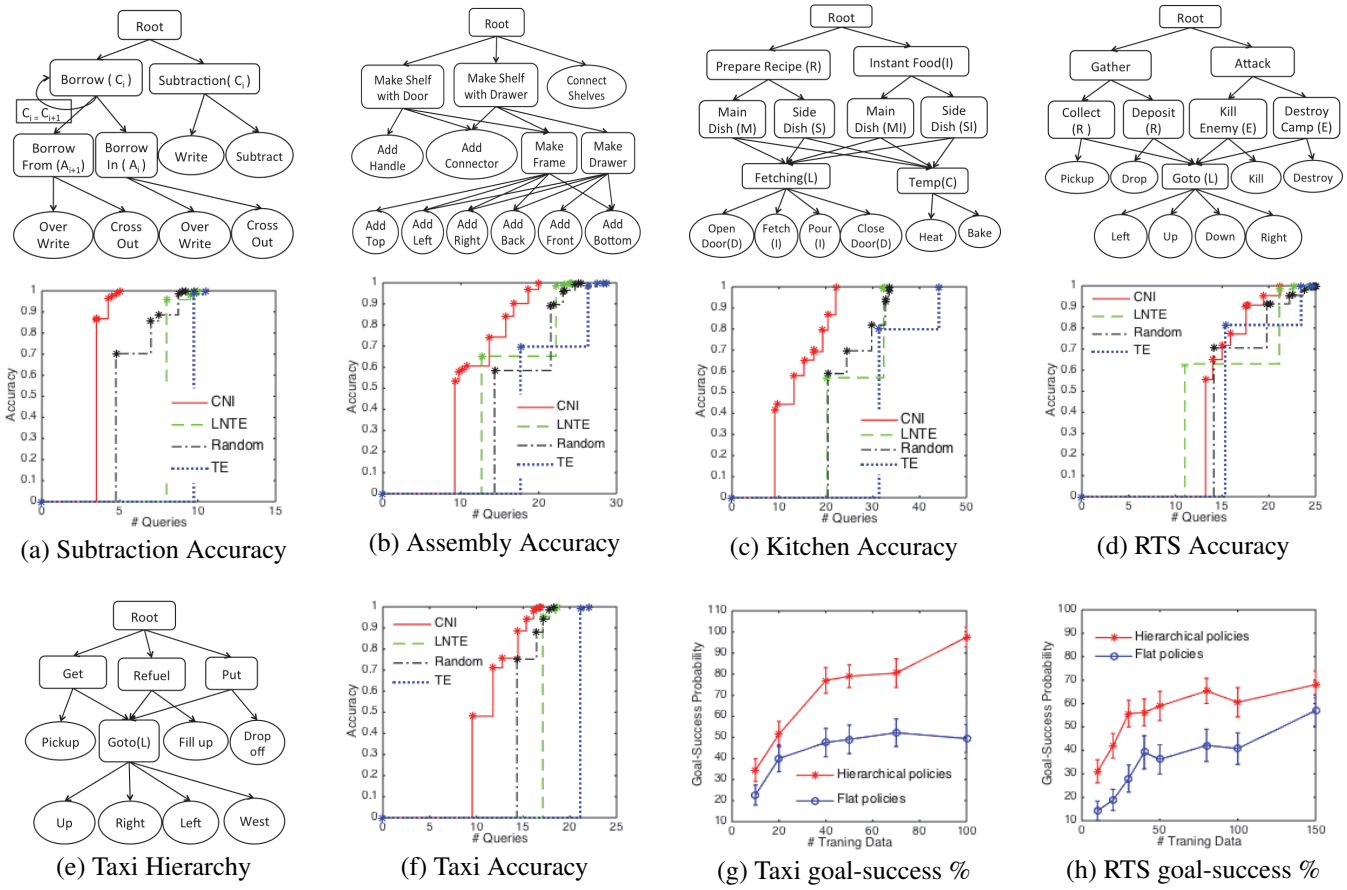
In each domain, we collected a set of training and test trajectories from a hand-coded hierarchical policy that follows the hierarchical structure shown in Figure 1. The trajectories are stored with the corresponding hierarchical intention structures which are used to answer queries. We use 10 training trajectories and 30 test trajectories in total. The results are averaged over 10 runs of the system with the same hierarchical policy but with a different set of randomly generated training and test trajectories. We used 70 sampled parse trees to select subgoal queries in each iteration.

Accuracy of Hierarchy Learning: Since one of the main motivations of this work is to learn the same hierarchical structure as used by the teacher, we first evaluate the accuracy of the hierarchies learned by our approach. In each domain, we compared the performance of our preferred heuristic, namely, maximizing *cost-normalized information (CNI)*, with three other trajectory selection strategies for learning hierarchical policies: 1) *Random* selection, 2) Maximizing *tree entropy (TE)*, and 3) Maximizing *length-normalized tree entropy (LNTE)*. Each heuristic was trained on the same hierarchical policy and the same set of training examples. Our performance metric is *accuracy*, which measures the fraction of times along the trajectory that the learned hierarchical policy agrees with the target policy about both primitive actions and the intention stack on test set.

Figures 1 (a)-(d), 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. Unlike 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 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 disambiguated using only 1 or 2 queries.

We first observe that selecting trajectories using TE performs poorly, even worse than random in all domains. The reason is that 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 and the Kitchen domains. However, the random strategy still performs better in the Taxi domain. This is because length does not necessarily reflect the number of queries needed to disambiguate a trajectory.

Our preferred heuristic, CNI, consistently performs better than all other heuristics in all domains with a possible exception of LNTE in RTS. These results indicate that CNI's approach of taking both the information gain and the cost of querying into account is effective. The shapes of the different curves suggests that CNI allows the learner to learn from more trajectories by spending fewer queries on each trajectory, achieving better performance after the same number



(a) Subtraction Accuracy (b) Assembly Accuracy (c) Kitchen Accuracy (d) RTS Accuracy

(e) Taxi Hierarchy (f) Taxi Accuracy (g) Taxi goal-success % (h) RTS goal-success %

Figure 1: First row and Figure (e) : The task hierarchies of Subtraction, Assembly, Kitchen, RTS and Taxi Domains. Second row and Figure (f): Comparison between different heuristics on trajectory selection strategies in five domains. The X-axis shows the number of queries and the Y-axis shows the accuracies of the task stacks at different points in the test trajectory. Figures (g) and (h): Comparison between flat and hierarchical policy learning in Taxi and RTS domains. The X-axis shows the number of training trajectories and the Y-axis shows the goal-success percentage.

queries as other heuristics.

Comparing to Flat Policy Learning: Here we compare the performance of actively learned hierarchical policies to traditional imitation learning of flat (non-hierarchical) policies. We compared the performance of our algorithm using CNI trajectory selection with the flat policy learning algorithm, whose policies are learned using TILDE [Blockeel and De Raedt, 1998]. Figures 1 (g) and (h) show goal-success percentage on a test set of problems vs. the number of the training trajectories for the Taxi and RTS domains. The results show that our active hierarchy learning approach is able to learn the tasks much more quickly in terms of number of training trajectories compared to the flat learning algorithm. Note, however, that this evaluation does not take into account the cost to the expert of answering the intention queries on the generated trajectories, which is zero for the flat learner. Thus, an overall comparison of the two approaches would need to weigh the relative cost of generating trajectories and answering queries. However, we see that the learning curve for the flat method is increasing very slowly for both problems, suggesting that for reasonable query costs the hierarchical learner would likely make the most efficient use of the expert's time.

6 Summary

We studied active learning of hierarchical policies in the form of PSDGs from trajectories generated by a hierarchical policy. We developed a novel two-level active learning framework, where the top level selects a trajectory and the lower level actively queries the teacher about the intention structure at selected points along the trajectory. We developed a new information-theoretically justified heuristic, cost-normalized information, for selecting trajectories, and employed Bayesian active learning for the lower-level query selection. Experimental results on five benchmark problems indicate that our approach compares better to a number of baselines in learning hierarchical policies in a query-efficient manner.

7 Acknowledgments

The authors thank the reviewers for their comments and acknowledge the support of ONR's ATL program N00014-11-1-0106.

References

[Baldridge and Osborne, 2003] J. Baldridge and M. Osborne. Active learning for hpsg parse selection. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 17–24, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[Blockeel and De Raedt, 1998] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artif. Intell.*, 101(1-2):285–297, May 1998.

[Byrne and Russon, 1998] R W Byrne and A E Russon. Learning by imitation: a hierarchical approach. *Behavioral and Brain Sciences*, 21:667–84; discussion 684–721, 1998.

[Dasgupta, 2004] Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In Lawrence K. Saul, Yair Weiss, and Lon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 337–344. MIT Press, Cambridge, MA, 2004.

[Dietterich, 2000] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res. (JAIR)*, 13:227–303, 2000.

[Golovin *et al.*, 2010] Daniel Golovin, Andreas Krause, and Debajyoti Ray. Near-optimal bayesian active learning with noisy observations. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 766–774, 2010.

[Hengst, 2002] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Maching Learning: Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250. Morgan Kaufmann, 2002.

[Hogg *et al.*, 2009] C. Hogg, U. Kuter, and H. Munoz-Avila. Learning hierarchical task networks for nondeterministic planning domains. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*, pages 1708–1714. AAAI Press, 2009.

[Hwa, 2004] R. Hwa. Sample selection for statistical parsing. *Computational Linguistics*, 30:253–276, 2004.

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

[Khardon, 1999] Roni Khardon. Learning to take actions. *Mach. Learn.*, 35(1):57–90, April 1999.

[Koechlin and Jubault, 2006] E. Koechlin and T. Jubault. Broca's area and the hierarchical organization of human behavior. *Neuron*, 50(6):963–974, June 2006.

[Lari and Young, 1990] K. Lari and S.J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[Mcgovern and Barto, 2001] A. Mcgovern and A.G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the eighteenth international conference on machine learning*, pages 361–368. Morgan Kaufmann, 2001.

[Mehta *et al.*, 2008] Neville Mehta, Soumya Ray, Prasad Tadepalli, and Thomas G. Dietterich. Automatic discovery and transfer of MAXQ hierarchies. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 648–655, 2008.

[Natarajan *et al.*, 2011] S. Natarajan, S. Joshi, P. Tadepalli, K. Kersting, and J. Shavlik. Imitation learning in relational domains: a functional-gradient boosting approach. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1414–1420. AAAI Press, 2011.

[Nau *et al.*, 2003] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F.Yaman. SHOP2: An HTN planning system. *J. Artif. Intell. Res. (JAIR)*, 20:379–404, 2003.

[Nejati *et al.*, 2006] N. Nejati, P. Langley, and T. Konik. Learning hierarchical task networks by observation. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 665–672. ACM Press, 2006.

[Pynadath and Wellman, 2000] David V. Pynadath and Michael P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, pages 507–514, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[Settles and Craven, 2008] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 1070–1079, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[VanLehn, 1987] Kurt VanLehn. Learning one subprocedure per lesson. *Artif. Intell.*, 31(1):1–40, 1987.

[Whiten *et al.*, 2006] A. Whiten, E. Flynn, K. Brown, and T. Lee. Imitation of hierarchical action structure by young children. *Dev Sci*, 9(6):574–82, 2006.