Learning Goal-Oriented Hierarchical Tasks from Situated Interactive Instruction

Shiwali Mohan and **John E. Laird**
Computer Science and Engineering Division, University of Michigan
2260 Hayward Street, Ann Arbor, Michigan 48109-2121
{shiwali,laird}@umich.edu

Abstract

Our research aims at building interactive robots and agents that can expand their knowledge by interacting with human users. In this paper, we focus on learning goal-oriented tasks from situated interactive instructions. Learning the structure of novel tasks and how to execute them is a challenging computational problem requiring the agent to acquire a variety of knowledge including goal definitions and hierarchical control information. We frame acquisition of novel tasks as an explanation-based learning (EBL) problem and propose an interactive learning variant of EBL for a robotic agent. We show that our approach can exploit information in situated instructions along with the domain knowledge to demonstrate fast generalization on several tasks. The knowledge acquired transfers across structurally similar tasks. Finally, we show that our approach seamlessly combines agent-driven exploration with instructions for mixed-initiative learning.

Introduction

With the recent advances in AI and robotics research, we can expect general-purpose intelligent agents to become pervasive in many aspects on human life. The complete space of tasks that the agents are expected to perform cannot be predicted ahead of time. This limits what can be preprogrammed in the agents. Therefore, a key design requirement is *taskability* - the agents should be able to learn diverse tasks from their experiences in their environments.

Several tasks such as those in domestic environments (Cakmak and Takayama 2013) can be characterized as achieving a goal state through execution of a sequence of actions. For example, for a task such as *set the table* the agent must instantiate a goal of achieving a set of spatial predicates (including *fork is on the right of the plate*) and execute a series of object manipulation actions to achieve it. Several of these tasks (such as *serve dinner*) can be hierarchically decomposed into child tasks (*make dinner*, *set the table*, etc). To learn these tasks, the agent must not only learn the goal definition and how to recognize goal achievement from sensory data, but also learn an execution policy defined over child tasks and actions that achieves the goal in the environment. Learning such tasks through self-directed experience alone can be slow, requiring repeated interactions with the environment. This has motivated research on incorporating human supervision and feedback in agent's learning.

Learning from demonstration (LfD) approaches (Argall et al. 2009) have recently gained prominence in the robotics community as a way of allowing naive human users to teach new tasks and actions to a robot. LfD approaches typically rely on traces obtained through teleoperation or kinesthetic training. Using regression-based methods, these traces can be used to directly approximate policy. Although LfD is useful in learning *primitive* action-control policies (such as for object manipulation), it is unsuitable for learning complex tasks such as those characterized earlier. LfD does not capture abstractions such as subtasks and the corresponding transition models that are required for reasoning about and learning goal-oriented tasks (Grollman and Jenkins 2010). It usually requires many examples in order to induce the intended hierarchical control structure (Allen et al. 2007). Moreover, the representations are task-specific and are not amenable to transfer to structurally similar tasks (Chao, Cakmak, and Thomaz 2011).

In this paper, we explore an alternative and complimentary approach for interactively learning new tasks based on explanation-based learning (EBL: DeJong and Mooney 1986). In our approach - situated interactive instruction - the human expert and the robotic agent are simultaneously embedded in a real-word domain. The human can observe the robot's workspace and how it manipulates its environment. Through mixed-initiative linguistic interactions, the instructor guides the agent to execute a novel task. This guided execution serves as a basis for acquiring task knowledge using EBL. EBL methods offer several advantages over traditional LfD. EBL methods work over relational representations that not only allow the agent to reason about the structure of the goal and how to achieve it but are also useful in linguistic communication. These methods are knowledge-intensive and exploit the agent's domain knowledge to deduce generally applicable knowledge from a relatively small number of examples. These properties make EBL an attractive alternative to study for interactive task learning.

The contributions of this paper are as follows. We propose a novel interactive learning paradigm for online learning of goal-oriented hierarchical tasks from linguistic task instructions. We demonstrate that the task representations and the proposed paradigm is *comprehensive* and can be

used to learn multiple tasks. The causal analysis in EBL can be combined with information about the structure of interactions for fast *generalization* to the entire space of task variation from only a few, highly specific examples. We further demonstrate that knowledge learned for a task is *transferable* and is useful in learning a different but structurally similar task. Finally, we show that the proposed learning paradigm is *mixed-initiative* and elegantly integrates agent-driven exploration with instructional guidance distributing the onus of learning between the agent and the instructor.

Background

Our approach is implemented in Rosie (Mohan et al. 2012) a generally instructable agent developed using the Soar cognitive architecture (Laird 2012). It has the mechanisms necessary for perceiving and acting in the real world along with appropriate long-term memories and learning algorithms such as chunking that are useful for implementing our task learning paradigm.

Rosie is embodied as a robotic arm (in Figure 1, left) that manipulates small foam blocks on a table-top workspace that simulates a kitchen. The workspace contains four locations: *pantry*, *garbage*, *table*, and *stove* that have associated simulated functions. For example, when a *stove* is turned on, it changes the simulated state of an object on it to *cooked*. To act in the world, the agent sends primitive commands to the controller that are translated to closed-loop policies. The commands include actions for object manipulation: $point\text{-}to(o)$, $pick\text{-}up(o)$, and $put\text{-}down(x, y)$; and simulated location operation: $open(l)$, $close(l)$, $turn\text{-}on(stove)$, and $turn\text{-}off(stove)$. Human instructors can interact with the agent through a chat interface. Rosie can learn *basic* concepts such as perceptual attributes of objects (color: *red*, size: *large*) and spatial relationships (*on the shelf*, *to the right of*) through instruction. This paper studies how the basic concepts can be composed to learn novel goal-oriented tasks.

The instruction cycle implemented in Rosie begins with an utterance from the instructor. The utterance is grounded to internal representations used by Rosie for action and reasoning (Mohan, Mininger, and Laird 2013). From the utterance content and the current interaction state, Rosie determines its next objective. It can be to learn a new concept (in response to *this is red*, take an action in the environment (*open the pantry*), or respond to the instructor's questions (*what is to the left of the blue object?*). If Rosie lacks knowledge either to ground the utterance or to pursue its objective, it asks questions such as *I do not know red. Can you show some examples?* The instructor can then present examples from the environment to teach the unknown concept.

Hierarchical Task Representation

Rosie's beliefs about its state $s \in S$ are encoded as a set of relational predicates P defined over the set of objects O. They include unary predicates that describe the state of an object, such as $open(pantry)$, and binary predicates that describe spatial relationships between two objects, such as $on(o1, o2)$). The predicates are represented symbolically for reasoning and task learning, however, they eventually ground out to functional and continuous perceptual state of the environment. We assume a pre-encoded set of primitive behaviors or actions A and corresponding models that predict the effects of their execution. To learn a task $t \in T$ the following concepts are acquired. We use the $store(o)$ task as an example to explain them. The $store(o)$ task requires the agent to place o in the open *pantry* and close it.

- **parameters**, a set of objects ($O^t \subseteq O$) that are involved in the application of the task t. Parameters can be explicit or implicit. Explicit parameters are provided as arguments in the task command. In *store the red cylinder*, the object described by *the red cylinder* is an explicit parameter. Implicit parameters are inherent to the task. The location *pantry* is an implicit parameter of the task *store* and is not specified in the task command. The agent must learn to associate the task with its implicit parameters.
- **availability conditions**, a set of state predicates ($P_A^t \subseteq P$) that have to be true conjunctively for the task t to be available for application.
- **child tasks**, $C^t \subseteq A \cup T$, a set of child tasks and actions for the task t. A child task of $store(o)$ is *move* which establishes a spatial relationship in between o and the *pantry*.
- **policy**, $\pi^t(s) : S \rightarrow C^t$ that selects child tasks and actions based on the state of the environment.
- **termination conditions**, a set of state predicates ($P_T^t \subseteq P$) that if true signify that the task goal has been achieved and the policy should be terminated.
- **model**, $M^t(s) : S \rightarrow S$ that predicts the future state s' after the task t is executed in the current state s. It is represented as a set of rules that encode how predicates in s transition to predicates in s' under task t.

Although our representation bears similarities with HTNs (Erol, Hendler, and Nau 1992) and MAXQ (Dietterich 2000) hierarchies, there are important differences. Control knowledge in HTNs is encoded as conditional actions whereas in our representation it is encoded as a state-sensitive policy. Encoding it as a policy allows the agent to robustly operate in dynamic environments where the state may change independent of agent's actions without extra planning. In contrast to our representation, typical MAXQ hierarchies operate in propositional domains.

Task Learning with Interactive EBL

To learn a novel task using EBL the agent must have access to instances of the task concepts identified in the previous sections and a domain theory to explain how the instances are valid examples of the task concepts. The instances are acquired interactively by prompting the instructor and situating the interactions in the perceptible state. The action and task models (pre-encoded and learned) serve as the domain theory. Our design tightly couples execution of the task with exploratory and instruction-driven learning. If the agent experiences any failures during task execution, it asks questions to resolve them or performs an exploration using its domain-knowledge. Below we describe how known tasks are executed first and then how novel tasks are acquired.

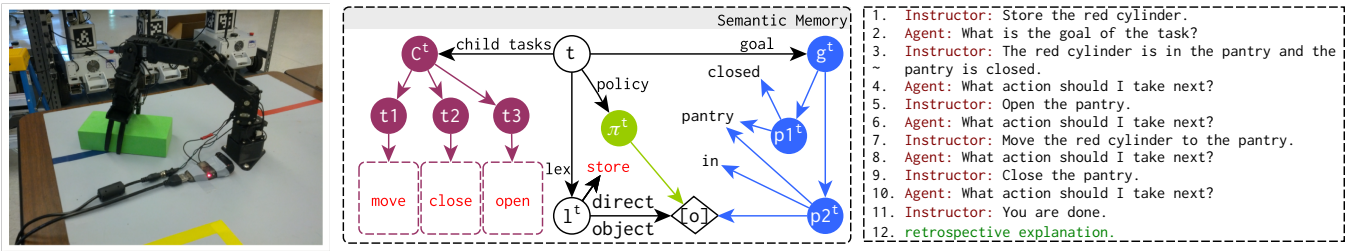

Figure 1: (left) the robotic arm (center) semantic representation of *store* (right) interactions for instance acquisition

Executing a known task On receiving a task command *store the red cylinder*, the agent generates a corresponding task instantiation by grounding it in its knowledge and perceptible state. Let this task instantiation be $t(o_1)$ where o_1 is an object that satisfies the linguistic description *the red cylinder*. The agent, then, begins to execute $t(o_1)$ (as described in Algorithm 1). The agent's relational semantic memory stores the general goal and policy description for a task along with the constraints on how they are instantiated. The agent can access these by querying its memory using the $\textbf{retrieve}(type?, t)$ function which returns the general definitions of the goal, policy, or subtasks of task t.

The graph (Figure 1) beginning at node t represents the task *store* and associates it with a goal definition (node g^t), its child tasks (node c^t), and the policy description (node π^t). The subgraph at node l^t encodes linguistic information for the verb *store*. Node $[o]$ represents a slot that can be filled by any object that satisfies the description in the direct-object of the verb *store*. The instantiation of the goal predicate in is constrained to the object in slot $[o]$ (*explicit* parameter) and $pantry$ (*implicit* parameter). The subgraph at node π^t encodes that the policy is parameterized by $[o]$. The graph also encodes the association between the t and its child tasks.

On retrieving the goal and policy descriptions, the agent instantiates them with the task parameters to generate a desired state and a policy. The policy is represented as rules in the agent's rule-based procedural memory. The availability conditions of the child actions/tasks determine if they can be executed in a state ($avail(s, c)$). Given the state and the set of actions/tasks that can be executed in that state, the policy suggests a child action/task based on the current state, which is then applied (Algorithm 1: line 11). The task execution is terminated when the desired state is reached (Algorithm 1: line 10).

Algorithm 1 Executing a task.

1: **procedure execute**(state s, task t)
2: $g \leftarrow$ **retrieve**(type *goal*, task t)
3: **if exists**(g) **then**
4: $d \leftarrow$ **instantiate**($g, \{o_1, o_2, ...\}$)
5: **else goal-query**(t)
6: $C^t \leftarrow$ **retrieve**(type *children*,task t)
7: $C \leftarrow$ **instantiate**($c^t, \{o_1, o_2, ...\}$)
8: $\pi^t \leftarrow$ **retrieve**(type *policy*, task t)
9: $\pi \leftarrow$ **instantiate**($\pi^t, \{o_1, o_2, ...\}$)
10: **while** $((d \subseteq s) \neq true)$ **do**
11: **if** $(\pi(s) \rightarrow c \in C)$ & $avail(s, c)$ **then execute**(s, c)
12: **else**
13: **if search**$(s, d, C, \pi) = false$ **then child-query**(t)

Learning a new task On being asked to execute a novel task, the agent first extracts the lexical structure used in describing the task. From the command *store the red cylinder*, it extracts the general syntactical knowledge `verb:store direct-object:[o]`. This is stored in the semantic memory (subgraph at l^t in Figure 1). This declarative representation is incrementally augmented as the agent learns about the concepts relevant to the $store$ task. The learning process is described below.

Termination conditions: If the agent is unsuccessful in retrieving a goal definition while executing a task, it asks a goal question (Figure 1:line 2, Algorithm 1: line 5). The instructor replies with a description of the goal state (Figure 1: line 3). The declarative definition of the goal is stored in the agent's semantic memory with the appropriate constraints to the lexical structure of task description. To operationalize the declarative definition, the agent performs a hypothetical explanation. The agent imagines the state s_h that will result on executing the task in the current state by appending the grounded goal predicates to the current state description. The agent then uses a set of domain-general rules to verify that every predicate in the goal $g(o_1, o_2, ...)$ is true in s_h and that s_h is a valid instance of the task's desired state. Chunking complies this verification into a termination rule that is of the form - `if` $g(o_1, o_2, ...) \subseteq s$ `then` $achieved\text{-}goal(s)$. The termination rule for the *store* task ($t(o)$) is - `if` $(in(o, pantry) \wedge closed(pantry) \wedge executing(t(o))) \subseteq s$ `then` $achieved\text{-}goal(s)$. Note that this rule is general and can match several states even though it is learned from a specific hypothesized state. This is because the rule only tests for a subset of predicates that are required to describe the complete environment state.

Policy: The task policy is learned during two different stages. A part of the policy is learned during execution of a task through immediate explanation. If the policy does not suggest any child task/action at state s (Algorithm 1: line 12), the agent performs a recursive iterative-deepening search for the desired state to depth K (Algorithm 2). For exploratory search in state s, the agent iterates through all available actions by applying their models and analyzing the resulting states (Algorithm 2: line 8). During this search, if an action is found to be useful in making progress towards the desired state, chunking compiles a policy rule. It collects all the predicates that were tested to apply the models and the

Algorithm 2 Exploring the action space

1: **procedure search**(state s, desired d, actions C, policy π)
2: **for** ($k = 0$, $k < K$, $k++$) **do**
3: $a \leftarrow \textbf{explore}(s, d, k)$
4: **if** $a \neq false$ **then return** $true$
5: **return** $false$
6: **procedure explore**(state s, desired d, depth $n = k$)
7: **if** $n = 0$ **then return** $false$
8: **for each** $(a | a \in C \wedge avail(s, a))$ **do**
9: $s' \leftarrow M^a(s)$
10: **while** $(\pi(s') \rightarrow c \in C)$ & $avail(s', c)$ **do** $s' \leftarrow M^c(s')$
11: **if** $(d \subseteq s')$ **then**
12: $F \leftarrow$ **collect-predicates**
13: **add**$(\pi(s | F \subseteq s) \rightarrow a)$ **return** $true$
14: **else**
15: **if** $(\textbf{explore}(s', d, n-1) \neq false)$ **then goto** 10
16: **return** $false$

termination rule (Algorithm 2: line 8). The left-hand side of the policy rule contains these predicates and the right-hand side contains the action. This rule is added to the task policy.

If the desired state is not found at depth K or earlier, the agent abandons exploration and asks the instructor for an action that it should take. The instructor replies with a child task/action command (Figure 1: line 5) which is grounded and executed in the environment. Such exploration and interactions continue until the agent achieves the goal state in the environment. The agent's *episodic memory* automatically stores the state (environmental and interaction) of the agent for each step. This history is available for later inspection and learning the remainder of the policy.

After the instructor indicates or the agent deduces that the instructed task execution is over and the goal state is achieved, the agent attempts to learn the remainder of the policy. It is learned through simulating task execution and retrospectively *explaining* the instructions (in Algorithm 3). To simulate the task execution, the agent queries its episodic memory for the environmental state s when it asked for instruction. It begins by instantiating the task goal definitions in state s to generate the desired state d. Next, it retrieves and instantiates all the child task representations that were used in the instructed execution and exploration of the task. The agent then recursively analyzes why the next instructed action a (obtained by looking up episodic memory) is useful in approaching the desired goal state. In each recursion, the agent applies the model of the instructed action M^a on the state s to generate the subsequent state s' (Algorithm 3: line 8). If the agent has learned policy for s' through its exploration, it is applied (Algorithm 3: line 9). If the goal is achieved in any subsequent state s' desired, a policy rule is learned (Algorithm 3: line 11,12). If not, it recurses further (Algorithm 3: lines 14, 15) by looking up episodic memory for the next instructed action. A policy rule of the task *store* $(t(o1))$ compiled through chunking is of the form - `if` $(in(o, pantry) \wedge open(pantry) \wedge executing(t(o))) \subseteq s$ `then` $select(close(pantry))$. Similar to the termination rule, the policy rules are general in that they apply to a set of states because they test only a subset of predicates necessary to describe an environmental state.

Availability Conditions: The availability conditions for

Algorithm 3 Explaining instructions retrospectively

1: **procedure explain-instructions**(time n, task t)
2: $s \leftarrow \textbf{query}(n)$
3: $d \leftarrow \textbf{instantiate}(\textbf{retrieve}(goal, t), \{o_1, o_2, ...\})$
4: $C \leftarrow \textbf{instantiate}(\textbf{retrieve}(children, t), \{o_1, o_2, ...\})$
5: $\pi \leftarrow \textbf{instantiate}(\textbf{retrieve}(policy, t), \{o_1, o_2, ...\})$
6: $a \leftarrow$ **first-action**
7: **procedure exploit**(s, a)
8: $s' \leftarrow M^a(s)$
9: **while** $(\pi(s') \rightarrow c \in C)$ & $avail(s', c)$ **do** $s' \leftarrow M^c(s')$
10: **if** $(d \subseteq s')$ **then**
11: $F \leftarrow$ **collect-predicates**
12: **add**$(\pi(s | F \subseteq s) \rightarrow a)$ **return**
13: **else**
14: **if** $a' \leftarrow$ **next-action then**
15: **if** $(\textbf{exploit}(s', a') \neq false)$ **then goto:** 8
16: **return** $false$

a task t_i (e.g. *place*) are learned while learning a policy of a parent task t_j (e.g. *store*). When the explanation/exploration of t_i is successful, an availability rule for t_i is compiled through chunking along with a policy rule for t_j. The rules take the form - `if` $(F^{t_i} \subseteq s \wedge executing(t_j)$ `then` $available(s, t_i)$. For example the *place* task $(t_1(o, in, pantry))$ is available during the execution of the task *store* $(t(o))$: `if` $(-in(o, pantry) \wedge open(pantry) \wedge executing(t(o))) \subseteq s$ `then` $avail(s, place(o, in, pantry))$.

Model: The model for a task t is a critical component of the domain theory for learning a parent task. In our formulation, an explicit representation of the model is not essential. The effects of task t in a state s can be predicted by simulating the execution of t in s by applying its policy. This is sufficient for learning a novel parent task.

Evaluation

We are interested in four desiderata of intelligent, interactive, taskable agents: *comprehensiveness* - the agent is capable of learning a diverse variety of tasks; *generality* - the agent should exploit its reasoning capabilities to generalize quickly to the complete space of task variation; *transfer*: the knowledge acquired in one task can be reused in a different but structurally similar task; and *mixed-initiative*: the onus of learning should be distributed between the learner and the instructor. Below we qualitatively and empirically analyze the proposed learning paradigm along these desiderata. The experiments were conducted on a simulated environment that is faithful to the continuous sensor data and control policies the robot works with. We used the simulator to generate the reported results because the evaluation space is large and generating results with the real robot is extremely time consuming. A representative sample of the scenarios described in the paper were successfully run on the real robotic arm, achieving results that are consistent with those reported in the paper.

Comprehensiveness The agent can learn *achievement* tasks, where the agent acts to achieve a composition of goal predicates in the environment. A summary of the seven tasks taught to the agent is in Table 1. The goals are composed of *state* and *spatial* predicates that eventually ground out to the

Explicit parameters	*Primitive policy space*	*Goal description*
place (obj,rel,loc)	pick-up, put-down	rel(obj,loc)
move (obj,loc)	pick-up, put-down	on(obj,loc)
discard (obj)	pick-up, put-down	in(obj,garbage)
store (obj)	open, pick-up, put-down, close	in(obj,pantry) closed(pantry)
cook (obj)	activate, pick-up, put-down, stop, wait-until	in(obj,stove) cooked(obj)
stack (obj1, obj2, obj3)	pick-up, put-down	on(obj1,obj2) on(obj2,obj3)
serve (obj)	activate, pick-up, put-down, stop, wait-until	in(obj,stove) cooked(obj)

Table 1: Learned tasks, parameters, policy space, and goals

functional and continuous state of the environment. The policy space for all tasks eventually grounds out to a set of primitive actions (including closed loop motor-control for object manipulation) that can be executed in the environment and an internal *wait-until* action that polls the environment for a state change (such as $cooked(o)$). The policy learned for task execution is flat if the instructions consist only of primitive actions or hierarchical if the instructions decompose the task into child tasks. For example, $serve(o)$ can be decomposed into $cook(o)$ and $place(o, on, table)$. $cook(o)$ can be further decomposed into its constituent tasks and actions. If the agent does not know the child task, it attempts to learn it before learning the parent task. The agent can learn tasks with *explicit* and *implicit* parameters. The learned *place* task is represented such that it can be used to achieve any known arbitrary spatial relationship between two objects and takes three arguments. These arguments explicitly identify all the information required to perform this task. A similar task *move* is defined for a specific spatial relationship ON between two object arguments. The relationship ON is an implicit parameter and is inherent to the *move* task. Similarly, other tasks such *store*, *discard*, *cook* have implicit location and relationship parameters. The hierarchical task representation and the proposed learning paradigm allow for the acquisition of achievement tasks that vary in their goal description, policy structure, and parameters.

Generality There are three reasons for generalization in our approach. The first reason is that the positional information in the training instances is abstracted away in the relational representations used for reasoning and learning. Prior work (Mohan et al. 2012) provides an empirical analysis of this. The second reason - *predicate selection* - is typical of EBL methods. The causal inference identifies the minimal and sufficient set of predicates from the complete state description that are required to apply the domain theory during learning. Consequently, the rules representing the availability and termination conditions and the policy apply in multiple states even though they have been learned from specific examples. The final reason is *variablization* of objects and relations in task representations. We use the structure of interactions to inform which objects can be variablized away. The objects and relations that are used in the task command are variablized and the objects and relations that do not occur in the task command but are used to describe the goal and to guide execution are stored with the task representation as constants. A more aggressive strategy suggests that all objects can be variablized. This strategy is over-general and tasks that have implicit parameters cannot be learned correctly and therefore, it was not considered.

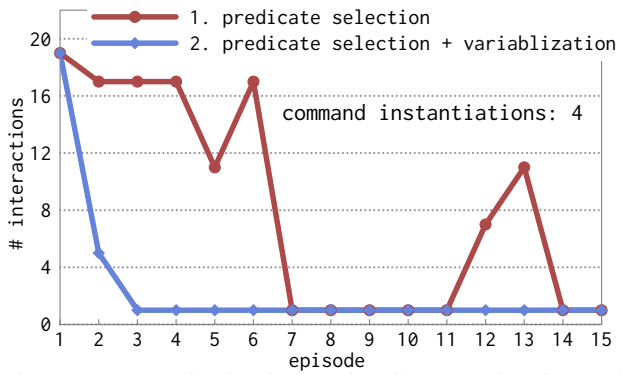

Figure 2: Generalization in learning flat execution for *cook*.

Our experiment scenario consisted of four objects, four known spatial relations, and four locations. We conducted separate trials for learning flat execution for five tasks *place*, *move*, *discard*, *store* and *cook*. A task trial consists of a series of episodes in each of which the agent is asked to execute the task with randomly generated parameters. Each episode begins in an initial state obtained by assigning random states to locations $open/close(pantry)$, the arm $(hold/-holds(o))$, and arbitrarily placed objects on the workspace. The environment can be in 16 initial states and the objects can be in infinitely many locations. If the agent asked a child query during a training episode, it is given the relevant primitive action. An episode terminates when the agent successfully executes the task in the environment. The exploration depth was set to 0 of this experiment.

A sample of the results generated from the experiment is shown in Figure 2. The graph shows the median number of interactions that occurred in every episode for executing *cook* over five trials for two variations of the learning algorithm. The first variation (red) only generalized through predicate selection. The second variation (blue) also variablized objects based on the structure of interactions. As expected, the majority of interactions occur during first few episodes during which the agent is trying to learn the task from interactions. The interactions drop as the trial progresses and the agent learns the task can execute it without any instructions. The number of interactions for the second variation (blue) drop sharply after only a few episodes. This establishes that even though it has been trained on only a very small sample of the possible initial states (16) and task-command instantiations (4 for *cook*), it is able to learn representations that generalize to the complete space of command instantiations and initial states. The first variation (red) cannot generalize to the complete space of command instantiations as quickly but does generalize to the complete space of initial states (the curve at episodes 11-14 is an artifact of randomization). Both variations are insensitive to the specific positional information of the objects in the training instances as both use relational representations and both learn the correct policy. The performance on other tasks was similar to what is shown in Figure 2. Even for *place*, which can have

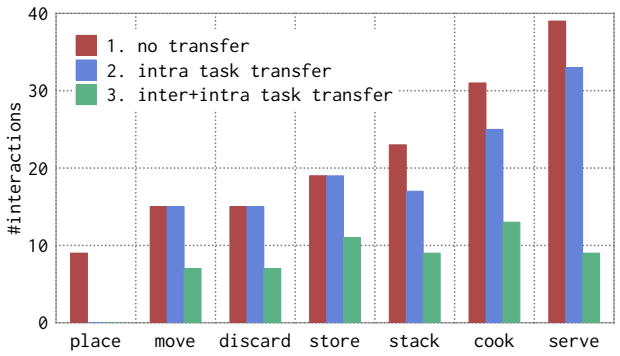

Figure 3: Instruction assisted transfer during learning.

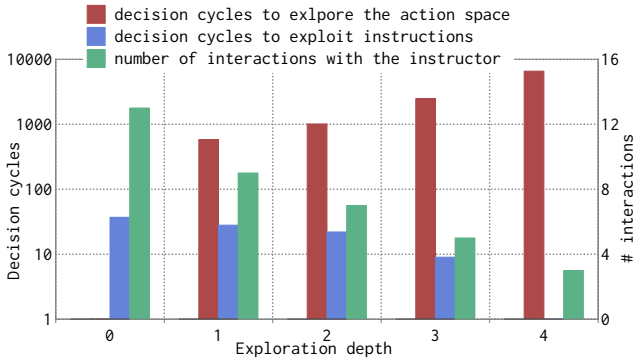

Figure 4: Learning *store* at different depths of exploration.

128 different instantiations, the second variation tapered to 1 interaction per episode after 6 episodes. The data demonstrates that both predicate selection and variablization contribute towards general learning.

Transfer Tasks in a domain may have similar structure, common sub-goals, and overlapping policy. An ideal learner should be able to exploit the inter-task similarities while learning a new task. For an interactive learner, the structure of interactions can play an important role in transfer of knowledge between tasks. Consider the tasks *store* and *cook* in our domain. Both of these tasks involve establishing a specific spatial relationship (in) between their parameters which is established by a policy over *pick-up* and *put-down*. This policy can be compiled in a *place* child task through instructions and can be used to teach both *store* and *cook*, resulting in savings in interactions. This compilation is also useful in intra-task transfer in tasks such as *stack* that involves multiple *pick-up* and *put-down* compositions.

Figure 3 shows how prior learning influences learning a new task. The agent was taught seven tasks sequentially with three variations of the learning algorithm. The tasks were taught hierarchically by decomposing them into child tasks through instructions. In hierarchical learning, if the child tasks are known to the agent, they are executed. If not, the agent learns the child task before learning the parent task. In the first variation (red), the agent's knowledge was reset after teaching each task (no inter-task transfer) and the learning of child tasks was turned off (no intra-task transfer). For the second variation (blue), learning of child tasks was turned on (intra-task transfer). Finally, for the third variation (green) the knowledge acquired for each task was maintained allowing for inter- and intra- task transfer.

The *place* task is a policy defined over *pick-up* and *put-down* and has no child tasks. The *move*, *discard*, *store* tasks require a single *pick-up* & *put-down* composition along with other actions. If the *place* task is known before the agent begins to learn these tasks, there are some savings in the number of interactions. There is no possibility of intra-task transfer. The task *stack* requires multiple executions of *pick-up* & *put-down* composition. Therefore, intra-task transfer is useful and saves some interactions. If *place* task is known prior to learning *stack*, further savings are achieved as the knowledge transfers from *place* to *stack*. Similar savings are observed in learning the *cook* and *serve* tasks when their child tasks are already known. The ability to transfer knowledge across tasks results in efficient learning as it significantly reduces the number of interactions required to learn a task.

Mixed-initiative learning Often in human controlled interactive learning such as learning by demonstration, the complete onus of learning is on the human user. The human has to provide good demonstration traces that will result in learning at appropriate levels on generality. However, an ideal interactive learner must be active and play a useful role in knowledge acquisition. It should not completely rely on the human instructor, but instead use its knowledge of the domain to explore available options. Such active learning biases acquisition towards knowledge that is required by the agent for task performance and reduces the load on the instructor. In our approach, the agent uses its action/task models to explore the space of available actions to depth K to find a path to the desired state. Chunking complies deliberate exploration into a policy. If the agent is unable to discover a path to the desired state, it abandons exploration and asks for guidance. During retrospection, the learning from agent-driven exploration and instruction-driven execution is integrated into a comprehensive policy.

Figure 4 shows the performance of the learning algorithm at different exploration depths for learning the store task in terms of the time spent in explorations (red bars as measured in Soar's decision cycles), the time spent in retroactive explanation (blue bars - decision cycles) and the number of interactions with the instructor (green bars). At depth 0 the agent does not perform any exploration and relies completely on instructions. The entire time used for learning (blue bars) is spent on retrospectively explaining the instructions. As the exploration depth parameter increases, the agent is more self-reliant, requiring fewer interactions, but spending more time exploring. At depth 4, it discovers the solution and does not ask any child-queries. Thus, the agent can solve simple problems on its own, only asking for help for difficult problems. In future, the approach will be extended so that the agent can explicitly reason about the expected cost of exploration and the cost of asking for help to influence its decision about beginning an interaction.

Related Work

Although there has been substantial research on endowing robotic agents with sophisticated multi-modal interac-

tion capabilities, few efforts have addressed the problem of designing taskable agents that can learn complex tasks via human-agent interaction. Most of the approaches frame task learning as learning compositions of known primitive actions. A few prior approaches have studied learning tasks from demonstration. Bentivegna, Atkeson, and Gordon (2004) proposed a framework that learns subgoals from segmented observed data. The system can use its experience to optimize its policy of selecting subgoals. Grollman and Jenkins (2010) formulate the problem of task acquisition as inferring a finite-state machine from segmented observations of a demonstrator performing the task. While the former is a batch learning system, the latter focuses on learning interactively during performance. In contrast to these efforts that rely on observational data alone, our research focuses on interactive linguistic input that naturally segments task execution. Our approach allows the agent to ask questions that are specific to its own knowledge state, resulting in fast learning and generalization.

Prior work on task learning from spoken dialog (Rybski et al. 2007; Cantrell et al. 2012; Meriçli et al. 2014) addresses learning procedures. Through linguistic constructions (obtained from imposing additional syntactic constraints), these agents can be programmed to execute procedures defined over their primitive, pre-encoded actions. An example instruction is - *when I say deliver message, If Person1 is present, Give message to Person1....* (Rybski et al. 2007). Using domain-general mechanisms, this instruction is translated into procedures that are added to the agents repertoire. A key departure of our work from these approaches is the task representation. Whereas the prior work represents tasks as procedures, we employ a hierarchical action-oriented representation that includes state-sensitive execution policies. Along with robust behavior, our representation ensures easy composability of tasks and knowledge transfer. Furthermore, in the prior approaches, the onus of learning is on the instructor who must explicitly identify the preconditions, termination criterion, and the procedure of doing a task. Our approach distributes the onus of learning between the human instructor and the agent - the instructor presents examples of correct behavior and the agent deduces the preconditions, termination criterion, and the execution policy.

Allen et al. (2007) demonstrate a virtual learning agent that learns executable task models from a single collaborative session of demonstration, learning, and dialog. The human teacher provides a set of tutorial instructions accompanied with related demonstrations in a shared environment, from which the agent acquires task models. The initiative of learning is on the human user. However, the agent controls certain aspects of its learning by making generalizations about certain tasks without requiring the human to provide numerous examples. Although our research shares motivations with this work, our learning paradigm is novel in the use of EBL methods to learn composable representations. The prior work does not make any claims about generality and transferability of their representations and learning.

The learning paradigm proposed in this paper makes significant contributions beyond the earlier work by Huffman and Laird(1995). The prior work lacked the capability of full retrospection and learned a task incrementally, needing n instructed executions for learning a task that requires n actions for achieving the goal. Full retrospection allows the agent to learn from fewer instructed executions. Our paradigm also integrates learning from agent-driven exploration with instructed execution which the prior work was unable to do. The earlier work neither identified the space in which tasks vary (including implicit/explicit parameters) nor identified the representations sufficient for representing a variety of tasks. The analyses of the generality and transferability is also novel to our work.

Discussion and Future Work

In this paper, we described an approach to interactively learn new tasks that has several properties desirable in an intelligent agent including *comprehensiveness*, *generality*, *transferability*, and *mixed-initiative*. However, it is limited in various ways. We have focused exclusively on learning achievement tasks. Although most tasks (Cakmak and Takayama 2013) can be characterized as such, maintenance (*keep the kitchen clean*) and performance (*patrol*) tasks are common in human environments. In future, we will investigate how the representations and the learning paradigm proposed in this paper can be extended to support learning of maintenance and performance tasks. Another limitation is that goals can only be defined as a conjunction of state predicates. Future efforts will address acquisition of wider variety of tasks and goals.

Arguably, the adopted generalization strategy is aggressive. Consider the variations, *store the eggs* and *store the rice*. The location where the argument objects are placed at is sensitive to their categorization. Our approach cannot learn a useful representation for such task variation. Future research will study the modifications required in the task representation and the learning paradigm to support such variation. Another issue is our assumption that the instructor does not make any instruction errors. However, this may not hold for instructions for complex tasks in partially observable domains and novice instructors. We want to investigate corrective instruction and associated learning strategies so that the agent can recover from incorrectly acquired knowledge. We also want to evaluate learning tasks from situated instructions in human-robot interaction contexts and study the mechanisms that make teaching tasks easier from the humans' perspective. Finally, we are interested in a comprehensive account of hierarchical task learning and will make efforts to integrate of our approach with learning primitive actions from demonstration.

Acknowledgment

The authors acknowledge the funding support of the Office of Naval Research under grant number N00014-08-1-0099. The authors thank Edwin Olson and the APRIL lab at the University of Michigan, Ann Arbor for the design and implementation of the robot including its perception and actuation algorithms. The authors also appreciate the insightful comments and suggestions made by the AAAI reviewers.

References

Allen, J.; Chambers, N.; Ferguson, G.; Galescu, L.; Jung, H.; Swift, M.; and Taysom, W. 2007. Plow: A collaborative task learning agent. In *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*.

Argall, B.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A Survey of Robot Learning from Demonstration. *Robotics and Autonomous Systems*.

Bentivegna, D.; Atkeson, C.; and Gordon, C. 2004. Learning Tasks from Observation and Practice. *Robotics and Autonomous Systems* 47:163–169.

Cakmak, M., and Takayama, L. 2013. Towards a Comprehensive Chore List for Domestic Robots. In *Proceedings of the Eighth International Conference on Human-Robot Interaction*.

Cantrell, R.; Talamadupula, K.; Schermerhorn, P.; Benton, J.; Kambhampati, S.; and Scheutz, M. 2012. Tell Me When and Why to do it! Run-time Planner Model Updates via Natural Language Instruction. In *Proceedings of the Seventh International Conference on Human-Robot Interaction*.

Chao, C.; Cakmak, M.; and Thomaz, A. 2011. Towards Grounding Concepts for Transfer in Goal Learning from Demonstration. In *Proceedings of the International Conference on Development and Learning*.

DeJong, G., and Mooney, R. 1986. Explanation-based Learning: An Alternative View. *Machine Learning*.

Dietterich, T. 2000. The MAXQ Method for Hierarchical Reinforcement Learning. *Journal of Artificial Intelligence Research* 13:227–303.

Erol, K.; Hendler, J.; and Nau, D. 1992. UMCP : A Sound and Complete Procedure for Hierarchical Task-Network Planning. *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*.

Grollman, D., and Jenkins, O. 2010. Can We Learn Finite State Machine Robot Controllers from Interactive Demonstration. *From Motor Learning to Interaction Learning in Robots: Studies in Computational Intelligence* 264:407–430.

Huffman, S., and Laird, J. E. 1995. Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*.

Laird, J. E. 2012. *The Soar Cognitive Architecture*. MIT Press.

Meriçli, Ç.; Klee, S.; Paparian, J.; and Veloso, M. 2014. An Interactive Approach for Situated Task Specification through Verbal Instructions. In *Proceedings of the Thirteenth International Conference on Autonomous Agents and Multi-Agent Systems*.

Mohan, S.; Kirk, J. R.; Mininger, A.; and Laird, J. 2012. Acquiring Grounded Representations of Words with Situated Interactive Instruction. *Advances in Cognitive Systems* 2:113–130.

Mohan, S.; Mininger, A.; and Laird, J. E. 2013. Towards an Indexical Model of Situated Language Comprehension for Real-World Cognitive Agents. In *Proceedings of the Second Annual Conference on Advances in Cognitive Systems*.

Rybski, P. E.; Yoon, K.; Stolarz, J.; and Veloso, M. 2007. Interactive Robot Task Training through Dialog and Demonstration. In *Proceedings of the Second International Conference on Human-Robot Interaction*.