Interactively Learning a Blend of Goal-Based and Procedural Tasks

Aaron Mininger and **John E. Laird**
University of Michigan
2260 Hayward Street
Ann Arbor, MI 48109-2121

Abstract

Agents that can learn new tasks through interactive instruction can utilize goal information to search for and learn flexible policies. This approach can be resilient to variations in initial conditions or issues that arise during execution. However, if a task is not easily formulated as achieving a goal or if the agent lacks sufficient domain knowledge for planning, other methods are required. We present a hybrid approach to interactive task learning that can learn both goal-oriented and procedural tasks, and mixtures of the two, from human natural language instruction. We describe this approach, go through two examples of learning tasks, and outline the space of tasks that the system can learn. We show that our approach can learn a variety of goal-oriented and procedural tasks from a single example and is robust to different amounts of domain knowledge.

Introduction

As robotic and AI technologies advance, interactive autonomous agents are becoming ever more prevalent in our everyday lives. We can assign tasks to these agents, whether it's setting an alarm on a phone, answering a question, or driving a car. As these agents become more capable and general-purpose, it will be impossible to fully anticipate and implement every possible task in advance. It will be crucial to extend, customize, and correct the tasks that an agent can perform in easy and natural ways. This is the goal of research in the recently identified AI challenge problem of Interactive Task Learning (ITL; Laird et al. 2017). Research in ITL covers a range of domains (such as robots and artificial personal assistants) and forms of interaction (such as speech, language, demonstrations, or kinesthetic training).

Laird et al. (2017) identify a set of desired characteristics of Interactive Task Learning. It should be *general*, so that the agent can learn a diverse set of tasks that involve diverse concepts, procedures, and goals. The task learning should be *effective*; allowing the agent to learn all aspects of the task through interaction and generalize what is learned to perform the task in the future. Finally, the learning should be *efficient*; requiring few examples. We focus on applying these principles to the area of mixed-initiative situated interactive instruction, where the human instructor and the robotic agent collaborate together and engage in a joint conversation with the purpose of having the robot learn a new task 'from scratch' (with no initial knowledge about the task). This allows the agent to learn while it is performing the task and ask questions as problems arise. It also allows the instructor to provide knowledge, corrections, and feedback.

Task learning includes learning how to actually perform the task – what actions or subtasks the agent needs to execute to complete the task. One way is to represent the goal of a task, then use a state-based *policy*, which is a mapping from each state to the appropriate action, to reach the goal. This policy can be learned, such as through reinforcement learning, or the agent can use planning capabilities to identify the next action the agent should take. Mohan et al. (2014) use this formulation along with Explanation-Based Learning (EBL; DeJong and Mooney 1986) to learn goal-oriented hierarchical tasks. This EBL approach relies on the insight that "it is possible to form a justified generalization of a single positive training example, provided the learning system is endowed with some explanatory capabilities" (Mitchell, Keller, and Kedar-Cabelli 1986). In order to learn new goal-oriented tasks via EBL, the agent must have sufficient domain knowledge so that it can explain why the actions it executed led it to the goal. Some of the benefits of this approach are that it works over relational representations, which allows the agent to reason over complex task structures and their connection to language, and it can exploit rich domain knowledge to learn from relatively few examples (Mohan and Laird 2014). In addition, this approach can learn diverse types of task knowledge, including task parameters, availability conditions, subtasks, policy information, termination conditions, and action models.

Another way of learning to perform a task is to learn a procedure, which directly represents the steps needed to accomplish the task. This procedure can include complex control flow, including loops, conditional, and enumeration constructs. One such example is the work of Meriçli et al. (2014), whose agent can learn procedural tasks in an office environment by learning an Instruction Graph. This graph represents the actions and the control flow between them. However, without knowing the goal, an agent with only a procedure has difficulty adjusting if something goes wrong,

unless the instructor specifically mentioned what to do in those circumstances. An advantage of procedures is that in many cases it is straight forward to correct or extend them.

Both of these formulations have their trade-offs, with each having different types of tasks for which they are most appropriate. A goal-based approach is flexible to changes in the initial state and can adapt to unexpected conditions and task variations. In addition, some tasks are easily formulated by defining a goal. If you want the agent to store the milk in the fridge, you may not care exactly how it is accomplished. It can also be more *efficient* to teach, since the agent might find a way to achieve the goal using its existing knowledge without needing further instruction. However, in order for the agent to use planning, it must have sufficient domain knowledge to internally simulate its actions. A procedure-learning approach need not rely on planning, which may allow it to learn a task in an environment in which it does not know the world dynamics. It also may be difficult to formulate some tasks as achieving a goal as opposed to following a sequence of steps. For example, the task of patrolling an office building involves driving to several locations (possibly multiple times), but does not involve achieving a final goal apart from the actions.

We present a hybrid approach to Interactive Task Learning that learns both goal-oriented tasks and procedural tasks, as well as tasks that have elements of both. We extend the explanation-based approach of Mohan et al. (2014) by expanding the goal representation to allow for an ordered sequence of steps (including subgoals), and modifying the policy learning to build up a sequence of procedural steps (subgoals) when the explanation fails. This makes the task learning more *general*, by allowing the agent to learn tasks without a goal or where the agent lacks sufficient domain knowledge to learn a full policy. It is *effective* in learning all aspects of the task, and it can perform the task in the future after only one training example. It also generalizes to new task variations with no additional training. In addition, the procedural learning is tightly integrated with the task learning system – it is not a separate module. This allows the agent to learn tasks that involve aspects of both, such as a task that has a goal but the agent does not know how to accomplish it, or a task where there are different formulations of subtasks in the same hierarchy. It also improves the instructional *efficiency*, both by allowing the instructor to choose whichever formulation is easier to instruct for a particular task, and by eliminating the need for instructions when the agent can internally search to determine the next action.

We begin with a review of related approaches to Interactive Task Learning, then give a brief description of the agent architecture. Then we review the explanation-based approach to learning tasks and describe what aspects of the task are learned and how they are learned. We then describe our extensions and how procedural tasks are learned. We provide a qualitative analysis of how these extensions expand the space of learnable tasks, and demonstrate how the agent learns two different tasks. We describe how the new approach can compensate for a lack of domain knowledge. Finally, we conclude with a discussion of the results and identify shortcomings and future work.

Background

Interactive Task Learning can cover a range of interaction modalities, domains, and learning approaches. For example, Learning from Demonstration involves learning a policy for motor actions from training such as teleoperation or kinesthetic guidance (Argall et al. 2009). These approaches are usually used for learning motor tasks such as pouring liquid from a cup, and not larger scale hierarchical tasks.

There is a significant body of work on teaching tasks through instruction. One class of approaches involves learning the task problem space (goals, actions, concepts) and then using planning. Hinrichs et al. (2014) use language and sketching to teach games like Tic Tac Toe. Their system converts the input into the Game Description Language (GDL). Cantrell et al. (2011) have a system that learns the meaning of a verb used in a command by learning a representation of its goal, then use a general planner to accomplish the task. She et al. (2014) describe a system that is not directly given the goal but instead is given a sequence of actions that accomplish the task. From there, it learns both the goal and the preconditions, and it uses a planner to carry out the task. All of these rely on having sufficient models of the low-level actions to enable planning and require some form of the goal.

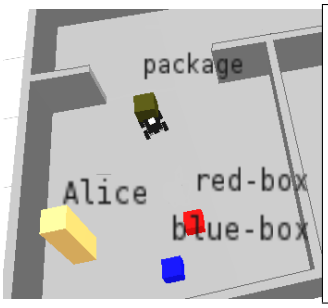
Other robotic ITL systems involve learning a procedure to carry out a task. The CoBot service robot (Meriçli et al. 2014) can learn a procedure that includes conditional and looping structures, to which the instructor can make corrections and changes. Mohseni-Kabir et al. (2015) have a system that learns a hierarchical task network (HTN) from a single example, and uses heuristics to group tasks and reduce the amount of instruction required. The system developed by Suddrey et al. (2017) also learns a HTN. It learns the preconditions of a task and can search for a way to satisfy the preconditions, giving some flexibility. However, without a direct representation of the goal, these systems are locked into performing a task a specific way.

Interactive Task Learning has also been applied to computing domains. For example, PLOW (Allen et al. 2007) can learn tasks in a web browser, where it can learn a procedure for tasks such as finding a hotel near a certain address using a webpage. LIA (Azaria, Krishnamurthy, and Mitchell 2016) can learn various personal assistant type tasks that can be procedurally executed. These approaches can generalize certain arguments to apply to similar situations, but do not actually represent the goal or use planning.

These systems focus on one task formulation or the other. In this work, we demonstrate a hybrid approach that learns both formulations, and applies to tasks that mix the two.

Agent Overview

This work builds on previous research on Interactive Task Learning, realized in an agent named Rosie (Mohan and Laird 2014), and implemented within the Soar cognitive architecture (Laird 2012). Soar provides a symbolic working memory, a rule-based procedural memory, a long-term semantic memory, an episodic memory, and interfaces to perception and action, as well as associated learning mechanisms for the long-term memories.



package(e_1) obj(e_1)
grabbed(e_1) red(e_2)
box(e_2) obj(e_2)
alice(e_3) person(e_3)
office(e_4) loc(e_4)
blue(e_5) box(e_5)
in(e_2, e_4)
left(e_2, e_3)
near(e_5, e_2)

Figure 1: A simulated world for the mobile robot and some of the corresponding state predicates.

Discard the soda.
What is the goal?
The goal is that the soda is in the garbage.
What do I do next?
Pick up the soda.
(The agent picks up the soda)
(The agent approaches the garbage)
(The agent puts the soda in the garbage)
I am ready for a new task

Figure 2: Dialog for teaching the *discard* task.

Perception and Actuation

The Rosie agent has been deployed on three platforms, including a tabletop arm, a 4-wheeled mobile robot, and the Fetch robot with a 7 DOF arm, as well as simulated versions of those robots. The same task-learning agent can operate across these domains by relying on a uniform representation of the world and basic actions. Domain-specific perceptual processing and procedural knowledge create and maintain a stable representation of the agent's current beliefs about the world, and implement a set of primitive tasks, such as `pick-up(obj)` or `go-to(loc)`.

The agent's beliefs about the world are represented in working memory as a set of entities $E = \{e_i\}$ and predicates over those entities $P = \{p_i\}, p_i = \texttt{name}(e_1, e_2, ...)$. Predicates over one entity are called properties, such as `visible(`e_1`)`, `red(`e_2`)`, or `table(`e_3`)`, and predicates over two entities are called relations, such as `in(`e_1`, `e_2`)` or `right-of(`e_3`, `e_4`)`. Entities are categorized as either *objects*, *persons*, or *locations*. Some example predicates are shown in Figure 1.

The instructor can interact with the agent using natural language through a chat interface. When a new sentence is received, the agent parses and grounds the sentence using knowledge in Soar based on Embodied Construction Grammar (Lindes and Laird 2017), producing a semantic representation of the sentence, connected to concepts in its working and semantic memories, many of which are grounded in its perception of the environment. When the agent communicates with the instructor, it produces a message structure which is converted into English using simple templates.

Task Learning

Rosie can learn all aspects of a new task through interactive instruction: task structure, goals, task decompositions, policy, preconditions, and postconditions. Work by Kirk et al. (2016) has also shown how the agent can learn new predicates that are used during by a task. The agent learns a general declarative representation of the task knowledge called the Task Concept Network (TCN) that it stores in its long-term semantic memory. It also learns procedural rules that apply the learned knowledge to the current situation when the agent is actually executing the task. The learning is *impasse driven*, which means that the agent attempts to execute the task until it is missing some knowledge required to make progress. Its response to the missing knowledge is to do internal reasoning or initiate an interaction with the instructor to obtain that knowledge. For example, if it is missing the knowledge to select the next action, it will either do an internal search to determine which action to select, or if that fails, ask the instructor what it should do. It then interprets the instruction, learns from it, and continues performing the task. In the following sections, we describe how each of these aspects of the task is learned in greater detail. Throughout, we will use the example of teaching the simple task: *'Discard the soda'*, which involves putting a soda can into the garbage. Example instructions can be seen in Figure 2.

Task Structure: When the agent receives a task command with a new name, it creates a new Task Concept Network (TCN) and extracts the argument structure from the command, replacing references to specific entities or predicates with *slots*, which are placeholders that will be filled in during a specific task execution. For example, with *discard,* there is a single argument, an entity (the soda), so the TCN is initialized with the task name and a single entity argument slot. More complex commands can have multiple arguments, e.g. *'Deliver the green box to the office.'* Refer to Figure 3 for the full TCN learned for discard.

Goal: Once the agent has a task structure, it attempts to perform it, but reaches an impasse because it does not know the goal. It then asks the instructor *'What is the goal?'* and the instructor can reply with a statement such as *'The goal is that the soda is in the garbage'*, which will be interpreted as a set of one or more predicates: $\{\texttt{in}(e_{soda}, e_{garbage})\}$. The agent adds a representation of the goal to the TCN, which includes representations of the goal parameters. Parameters that appear in both the goal and the task command (e_{soda}) are *explicit parameters* and are generalized by being linked to the same slots in the TCN. Parameters that only appear in the goal ($e_{garbage}$ and `in`) are assumed to be definitional for that goal and are not generalized. They are added as new slots with implicit values (descriptions of the argument that can be matched against the state).

The agent then constructs a representation of the goal in working memory using the general version in the TCN. This goal is connected to specific entities in the world, either matched through the arguments in the task structure (for explicit parameters) or through the implicit values for that slot (for implicit parameters). As a side effect of this

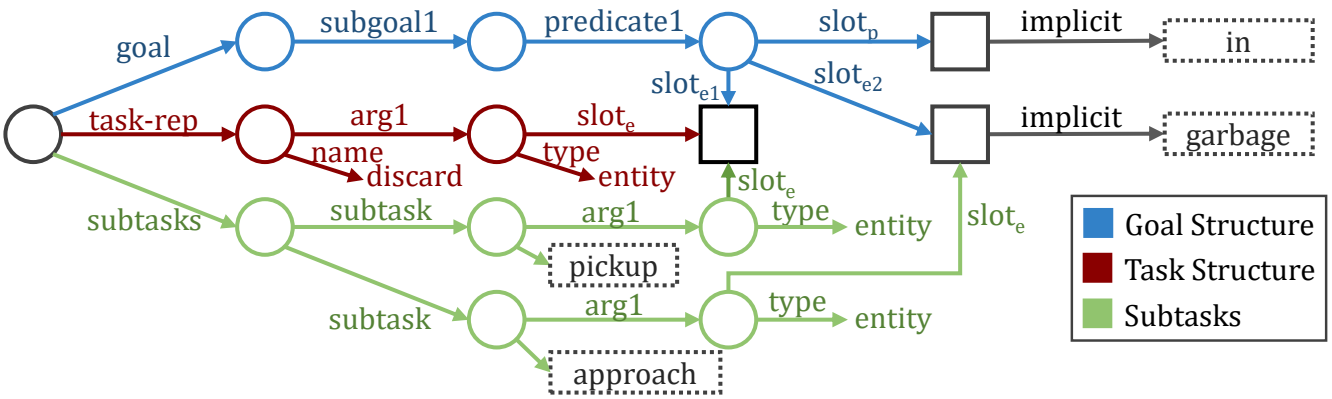


Figure 3: Task Concept Network for the *discard* task. Argument slots are shown as squares. Dashed rectangles represent the root nodes of other concepts in semantic memory. The third subtask for `putdown` is omitted.

processing, a rule is learned through Soar's chunking mechanism that in the future will elaborate the goal given the current task and it incorporates the matching logic in its conditions. In the case of *discard*, it learns a rule summarized as [`if task=discard(`e_i`) and garbage(`e_j`) then goal=in(`e_i`, `e_j`)`]. This is a rule that applies to a discard task even if there is a different (non-soda) argument.

Task Decomposition: To actually execute the task, the agent must decompose it into a sequence of subtasks. Each subtask is either a previously learned task, or one of the hand-coded tasks that the agent initially knows how to execute (such as `pick-up` or `go-to`). If it has a goal for the task, it performs an internal search to find a solution, which if successful, determines the next subtask to select. If it cannot identify the next subtask, it asks the instructor *'What should I do next?'* and the instructor can respond with the next subtask (e.g. *'Pick up the soda.'*). Rosie performs that subtask in its environment, and then uses internal planning to identify the next two actions, `approach(`$e_{garbage}$`)` and `put(`e_{soda}`, `$e_{garbage}$`)`, thus avoiding further instruction.

The agent learns each subtask to use in the future through a process that is very similar to learning the goal. First it stores a general structure of the subtask in the TCN, where the arguments are connected to slots (see Figure 3). Then, it matches that general structure against the current state to construct a proposal of that subtask. Through chunking, it also learns a rule to propose that subtask during the parent task that is connected to specific entities in the world. This proposal rule includes additional constraints inherent to the subtask, such as if putting down an object the object must be grabbed. For the discard task, it would learn a rule summarized as [`if task=discard(`e_i`) and visible(`e_i`) and not grabbed(`e_i`) then propose pickup(`e_i`)`].

Policy: Once the task is successfully completed, the agent knows which subtasks are used in the parent task, but does not have a specific policy for *when* to do each subtask. It must learn a policy mapping the current state to the appropriate subtask $\Pi : S \rightarrow T$. After the task is finished, the agent does a retrospective analysis of its experience and uses its domain knowledge to generate an explanation of why those subtasks led to the goal being satisfied. First, it retrieves the initial state (state of the world when the task began) from Soar's episodic memory, along with the sequence of subtasks that it executed. It then repeatedly selects each subtask in order, and simulating the effects of the subtask on the state until it achieves the goal. Soar's chunking mechanism identifies the features of the state that caused each subtask to succeed, and learns a rule that prefers the executed subtask in a given state. This EBL approach ignores irrelevant aspects of the state and learns a general rule that applies to new and different situations. For the discard task, it learns a rule that can be summarized as: [`if task=discard(`e_i`) and grabbed(`e_i`) and garbage(`e_j`) and near(`e_{self}`, `e_j`) then perform putdown(`e_i`, in(`e_j`))`].

Preconditions: The knowledge it learns from the above methods is sufficient for the agent to perform the task in the future. However, to truly learn hierarchical tasks, the agent must be able to use and plan with a learned task. Thus it must learn the preconditions (when it can perform a task) and postconditions (what the task accomplishes). To learn the preconditions, the agent again retrieves the initial state when the task began, and simulates performing the task using the policy. If it succeeds in reaching the goal, it learns a rule to propose the task in a similar manner to learning the policy. The rule will incorporate the features of the state that had to be present in order for the entire task to succeed. For discard, this would be: [`If object(`e_i`) and visible(`e_i`) and garbage(`e_j`) then propose discard(`e_i`)`].

Postconditions: Rosie does not have sufficient domain knowledge to model all of the side effects of each subtask. Thus, when the agent learns how the subtask changes the world (postconditions), it only includes what the agent *intends* to accomplish. It uses the learned representation of the goal as the postconditions, thus the postconditions for discard would be to add `in(`e_{arg1}`, `$e_{garbage}$`)` to the state.

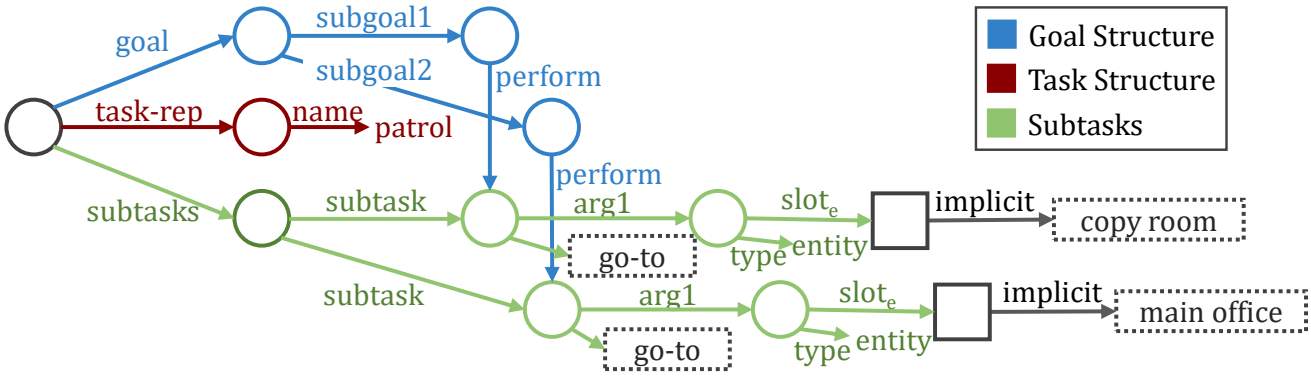


Figure 4: Task Concept Network for the patrol task with two steps. Argument slots are shown as squares. Dashed rectangles represent the root nodes of other concepts in semantic memory.

Learning Procedural Tasks

The above approach is a powerful and flexible way of learning new tasks, since the agent can use its pre-existing knowledge to adapt to different initial conditions or problems that arise during execution. For example, in the discard task, the agent expects to see the garbage at the start. If the agent was later told to do `discard` when the garbage was not visible, the `approach` subtask would fail. It could use its knowledge of the goal and ability to search for a solution. Using its knowledge it would identify and perform the additional subtask of `find`$(e_{garbage})$. The explanation-based approach to learning the policy assumes that there is a well-defined goal and that the agent can predict how the subtasks will achieve that goal. However, if one of those assumptions is broken, the agent cannot learn the policy and thus will not learn how to execute the task in the future.

Some tasks do not have a goal that can be easily expressed as achieving some external state in the world. For example, consider a *patrol* task where the agent is told to drive in a predetermined route around a building, going from one location to another. When the task is finished, the state of the world has not changed. Instead of achieving a new state of the world, the purpose of patrol is to perform the movement subtasks in a specific order – to follow a specified procedure. In other tasks, even when there is an explicit goal, the agent might lack sufficient domain knowledge to know why the subtasks it executed achieve the goal. For example, it may not know how a microwave works, but can follow instructions to use one. In this case, the agent should still learn to repeat the same procedure in the future, as opposed to simply failing to learn anything.

We extend the previous work on goal-based tasks by expanding the goal representations to include an ordered list of subgoals and once in a subgoal, have the agent execute a specific subtask instead of attempt to achieve a set of desired predicates in the world. We also change how the policy is learned by splitting it into multiple learning episodes and handling learning failures by adding new procedural subgoals onto the subgoal list. This process is described in detail in the following sections.

Expanding the Goal Representation

The previous work is only able to represent a single goal that consists of a set of one or more predicates over entities in the world. For example, the goal of `store`(e_{milk}) is `in`$(e_{milk}$, $e_{fridge})$ `and closed`(e_{fridge}). We expand the goal representation to include an ordered list of subgoals. An individual subgoal can be a set of predicates as before, but now it can also be a reference to a subtask. This represents the subgoal of performing the subtask, and allows the agent to represent a procedural task as a series of procedural subgoals. Consider a very simple patrol task that consists of going to the copy room and then going to the main office. This would be represented as two subgoals, one corresponding to the subtask `goto`$(e_{copyroom})$ and the second corresponding to the subtask `goto`$(e_{mainoffice})$. In the Task Concept Network, the subgoals have links to the appropriate subtask. An example of this new representation is shown in Figure 4.

This new goal representation requires that the state includes the current subgoal. When the agent starts to execute a subtask, it retrieves the initial subgoal. When the current subgoal is satisfied, the agent then retrieves the next one. When the final subgoal is satisfied, the agent completes the task. For state-based subgoals, the agent can determine it is satisfied by detecting the subgoal in the state. For procedural-based subgoals, the agent must deliberately mark the subgoal as being satisfied once the corresponding subtask has finished. The goal elaboration rule that constructs the current goal connected to the state, is learned in the same way as before, with the caveat that the rule must not only test the current state and task, but also the current subgoal.

Policy Learning

In the previous iteration of retrospective learning, the agent tried to learn the entire policy at once. If this failed due to missing domain knowledge about how a subtask modified the state, the agent would not learn anything. We therefore split the policy learning into individual learning episodes, where in each episode the agent attempts to learn one piece of the policy. This process is shown in Algorithm 1. The

Algorithm 1 Policy Learning

1: **procedure learn-policy**(task T)
2: $TCN \leftarrow$ **retrieve-TCN**(T)
3: $[subtask_i] \leftarrow$ **retrieve-subtasks**(T)
4: **for** $i = N \rightarrow 1$ **do**
5: $s_i \leftarrow$ **initial-state**$(subtask_i)$
6: $g \leftarrow$ **subgoal**(s_i, T)
7: $s_{next} \leftarrow$ **simulate**$(s_i, subtask_i)$
8: $plan \leftarrow$ **search**(s_{next}, g, Π)
9: **if** $g \neq \emptyset$ and $plan \neq \emptyset$ **then**
10: $s_{pre} \leftarrow$ **preconditions**$(st_i, plan)$
11: $\Pi(s_{pre}) \leftarrow subtask_i$
12: **else**
13: $g_{new} \leftarrow$ **create-subgoal**$(subtask_i)$
14: **prepend-subgoal**(TCN, g_{new})

agent retrieves the TCN for task T from semantic memory (line 2) and retrieves the series of subtasks it executes from episodic memory (line 3). The agent then works backwards. If it performed N subtasks, it starts with the Nth subtask ($subtask_N$) and goes through each until the first. For each subtask $subtask_i$ it retrieves the state of the world at the beginning of that subtask s_i and the subgoal at that time g (lines 5-6). It applies subtask $subtask_i$ to state s_i and then uses the existing policy Π to search for a plan to achieve the goal (line 8). If this succeeds, the agent uses the explanation to determine what was true in the state in order for the subtask and plan to successfully reach the goal (s_{pre}, line 10). This then is added to the policy (line 11). If there was no goal, or the plan failed, then a new goal is created to perform $subtask_i$ and is added to the beginning of the subgoal list in the TCN (lines 13-14).

In the case of a procedural task of N steps, it creates a list of N subgoals. In the case where there is a goal but the agent was unable to explain the entire policy, from the point at which the agent can generate an explanation, it can learn a policy, but not before that point. Thus, it then creates a list of procedural subgoals up to that point, and learns a policy from there. This means a task can have both procedural and goal-oriented components.

Evaluation

To evaluate our system, we first demonstrate the agent learning two different tasks, and describe what the agent learns and how the agent adapts to variations in the task. We then give a qualitative analysis of the space of tasks that our system can learn and its limitations.

Task Learning Examples

We now go through two examples of task learning, one which is a purely procedural task, and the second which demonstrates how the agent can still learn a task when domain knowledge is missing. These were performed with a simulated version of the mobile robot in a multi-room office environment, although we can perform them on the physical robot. This is the same simulated environment as shown in Figure 1.

Lead a tour.
What is the goal or first subtask?
Go to the main office.
(When finished)*What do I do next?*
Say 'This is the main office.'
What do I do next?
Go to the copy room.
What do I do next?
Say 'Here is where you can print papers.'
What do I do next?
You are done

Figure 5: Dialog of teaching the *tour* task.

Giving a Tour: The first task is to lead a tour of the building. For brevity, this involves only two locations, but there could be many more. The interaction for this task can be seen in Figure 5. The instructor says *'Lead a tour.'* This is the first time the agent has seen this task, so it creates a new TCN with a procedural structure with a single concept argument (for the concept tour). The instructor never gives a goal, but starts going through a list of tasks. Each time the agent executes the subtask and then asks what it should do next. Finally, the instructor says *'You are done.'* At this point the agent begins the retrospective policy learning with performing the final subtask (Saying 'Here is where you make copies'), but without a goal, the agent fails to learn a policy. The agent creates a new goal with the first subgoal of performing the *say* subtask. It keeps working back until it has created a list of four ordered subgoals, one for each subtask.

The next time the agent is told to lead a tour, it retrieves the first subgoal and performs the first subtask (driving to the main office). Once that is finished, it marks that subgoal as satisfied and advances to the next one. This occurs until all the subtasks are finished. Thus it is able to accomplish the task without any additional instructions.

Deliver: The second example is that of *deliver*, shown in Figure 6. The instructor gives the command *'Deliver the package to David'* and provides the goal of *'The goal is that David is holding the package'*. Normally, the agent would have sufficient domain knowledge to internally plan and solve the problem without any additional instruction. However, for this example we removed the model for `pickup` which adds the predicate `grabbed(package)`. Since `grabbed` is a precondition for the `give(package, David)` subtask, the search fails and the agent asks what to do next. The instructor says *'Pick up the package'*, and the agent does it. From there, the agent uses planning to find the next two steps: `find(David)` and `give(package, David)`. Once it gives the package to David, it recognizes that the goal was achieved, and completes the task.

During the policy learning, the agent first tries to learn the policy for `give`, and is able to simulate achieving the goal. Thus it learns the policy rule [`if task=deliver(`e_i`, `e_j`) and grabbed(`e_i`) and visible(`e_j`) then do give(`e_i`, `e_j`)`]. Similarly, it successfully learns a policy rule for `find` as [`if task=deliver(`e_i`, `e_j`) and grabbed(`e_i`) and not visible(`e_j`) then`

Deliver the package to David.
Please tell me the goal or the first action.
The goal is that David is holding the package.
(The search for the next action fails due to missing knowledge)
What should I do next?
Pick up the package.
(Rosie picks up the package)
(Rosie finds David)
(Rosie gives the package to David)

Figure 6: Dialog of teaching the *deliver* task without the model for the `pickup` subtask.

`do find(`e_j`)`]. However, when it tries to learn a policy for `pickup`, it fails because it is missing the knowledge that it results in the object being grabbed. Thus it pushes a new subgoal of performing the pickup action.

Afterwards, the agent is told to deliver a soda to Bethany. It will attempt the first subgoal, which is to perform `pickup`. However, suppose there is no soda in the room, and thus it cannot perform that action. But just as with other goals, it can try to search for a way to satisfy that subgoal. It finds that if it performs the subtask `find(`e_{soda}`)`, the soda will become visible and the `pickup` action will succeed. This will also get incorporated into the policy during the retrospective learning as the rule `if task=deliver(`e_i`, `e_j`) and subgoal=pickup(`e_i`) and not visible(`e_i`) then do find(`e_i`)`. This demonstrates the benefit of formulating these procedural steps as subgoals and allowing the system to use planning to handle variations in the task. Note that it is does all of this without requiring any further instructions, and generalizes to new variations of deliver involving different objects and people.

Space of Learnable Tasks

This work has expanded the space of tasks that Rosie can learn to any task that can be broken down into a series of individual subtasks. In the least general case, Rosie can memorize the task demonstration as a procedure and follow it in the future. If the agent is provided with a description of the goal and additional domain knowledge in the form of subtask models, it can learn a more flexible policy, which allows some variation in the task. Currently, the subtasks must be singular, meaning they involve specific entities in the world and cannot have collections or numbers of entities (such as *all the red blocks* or *five drinks*). These tasks must be eventually decomposable into the initial set of subtasks that the agent knows how to do. It cannot learn new primitive motor tasks; it can only learn compositions of existing motor tasks. Similarly, the tasks are limited to entities and properties that the agent knows how to perceive and incorporate into the state representation. The agent also cannot perform tasks that involve complex control flow, such as loops, enumerations, or conditional subtasks and cannot learn task variations depending on the arguments. Currently, to teach a variation on a task we would use a different verb (e.g. *'Deliver the (obj) to (person)'* vs *'Take the (obj) to the (location)'*). Our approach drastically increases the number of tasks that the agent can learn. Previously, the same restrictions above still applied, but the agent required that the task had a single goal that could be described as a set of state predicates, and that the agent knew sufficient models to explain how that goal was achieved. Examples of new tasks that were previously not possible include patrolling a route, giving a tour, making an announcement, and driving in a square.

Discussion

We present a hybrid approach to Interactive Task Learning that can learn tasks that are formulated as achieving a goal, and tasks that are formulated as following a procedure. When the agent has a goal, it uses its rich domain knowledge to generate plans that will achieve that goal. This is flexible to variations in the initial state and ways of accomplishing the task. But if that domain knowledge is incomplete or if there is not external goal to achieve, a procedure can provide a different way of accomplishing a task, which is less flexible but requires less knowledge. In addition, since our approach unifies these two formulations, it can learn tasks that are a blend of the two – where a goal is given but the agent is unable to generate a complete plan. The `deliver` example also shows that with a hybrid approach, the agent can utilize its planning capabilities even during procedural steps. This ability to learn either task formulation is a significant step in improving the *generality* of an ITL agent. It also is *effective* at learning all the aspects of a task and is able to perform the task in the future without needing additional instructions. This approach is also able to learn many tasks after a single training example, which makes the learning very *efficient*. In addition, this explanation-based approach produces general results that can transfer to new variations in the task. Since entities are parameterized, learning a task such as *deliver* for one object automatically transfers to other objects with the same affordances. Note that this transfer only happens within variations of a task, not between different tasks.

There is also a benefit in having an approach that can learn diverse task formulations, which distinguishes this work from those that only learn goal-based or procedural formulations. Indeed, one may be able to learn all the tasks mentioned using only one or the other. However, we have shown in the *deliver* example is that incorporating planning into a procedure can accommodate simple variations and reduce the number of instructions needed. Alternatively, one could possibly formulate the tasks like *give a tour* as achieving a goal and solve using planning (i.e. formulate it as a goal involving information gathering and communication). However, the goal would have to include temporal ordering information, and require that the state representation include complex information and communication predicates. It would also be difficult for a non-expert user to try and describe such a goal, as opposed to listing the steps.

As discussed in the previous section, there are limitations of the tasks that the agent can learn. Since we can only learn tasks that can be decomposed into a series of singular subtasks, any task our agent can learn could also be solved by a number of traditional planners. However, our approach is

not directly competing with planning research, in fact, integrating a better planner would improve the overall performance. Our work addresses two key concerns. First, how can an agent learn a representation of the task through instruction that can then be used during planning, and second, what if the agent lacks the necessary knowledge for the planner to succeed?

There are also task formulations that our agent cannot currently handle. Some examples include preventing something from occurring, maintaining some condition, or maximizing some quantity. It is an area of future research to see how well these formulations can be accommodated into the current framework. Ideally we would like to have a unified approach that can handle all of these. In addition, there are other approaches which can learn tasks which are more than sequences. For example, Meriçli et al. (2014) can learn a procedural Instruction Graph that incorporates additional control constructs, such as loops and conditionals. We plan on expanding the goal and task representations further to incorporate those structures, rather than being limited to a sequence, and allowing arguments to be collections of entities. In addition, our agent cannot learn variations of a task where the goal or procedure is different depending on the argument being used (such as storing milk in the fridge verses cereal in the pantry). This could be addressed by adding conditional structures.

Finally, the instructor cannot correct or extend a procedure, which would be one way of modifying a task. Our agent only generalizes when it can create an explanation to justify it, so this is likely to be correct if the domain knowledge is correct. However, if some important aspect of the task is not captured in the explanation, then it may learn an incorrect policy. Another area of future work is to allow the agent to learn new task models. This could allow it to initially learn a procedure, but then later move on to a more flexible policy once it has acquired the additional knowledge. These action models could be obtained through instruction or learning from experience.

Acknowledgments

We thank current and former students who have contributed to our research on interactive task learning, including James Kirk, Shiwali Mohan, and Mazin Assanie.

The work described here was supported by the Air Force Office approaches can learn that are worthy of future work of Scientific Research under Grant Number FA9550-15-1-0157 and the Office of Naval Research under Grant N00014-08-1-0099. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the AFOSR, ONR, or the U.S. Government.

References

Allen, J.; Chambers, N.; Ferguson, G.; Galescu, L.; Jung, H.; Swift, M.; and Taysom, W. 2007. Plow: A collaborative task learning agent. In *AAAI*, 1514–1519.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

Azaria, A.; Krishnamurthy, J.; and Mitchell, T. M. 2016. Instructable intelligent personal agent. In *AAAI*, 2681–2689.

Cantrell, R.; Schermerhorn, P.; and Scheutz, M. 2011. Learning actions from human-robot dialogues. In *RO-MAN, 2011 IEEE*, 125–130. IEEE.

DeJong, G., and Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine learning* 1(2):145–176.

Hinrichs, T. R., and Forbus, K. D. 2014. X goes first: Teaching simple games through multimodal interaction. *Advances in Cognitive Systems* 3:31–46.

Kirk, J. R., and Laird, J. E. 2016. Learning general and efficient representations of novel games through interactive instruction. *Advances in Cognitive Systems* 4.

Laird, J. E.; Gluck, K.; Anderson, J.; Forbus, K. D.; Jenkins, O. C.; Lebiere, C.; Salvucci, D.; Scheutz, M.; Thomaz, A.; Trafton, G.; et al. 2017. Interactive task learning. *IEEE Intelligent Systems* 32(4):6–21.

Laird, J. E. 2012. *The Soar cognitive architecture*. MIT press.

Lindes, P., and Laird, J. E. 2017. Ambiguity resolution in a cognitive model of language comprehension.

Meriçli, C.; Klee, S. D.; Paparian, J.; and Veloso, M. 2014. An interactive approach for situated task specification through verbal instructions. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 1069–1076. International Foundation for Autonomous Agents and Multiagent Systems.

Mitchell, T. M.; Keller, R. M.; and Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine learning* 1(1):47–80.

Mohan, S., and Laird, J. E. 2014. Learning goal-oriented hierarchical tasks from situated interactive instruction. In *AAAI*, 387–394.

Mohseni-Kabir, A.; Rich, C.; Chernova, S.; Sidner, C. L.; and Miller, D. 2015. Interactive hierarchical task learning from a single demonstration. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, 205–212. ACM.

She, L.; Cheng, Y.; Chai, J. Y.; Jia, Y.; Yang, S.; and Xi, N. 2014. Teaching robots new actions through natural language instructions. In *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*, 868–873. IEEE.

Suddrey, G.; Lehnert, C.; Eich, M.; Maire, F.; and Roberts, J. 2017. Teaching robots generalizable hierarchical tasks through natural language instruction. *IEEE Robotics and Automation Letters* 2(1):201–208.