Workshop on Human-Robot Collaboration for Industrial Manufacturing,
Robotics Science and Systems, Berkeley, CA, July 2014.

Collaborative Learning of Hierarchical Task Networks from Demonstration and Instruction

Anahita Mohseni-Kabir, Sonia Chernova and Charles Rich
Worcester Polytechnic Institute, Worcester, MA, USA
{amohsenikabir, soniac, rich}@wpi.edu

***Abstract*— We present an approach for learning complex procedural tasks from human demonstration. Our main innovation is to view the interaction between the human and the robot as a mixed-initiative collaboration. Our work utilizes hierarchical task networks to enable the robot to learn complex procedural tasks. Our contribution is to integrate hierarchical task networks and collaborative discourse theory into the learning from demonstration paradigm to enable robots to learn complex tasks in collaboration with the human teacher. We demonstrate techniques for generalization based on merging multiple demonstrations and policy refinement through question asking. We validate our approach in a car maintenance domain.**

I. Introduction

In this work, we focus on advancing the state of the art in intelligent agents that can learn complex procedural tasks from humans. Our work draws on several research areas, primarily robot learning from demonstration (LfD), hierarchical task network (HTN) planning, and collaborative discourse theory. Our approach is to view the interaction between the human teacher and the learning agent as a mixed-initiative *collaboration*, in which both parties are committed to the shared goal of successful learning, and in which both parties make contributions in the form of both actions and communication, including verbal instructions, asking questions, and critiquing. For this, we draw heavily on collaborative discourse theory and tools.

In this paper, we present algorithms that will allow a robot to learn complex tasks from a human teacher. Our approach is based on the conjecture that it is often easier for people to generate and discuss examples of how to accomplish tasks than it is to deal directly with task model abstractions. Our approach is thus for the robot to iteratively learn tasks from human demonstrations and instructions[1]. In addition, the robot is able to ask questions, to which human responds. By engaging the robot as an active partner in the learning process, and by using the hierarchical structures, we believe that complex tasks can be naturally taught by non-expert users.

Finally, because our target application domain is learning complex procedural tasks, such as manufacturing, equipment inspection and maintenance, we use HTNs as the representation of the learned knowledge. HTNs are a powerful framework for representing and organizing task models that is both efficient for computers and natural for humans. The hierarchical structure facilitates reusing knowledge in similar domains. In our work, we use the ANSI/CEA-2018 [11] standard for HTNs. In this standard, an HTN is an abstract, hierarchical, partially ordered representation of the actions typically performed to achieve goals in the application domain. This choice of representation is also synergistic with collaborative discourse theory, which is based on HTNs.

In summary, our work makes contributions in the following areas:

1. a unified system that integrates hierarchical task networks and collaborative discourse theory into the learning from demonstration;
2. a novel approach for learning task structure from a small number of demonstrations, including the task hierarchy, temporal constraints and inputs/outputs of a task;
3. novel generalization techniques that reduce the number of demonstrations required to learn the task through input generalization and merging;
4. integration of mixed-initiative interaction into the learning process through question asking.

In the following sections, we begin by discussing related work in the areas of collaborative discourse theory and robot learning from demonstration. We then present an overview of our system and describe the learning, generalization and question asking components in detail. We then present validation results in a simulated domain.

II. Related Work

In this section, we discuss related work in collaborative discourse theory and robot learning from demonstration, particularly as applied to planning.

[1]We differentiate between demonstration and instruction, and use *instruction* to indicate that the teacher tells the robot the steps of a task without any execution of these steps, and *demonstration* to indicate that the robot (or teacher) also performs the desired action.

A. Collaborative Discourse Theory

Discourse is the technical term for an extended communication between two or more participants in a shared context, such as collaboration. Collaborative discourse theory refers to a body of empirical and computational research about how people collaborate [5], that uses the same hierarchical task abstraction notions expressed in HTNs. We are applying the principles of collaborative discourse theory in this work to improve the effectiveness of the interaction between the human teacher and robot learner (such as to help them jointly manage their focus of attention) using a software tool called Disco, which is the open-source successor to Collagen [12].

B. Learning from Demonstration

Several past approaches have combined planning and learning from demonstration [1]. Nicolescu and Mataric [9] implemented a system for LfD that enables a robot to learn and refine representations of complex tasks, and to generalize multiple demonstrations into a single graph-like representation. In [14], Veeraraghavan and Veloso presented a demonstration based approach for teaching a robot sequential tasks with repetitions. Our approach differs from the above methods in representation, and in that it integrates learning through demonstration, instruction and asking questions. Niekum et al. [10] presented a method for automatically segmenting low-level trajectory demonstrations, recognizing repeated skills, and generalizing complex tasks from unstructured demonstrations. In contrast, our work focuses on high-level task learning, in which the task structure is more intuitively clear to users, allowing us to leverage human knowledge to learn from fewer demonstrations.

HTNs have been integrated with LfD by Rybski et al. [13] in an algorithm that combines spoken language understanding, dialog, and physical demonstration. Their task representation allows tasks to be constructed from smaller subtasks in order to create complex hierarchical structures. The robot can also verify the task with the human through dialog and permit the human to add additional conditional cases. Their approach differs from ours in that it does not focus on learning ordering constraints, parametrization, and generalization from multiple demonstrations. Furthermore, the robot's questions are mostly focused on filling unspecified conditions(i.e. else statements) while our work seeks to ask more general questions about the HTN structure.

An important recent innovation in LfD, which we will employ in our work, is to incorporate *active learning*, by which we mean that the robot requests certain information by asking questions. The key research challenge in active learning is to formulate questions that the user can understand and which will have a high utility to the robot's learning algorithm. Chernova and Veloso [3] developed confidence-based autonomy (CBA), an interactive algorithm for policy learning from demonstration which enabled the robot to ask the human to label the states in which it is uncertain of which action to take. In later work, Cakmak and Thomaz [2] formalized three query types – label, demonstration and feature questions – and studied their use in LfD.

Several other works have focused on learning from instruction. Huffman and Laird [6] explored the requirements for instructable agent design, and demonstrated that a simulated robotic agent could learn useful instruction in natural language. In [7], Mohan and Laird extended this work by instantiating their design in the Soar cognitive architecture. In this work, they also used a hierarchical structure for their task representation. Our approach differs from theirs in our focus on learning ordering constraints, parametrization and generalization and independence from a particular cognitive architecture. Our additional distinction is that our robot will ask questions about the structure of the tasks, for example concerning ordering constraints or applicability conditions, whereas in their approach, the questions are focused on learning the steps of the tasks.

Garland et al. [4] present a task model development environment that infers task models from examples. The novel aspect of their environment was the support for a domain expert to refine past examples. In contrast to our work, which seeks to learn from very few examples, the presented work requires many demonstrations from the human teacher. Moreover, our system can also ask questions from the human teacher, whereas their approach entirely relies on the refinement from the teacher.

III. System Overview

Fig. 1 presents an overview of our system architecture. The key collaborative learning components are highlighted in blue and are discussed in detail in Section IV.

The architecture consists of two subsystems, learning and execution. The learning subsystem is responsible for learning the hierarchical task structure, ordering constraints, and inputs/outputs of tasks. After learning this general structure, we apply the Generalization module, which improves the generality of the task model through a set of heuristics. The Question Asking module is responsible for asking questions. The entire learning process is supported by Disco, an implementation of collaborative discourse theory and ANSI/CEA-2018. Disco's dialog management capabilities are used to maintain a focus stack which keeps track of the current topic and has expectations for what needs to be said or done next.

During the execution of a learned task, the Disco planner decomposes each non-primitive task in the HTN into its subtasks. When the planner reaches a primitive

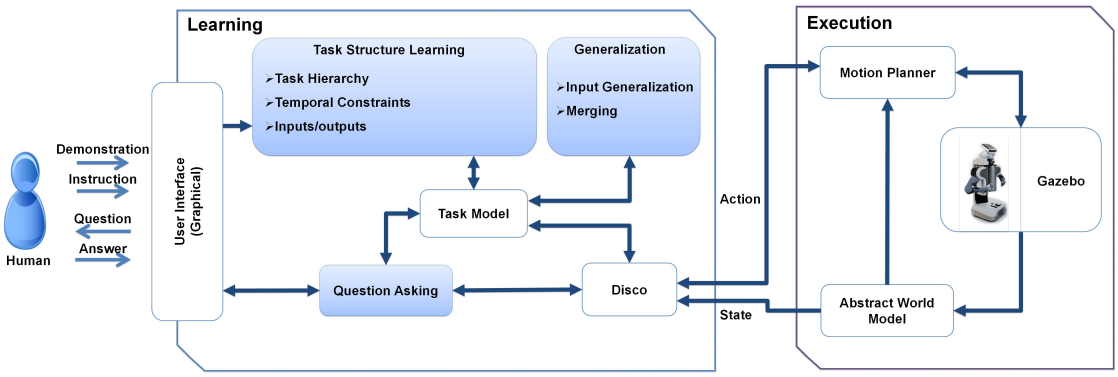

Fig. 1: System architecture

task, the primitive task is sent as an action command to the execution subsystem. The execution subsystem includes the Motion Planner, Abstract World Model (AWM), and the learning agent embodied either as a physical PR2 or as a Gazebo simulation. The Motion Planner receives primitive task commands, generates plans for their execution, and sends the plans to the PR2 or Gazebo. The AWM module tracks the current state of the world based on the robot's sensor data. Failures in the execution of the task are propagated back to Disco both through the Motion Planner (planning failure) and AWM (execution failure). Fig. 2 presents a snapshot of the execution of `RotateTires` task in Gazebo with the recent executed primitives shown under it.

The inputs to the system are the human's demonstrations, instructions and answers to questions. Fig, 4 shows an example learning scenario for the `RotateTires` task that utilizes all three types of interaction – demonstrations, instructions and questions. The goal of this process is to learn a task representation similar to the hand-coded HTN shown in Fig. 3 (the exact structure of the hierarchical model is determined by the human's demonstrations, instructions and answers to questions, and it may change based on the order of the inputs). In the current implementation, the user interacts with the robot via a GUI connected to Gazebo. This graphical user interface is completely menu-based and does not involve any natural language processing. Each demonstration starts by clicking on the `Start` button. The human teacher then chooses the primitive or non-primitive tasks as the steps of this new demonstration. If the chosen task is a defined non-primitive task or a primitive task, the task will be executed and learned by the robot as part of the new demonstration, otherwise the task will be learned as part of the new instruction. The demonstration ends by clicking on the `Done` button (Fig. 4 shows the log of this interaction.). Fig. 5 shows an example interaction between the robot and human during task execution. Here again, the robot may ask questions about task execution preferences.

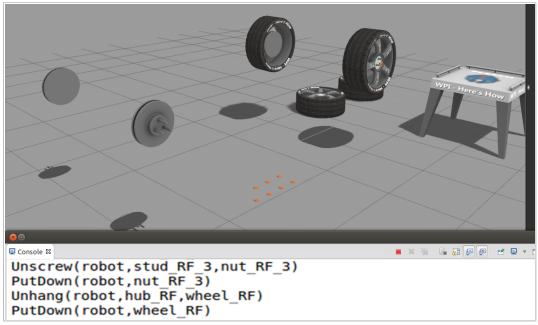

Fig. 2: Execution of RotateTires task in Gazebo

IV. Learning Subsystem

In this section, we provide a comprehensive overview of the learning subsystem and its components.

A. Task Structure Learning Module

The Task Structure Learning module is responsible for learning the task structure of the current demonstration sequence and generating the task model. This module integrates three functions:

1) Generating the Task Hierarchy: One of the core functionalities of the learning subsystem is to learn the implicit hierarchical structure of the demonstration sequence. Each task in an HTN has one or more *recipes*, or methods for decomposing non-primitive actions. Each recipe specifies a set of steps that are performed to achieve the non-primitive action that is the collective objective of the steps (e.g., rotate tires in an x-pattern or front to rear). An *optional step* in an HTN represents a step that is not required for execution and whose execution depends on user preference.

In each new demonstration sequence, the user either teaches a new non-primitive action (e.g., `UnscrewHub`) or demonstrates a new recipe for an existing non-primitive action (e.g., `xPattern`) by using existing non-primitive and primitive actions.

The system provides the user with the flexibility to teach tasks bottom-up, top-down, or the mix of these approaches. In the bottom-up approach, the user starts

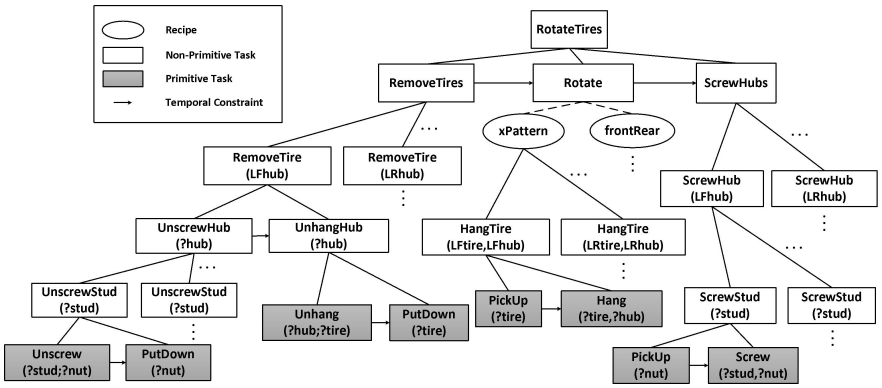

Fig. 3: Hand coded HTN for tire rotation

```
Human: "Please add RemoveTires, Rotate, and ScrewHubs as
non-primitive tasks"
Human: "Begin instruction of RotateTires"
  Human: "RemoveTires"
  Human: "Rotate"
  Human: "ScrewHubs"
  Human: "Done"
...
Human: "Please add UnscrewStud as a new non-primitive task"
Human: "Begin instruction of UnscrewHub"
  Human: "UnscrewStud on stud1 of LFhub"
  Agent: "Should I achieve UnscrewStud on stud2
  and stud3 of Lfhub"
  Human: "Yes"
  Agent: "I learned to perform UnscrewStud on stud2 of LFhub"
  Agent: "I learned to perform UnscrewStud on stud3 of LFhub"
  Agent: "Done"
Human: "Begin demonstration of UnscrewStud"
  Human: "Please perform Unscrew on stud1 of LFhub"
  Agent performs Unscrew on stud1 of LFhub
  Human: "Please perform Putdown on Nut17"
  Agent performs PutDown on Nut17
  Human: "Done"
  ...
...
```

Fig. 4: Scenario for learning the HTN shown in Fig. 3

with the primitive tasks and combines them into larger non-primitives (the demonstration of `UnscrewStud` in Fig. 4 is an example of bottom-up approach.). In top-down approach, the user starts with the top-level non-primitives (e.g., `RotateTires`, as in Fig. 3) and incrementally decomposes until the primitives are reached (the demonstration of `RotateTries` and `UnscrewHub` in Fig. 4 are examples of top-down approach.). Thus, the user's demonstrations help the robot to iteratively complete its hierarchical task model.

2) Associating Inputs/Outputs: Each task within the HTN has zero or more *inputs* and *outputs* associated with it. Examples of inputs and outputs can be seen in Figure 3, preceded by a question mark. The input of a task must be specified by the user at demonstration time (e.g., a specific `hub` must be selected when performing a `RemoveTire` action), thus allowing the input to be added to the task definition. Determining the output of a task is more complicated, since we must determine

```
Human: "Please execute RotateTires"
  Agent starts executing RemoveTires
    Agent performs Unscrew on stud1 of LFhub
    Agent performs PutDown on Nut19
    ...
  Agent starts executing Rotate
    Agent: "Should I achieve Rotate by executing xPattern
    or frontRear?"
    Human: "xPattern"
    ...
  ...
  Agent starts executing ScrewHubs
    ...
    Agent performs Screw on stud3 of RRhub
  Agent: "Done"
```

Fig. 5: Scenario for executing the HTN shown in Fig. 3

whether the output of a subtask relates to this step alone or must be propagated up the hierarchy. The algorithm for output propagation is beyond the scope of this paper.

3) Learning the Temporal Constraints: Demonstrations are intrinsically totally ordered, i.e., any set of (discrete, non-overlapping) actions performed in the real world occur in sequence. However, in many cases, only some of the demonstrated ordering is fixed. For example, in tire rotation task, even though you must demonstrate unhanging all four tires in some order, the order does not matter. Learning the minimum required ordering constraints is important to have a more flexible and reusable plan.

One of the contributions of the paper is an automated algorithm for finding the temporal constraints between steps in a recipe. Past approaches have mostly focused on learning these constraints from multiple demonstrations, which requires many demonstrations in a task with many steps. In addition to using past approaches, we are using a new technique [8], which enables us to learn these temporal constraints from a single demonstration.

The general rule for finding the temporal constraints is that if the output of one step is the input of another step, then these steps are ordered. To give a concrete example in tire rotation task, suppose the user demonstrates `UnhangHub` task with a `hub` as an input by using two

primitive actions: Unhang with hub as its input and tire as its output, and PutDown with tire as its input. In this example, there is a temporal constraint between Unhang and PutDown since the input to PutDown is the output of Unhang (Fig. 3).

Other temporal constraints are found by propagating the inputs and outputs up to the root of the tree and applying the mentioned general rule to them. For example, as shown in Fig. 3, RemoveTires should be done before Rotate since tires are the output of Unhang which are then used as an input of HangTire in Rotate.

B. Generalization Module

The generalization module performs two functions, input generalization and merging, explained below.

1) Input Generalization: Generalizing over the inputs of a task is useful because it makes the task structure more reusable and flexible, and because it reduces the number of demonstrations required for learning the task. We have implemented two input generalization methods:

Type generalization: Suppose we have two instances of the PickUp task with two different types of inputs, Tire and Nut. Instead of maintaining two separate actions with different types of inputs, our goal is to generalize the input of the PickUp task as relating to some more abstract type, such as MovableObject. To aid in this generalization, we rely on a simple object ontology that classifies objects of different types. Specifically, when encountering two different inputs to the same task (e.g., Tire and Nut), we reference the ontology and designate the parent class, if any, as the new input (e.g., MovableObject).

Part/whole generalization: Suppose we have three inputs LFhub.StudA, LFhub.StudB, and LFhub.StudC (the three studs on the left front hub) as inputs of a ScrewHub task. Instead of representing all three inputs explicitly, we generalize to one input, namely the object which is their shared parent (LFhub). This form of generalization significantly simplifies the interaction between the human and the robot as it eliminates the need for the user to specify three studs explicitly in all following demonstrations.

2) Merging: Once the current demonstration sequence is encoded as an HTN using the above methods, we check whether any previous demonstrations exist for this task. If this is the only demonstration, then the HTN remains unchanged. However, when the human provides multiple demonstrations of the same task, the system merges them to allow for generalization across the two examples. This approach has three advantages. First, this avoids adding a separate recipe for each tiny difference between the new demonstration for a specific task and the previously learned model. Second, by merging different demonstrations and factoring the common steps between them, we are postponing the system's needs to choose between the recipes, resulting in a more robust system. Third, this reduces the number of demonstrations required to learn the task.

Fig. 6 shows an example of merging two demonstrations in tire rotation task. In this example, instead of having two recipes for RotateTires task, we capture their common steps by adding the task Rotate (the task name is specified by the user).

The pseudocode for the merging algorithm is shown in Algorithm 1. First, the MERGETASKS function is called with current model (existing task) and the new demonstration for the task. If the new demonstration is satisfiable by the current model, there will be no changes in the model and the merge ends (lines 2-3). If the new demonstration is not mergeable with the current model, the new demonstration will be added as an alternative recipe of the current model (lines 6-7). Otherwise, if the models are mergeable, the new demonstration will be combined with current model using the MERGE function (lines 4-5).

Mergeability is determined by the MERGEABLE function through two steps: GetRootRecipes and FindMaxLCS. The GetRootRecipes function returns all possible recipes of the model (e.g., if this function is called on the Rotate function in Fig. 3, it will return the steps of xPattern recipe, and the steps of frontRear recipe.). Then the FindMaxLCS function is called to determine which recipe shares more common steps with the new demonstration by using Longest Common Subsequence (LCS) algorithm (line 12). If there are no shared steps between any of the recipes and the new demonstration, the model and the demonstration are not mergeable. Otherwise the recipe with the most shared steps will be chosen to be merged with the new demonstration.

To merge the LCS recipe with the new demonstration, the MERGE function iteratively selects two pairs of shared steps in the two HTNs, and merges the steps between them. This process is illustrated in Figure 7. First, given the new demonstration and the base recipe, we add start and end steps to each (line 16). Next, we calculate the overlap (using LCS) between the base and the new (line 17). Then, for each pair of consecutive steps in the overlapArray, we find the corresponding steps in the base and new (lines 21-26) (note that BI and NI are the steps between pairs of equivalent consecutive steps in base and new, respectively). If BI and NI are empty, we will continue by choosing the next equivalent consecutive steps (lines 27-28). If NI is not empty and BI is empty, we will add NI steps to the model but we will mark them as optional steps (line

30), but if BI is not empty and NI is empty, we will make BI steps in model as optional (line 32). If neither of them is empty, we replace BI with a new task, and BI and NI become alternative recipes in the new task (lines 34-36).

Importantly, the merge operation is not commutative, and thus changing the order in which demonstrations are performed will change the hierarchical structure of the HTN. However, the execution of the primitive actions within the model is preserved, regardless of demonstration order, ensuring correct execution of the task regardless of demonstration order.

Algorithm 1 Merge algorithm

```
1: function MERGETASKS(model, demonstration)
2:   if Satisfiable(model, demonstration) then
3:     return
4:   else if (maxLCS ← Mergeable(model, demonstration))
     ≠ NULL then
5:     Merge(maxLCS, demonstration)
6:   else
7:     AddAlternativeRecipe(model,demonstration)
8:   end if
9: end function

10: function MERGEABLE(model, demonstration)
11:   modelRootSeqs ← GetRootRecipes(model)
12:   maxLCS ← FindMaxLCS(modelRootSeqs,
     demonstration)
13:   return maxLCS
14: end function

15: function MERGE(base, new)        ▷ base is the chosen
    recipe, and new is the new demonstration
16:   Add start and end to base and new
17:   overlapArray ← Overlap(base,new)
18:   for Pi ∈ overlapArray do
19:     C1 ← P_i in overlapArray
20:     C2 ← P_{i+1} in overlapArray
21:     B1 ← find pointer to C1 in base
22:     B2 ← find pointer to C2 in base
23:     N1 ← find pointer to C1 in new
24:     N2 ← find pointer to C2 in new
25:     BI ← steps between B1 and B2
26:     NI ← steps between N1 and N2
27:     if BI = empty ∧ NI = empty then
28:       Do nothing
29:     else if BI = empty ∧ NI ≠ empty then
30:       modelRecipe ← AddOptional(base,BI)
31:     else if BI ≠ empty ∧ NI = empty then
32:       modelRecipe ← MakeOptional(base,NI)
33:     else
34:       modelRecipe ← MakeNewRecipe(BI)
35:       demRecipe ← MakeNewRecipe(NI)
36:       AddAlternativeRecipe(modelRecipe,demRecipe)
37:     end if
38:   end for
39: end function
```

C. Question Asking Module

The final component of our system is the Question Asking module, which is central to our mixed-initiative interaction system. The ability to ask questions is key to any learning process. Since the robot and teacher represent knowledge differently, the teacher does not always know what additional information the robot requires. The robot is able to expedite the learning process by asking questions when it lacks information. Additionally, question asking decreases the burden on the teacher and makes the interaction more natural. Our current system supports five question types:

1. When the merging process results in the creation of a new alternative recipe, the robot is able to ask the user about its applicability conditions. For example, a question such as "When should `Rotate` be performed through: `HangTire(LFtire,LRhub) HangTire(RFtire,RRhub)...`?". In response, the user can either add an applicability condition (e.g., perform this recipe for all Ford models), or not. By choosing the latter option, recipe selection is postponed until execution, at which time the user will be queried again with a choice of recipes.
2. Directly related to the above, when there is a choice of multiple recipes available for execution, and more than one meets the applicability conditions, the robot asks the user to select between the recipes (Fig. 5).
3. When the merging process leads to the creation of a new task, the robot asks the user to specify the name for the task (e.g., the name `Rotate` in Fig. 6 is specified by the user).
4. If the user does not specify the input of an action during execution (e.g., the human tells the robot to perform `Unscrew` task without specifying an input), the robot asks for the value of the missing input.
5. In tasks with repeated steps, the robot automatically infers the repeated behavior and queries the user to confirm whether the currently repeated pattern should be generalized to other inputs. Many manufacturing and maintenance tasks include repeated steps which are performed on the same types of objects (e.g., in tire rotation task, robot performs the `RemoveTire` task on all four tires). Instead of requiring the human to demonstrate all of the steps of the task, the robot asks whether the same action should be applied to other inputs of a matching type (e.g., other tires). This form of generalization greatly reduces the burden on the teacher, and the question ensures that over-generalization does not accidentally occur. An example of this type of interaction is shown in Fig. 4.

Together, these questions result in a fluid, mixed-initiative interaction during the learning and execution

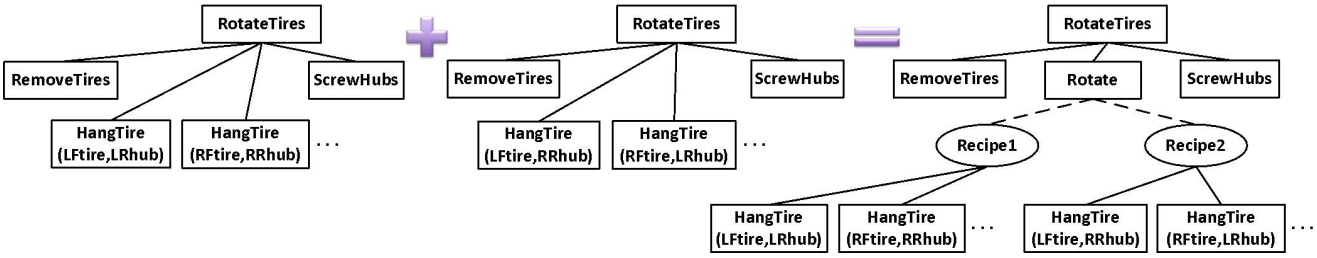

Fig. 6: Example of applying the merge algorithm to RotateTires task (For simplicity, temporal constraints' arrows are not shown.)

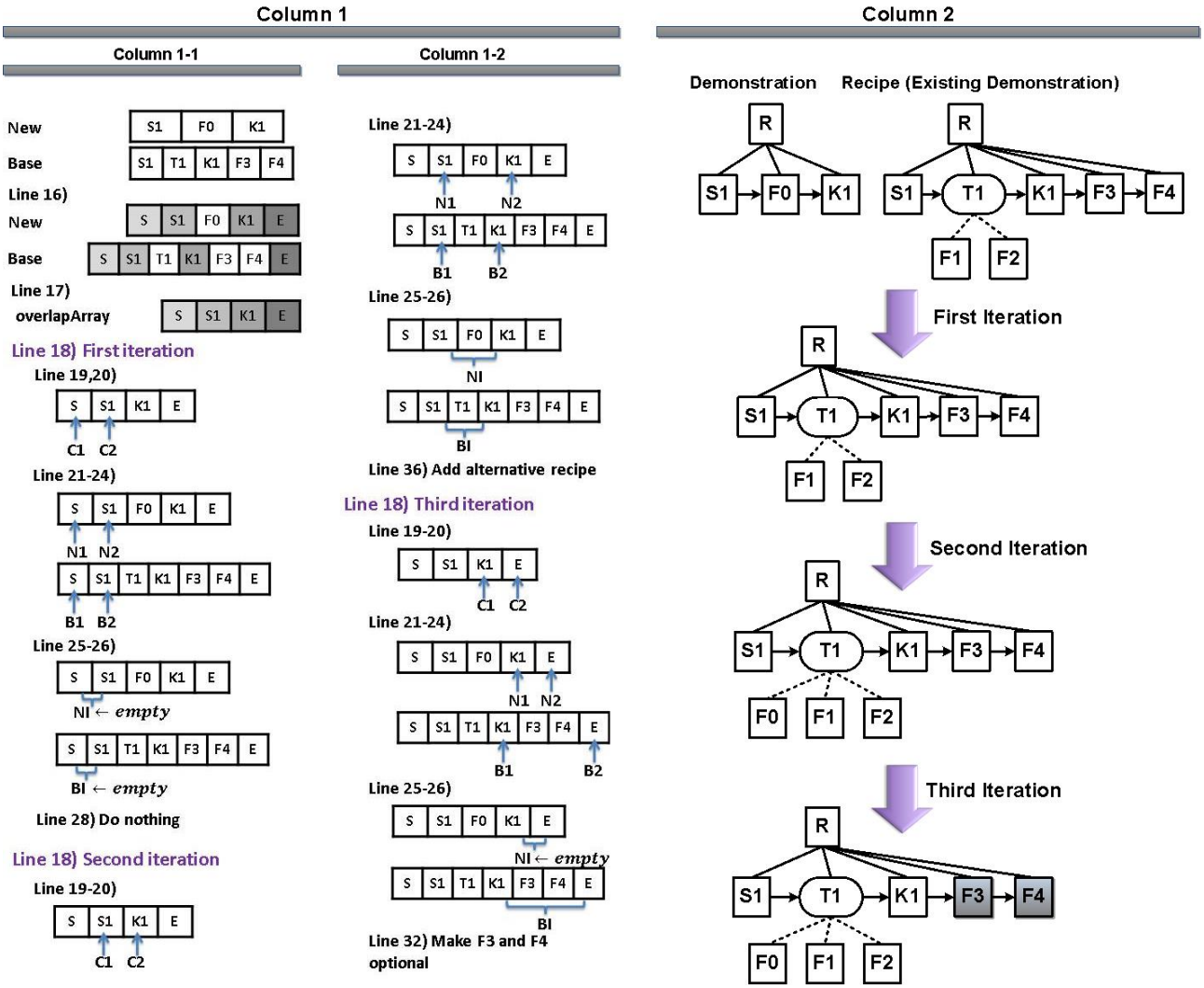

Fig. 7: An example for the execution of the MERGE function in Algorithm 1 is shown in column 1. Column 2 shows the changed HTN after each iteration in column 1 (In column 2, the oval with dashed lines attached to it shows alternative recipes for that specific task and gray boxes show optional steps.).

process. In future work, we will study the relative tradeoffs of presenting each type of question to the user.

V. Evaluation

We evaluated our complete system in a preliminary pilot study using the Gazebo simulation of the car maintenance domain. Specifically, the robot was taught the tire rotation task presented in Figure 3, which consists of first removing the tires by uncrewing and unhanging the hubs, then rotating the tires in one of two patterns, hanging the tires and then screwing them on. Tire rotation was chosen because the task is relatively simple, requiring only six unique primitive actions (`PickUp`, `PutDown`, `Hang`, `Unhang`, `Screw`, `Unscrew`), but highlights the benefits of using the HTN representation, including alternative recipes, hierarchy and inputs/outputs.

The results of this study shows that using the methods described in this paper, we are able to teach the complete task structure in 26 demonstration steps. The interaction between the user and the robot includes 7 user demonstrations, 4 user instructions and 11 robot questions. The final task model is equivalent to the HTN shown in Figure 3.

Importantly, we note that the complete execution of the tire rotation task requires 64 steps. Thus, remarkably, we are able to teach not only one, but two, ways of performing this task (i.e., both variants of `Rotate`) in fewer steps than it takes to perform the task once all the way through.

VI. Conclusion

In this paper, we presented an approach that integrates HTNs and collaborative discourse theory into learning from demonstration paradigm to enable robots to learn complex procedural tasks from human users through mixed-initiative interaction. Our work contributes novel techniques for learning a generalizable task representation from a small number of demonstration, generalization across multiple demonstrations through merging, and integration of question asking into the learning process. We evaluated our approach in a pilot study using simulated robotic environment. Our future work will include user study evaluation, as well as extension to real-world domains using the PR2 robot.

Acknowledgement

This work is supported in part by ONR contract N00014-13-1-0735. The opinions expressed in this document are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.

References

[1] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[2] Maya Cakmak and Andrea L Thomaz. Designing robot learners that ask good questions. In *ACM/IEEE International Conference on Human-Robot Interaction*, pages 17–24. ACM, 2012.

[3] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1):1, 2009.

[4] Andrew Garland, Kathy Ryall, and Charles Rich. Learning hierarchical task models by defining and refining examples. In *International Conference on Knowledge Capture*, pages 44–51, 2001.

[5] Barbara J. Grosz and Candace L. Sidner. Attention, intentions, and the structure of discourse. *Comput. Linguist.*, 12(3):175–204, July 1986.

[6] Scott B. Huffman and John E. Laird. Flexibly instructable agents. *Journal of Artificial Intelligence Research*, 3:271–324, 1995.

[7] Shiwali Mohan and John E Laird. Towards situated, interactive, instructable agents in a cognitive architecture. In *AAAI Fall Symposium Series*, 2011.

[8] Anahita Mohseni-Kabir, Charles Rich, and Sonia Chernova. Learning partial ordering constraints from a single demonstration. In *ACM/IEEE International Conference on Human-robot interaction*, pages 248–249, 2014.

[9] Monica N Nicolescu and Maja J Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *AAMAS*, pages 241–248, 2003.

[10] Scott Niekum, Sarah Osentoski, George Konidaris, and Andrew G Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 5239–5246, 2012.

[11] Charles Rich. Building task-based user interfaces with ANSI/CEA-2018. *IEEE Computer*, 42(8):20–27, 2009.

[12] Charles Rich, Candace L Sidner, and Neal Lesh. Collagen: applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22 (4):15, 2001.

[13] Paul E Rybski, Kevin Yoon, Jeremy Stolarz, and Manuela M Veloso. Interactive robot task training through dialog and demonstration. In *ACM/IEEE Int. Conf. on Human-Robot Interaction*, pages 49–56, 2007.

[14] Harini Veeraraghavan and Manuela Veloso. Learning task specific plans through sound and visually interpretable demonstrations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2599–2604, 2008.