Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Experience-Based Robot Task Learning and Planning with Goal Inference

Vahid Mokhtari, Luís Seabra Lopes and **Armando J. Pinho**
IEETA, University of Aveiro, Aveiro, Portugal
{mokhtari.vahid, lsl, ap}@ua.pt

Abstract

Learning and deliberation are required to endow a robot with the capabilities to acquire knowledge, perform a variety of tasks and interactions, and adapt to open-ended environments. This paper explores the notion of *experience-based planning domains* (EBPDs) for task-level learning and planning in robotics. EBPDs rely on methods for a robot to: (*i*) obtain robot activity experiences from the robot's performance; (*ii*) conceptualize each experience to a task model called *activity schema*; and (*iii*) exploit the learned activity schemata to make plans in similar situations. Experiences are episodic descriptions of plan-based robot activities including environment perception, sequences of applied actions and achieved tasks. The conceptualization approach integrates different techniques including deductive generalization, abstraction and feature extraction to learn activity schemata. A high-level task planner was developed to find a solution for a similar task by following an activity schema. In this paper, we extend our previous approach by integrating goal inference capabilities. The proposed approach is illustrated in a restaurant environment where a service robot learns how to carry out complex tasks.

Introduction

Robots are today leaving industrial environments and start becoming part of our everyday life. This evolution raises the major challenge of personalizing the programming of our robots and making them compatible with human-inhabited environments. Despite the impressive results of manual robot programming, handcrafted solutions are not likely to transfer to the large variety of tasks and environmental states. A new approach seems essential to *learn* the appropriate behavior in many situations. We are interested here in autonomous intelligent robots that are able to *interact* with their environment, *acquire knowledge* to adapt to dynamic and changing environments, and *act deliberately* in order to achieve their mission. Acting deliberately means performing actions that are selected based on reasoning motivated by some intended objectives. Deliberation endows autonomous intelligent robots with adaptability and robustness (Ingrand and Ghallab 2015). This paper presents an approach to endow a robot with capabilities to adapt to changes in its environment and tasks, and to improve its models and behaviors.

The work described in this paper is based on the general idea of *learning from experience*, in which a human user instructs a robot how to perform complex tasks, and allows the robot to conceptualize its experiences. More specifically, the main contribution of this work is the development and demonstration of a task learning and planning system for complex robotic applications. This system integrates previously developed methods as well as new techniques. We propose a learning system that allows for: (*i*) verbally instructing a robot how to carry out complex tasks using a set of predefined primitive behaviors and previously learned concepts; (*ii*) extracting plan-based robot activity experiences from working memory contents; (*iii*) conceptualizing robot activity experiences in the form of *activity schemata*; and (*iv*) to close the loop, exploiting activity schemata to make plans for similar situations. This learning system aims to enable a service robot to learn complex task models in a restaurant environment. We assume that the robot is equipped with a set of basic skills, e.g., `move_base`, `pick_up_object`, `place_object` etc. and we focus on a strategy that would help the robot to construct a high-level task representation of a complex task (e.g., serve a coffee to a guest) built from the existing behavior set.

An activity schema is a learned goal-directed task model consisting of a sequence of abstract operators, a goal to be achieved during problem solving, and the features of objects involved in an experience. Activity schemata are learned for two main purposes. On one hand, they are used for grounding task vocabulary used by the human user. This way the user's language is transmitted to the robot. Activity schemata also capture normative principles about how a task should be achieved (e.g., although a guest can be served on the left side, s/he should be served on the right by social convention). On the other hand, for complex tasks, an activity schema can serve as a guide for speeding up search.

Figure 1 provides an overview of the developed learning and planning system. *Working memory* is an RDF[1] database where different modules share information. Every unit of data written to the working memory is a predicate with temporal extent, called *fluent* (Hertzberg et al. 2014). The *user interface* allows a human to instruct a robot how to perform

[1]http://www.w3.org/RDF/ "Resource Description Framework (RDF)"

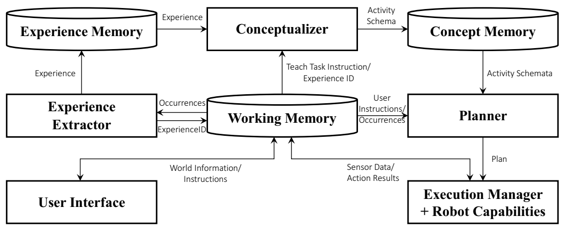

Figure 1: An overview of the learning and planning system.

a complex task using a set of primitive operators (i.e., basic behaviors) as well as teach that task to the robot (Lim et al. 2014). During task execution, the *experience extractor* extracts a subset of fluents including the sequence of applied actions, from working memory. This information is then recorded as a plan-based robot activity experience into the *experience memory* (Mokhtari et al. 2016). The *conceptualizer* receives an experience identifier, takes the respective description from experience memory, and constructs and records an activity schema into the *concept memory*. The role of experience memory and concept memory in this system is similar to that of episodic memory and semantic memory in cognitive science (Wood, Baxter, and Belpaeme 2012). Activity schemata are abstract semantic structures that are used later during planning to find solutions for similar problems. The developed *planner* uses a modified A^* heuristic search approach that takes a ground task as well as the initial state of the environment as inputs and follows a task relevant activity schema to find a solution to the given task. The *execution manager* receives the plan generated by the planner and dispatches the planned actions to the robot platform, and records success or failure information into the working memory (Konečný et al. 2014). The conceptualizer and planner are addressed in this paper.

In this paper, we present several improvements and extensions to our earlier works, one focused on experience extraction and conceptualization (Mokhtari et al. 2016) and the other on formalization of experience-based planning domains and planning capabilities (Mokhtari et al. 2015). In particular, the main novelty of this paper with respect to the previous works is the inclusion of goal inference capabilities and the improvement of feature extraction and planning to take into account the inferred goal. Moreover, we present a more extensive evaluation of the learning and planning system. We demonstrate the performance of this system in a real environment, where a service robot performs a given task within a set of restaurant scenarios. The experiments are carried out both in a gazebo simulated environment and on a real *PR2* robot.

Related work

Task planning faces two main problems. On one hand, if the task is complex, some extra knowledge on how to plan the task is needed to speed up the search for a solution. On the other hand, when there are different alternatives to achieve a goal, some alternatives may be preferable based on different factors, such as social norms, physical constraints, etc. Therefore, task planning systems often require generic task models that specify how tasks should be correctly achieved and/or guide the search for a task plan. The focus of this paper is precisely on the acquisition and exploitation of these task models.

Some previous research has focused on interaction capabilities for teaching high-level task knowledge (Rybski et al. 2007; Mohseni-Kabir et al. 2015). Here, the teacher uses a convenient communication mechanism to teach action compositions, i.e., to directly transfer high-level planning knowledge to the robot. However, the acquired models lack flexibility to be adapted to slightly different variations. Moreover, these approaches do not involve learning from experienced episodes.

Much of the work has focused on research into *learning from demonstration* (LfD) and artificial cognitive systems to quickly transfer task models to a robot (Billard et al. 2008; Argall et al. 2009; Mugan and Kuipers 2012). Nicolescu and Mataric (2003) present an LfD approach to enable a robot to learn and generalize complex tasks from multiple demonstrations. A Longest Common Subsequence between different topological representations of tasks is computed for representing a common sequence of behaviors as an abstract behavior network.

In some approaches that learn task models through interaction with human tutors, the learned models represent the achieved goals and not the actions required. She et al. (2014) identify the goal simply as a difference between the start and end states of an experience guided through human-robot dialog. In another approach, the task model includes, not only the goal, but also a set of ordering constraints between actions (Ekvall and Kragic 2008). This representation is an alternative to the DAG representations proposed by others.

Overall, the learning techniques used so far in task learning from instructions and/or demonstrations are characterized by poor expressivity of the adopted representations as well as by limitations of the generalization techniques. The inference of task goals is either missing or too simplistic. Moreover, the exploitation of the learned knowledge by a planner is given little attention.

The application of classical Artificial Intelligence (AI) techniques for knowledge representation, planning and learning are likely to significantly improve the capabilities of experience-based planning systems. Numerous methods have been proposed in AI for improving planners by acquiring different types of planning knowledge. These efforts are usually concerned with planning speed, but may be relevant for other purposes (Zimmerman and Kambhampati 2003; Jiménez et al. 2012; Ingrand and Ghallab 2015). *Macro-operators* are among the first attempts to improve planning systems by learning compound actions from individual actions frequently used together (Fikes, Hart, and Nilsson 1972; Chrpa 2010). Some works have addressed how to infer a *generalized plan* which works over all instances of a class of problems by efficiently instantiating plans for given problems (Hu and De Giacomo 2011). Other works have focused on learning *Hierarchical Task Networks* (HTN) for hierarchically representing planning knowledge about a problem domain (Hogg, Muñoz-Avila, and Kuter 2014). Although these approaches have seldom been used in robotics (Ingrand and Ghallab 2015), they use more expressive representations and more advanced algorithms than those used so far in robot learning from demonstration, thus we see high application potential.

In this work, we assume an equipped robot with a set of basic skills, and develop a strategy for constructing a high-level task representation of a complex task, built from the existing behavior set. Our approach learns a task model from a single task demonstration. An operators abstraction hierarchy is employed to support more generic concepts, in contrast to other LfD approaches that generalize from multiple demonstrations. We present a method for inferring the goal in a generalized experience, and a mechanism of generating a plan from a learned task model for a given task problem.

Representation

A formal definition of *Experience-Based Planning Domain* (EBPD), i.e., a planning domain that evolves through learning from experiences is proposed in this section.

Definition 1 *An* EBPD *is a tuple,*

$$\mathcal{D} = (\mathcal{L}, \Sigma, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E}, \mathcal{M}),$$

where $\mathcal{L}$ is a first-order logic language that has finitely many predicate and constant symbols, Σ is a set of ground atoms of $\mathcal{L}$ that are always true (i.e., static world information), $\mathcal{S}$ is a set of states in which every state $s \in \mathcal{S}$ is a set of ground atoms of $\mathcal{L}$ which may become false (i.e., transient world information), $\mathcal{A}$ is a set of abstract operators, $\mathcal{O}$ is a set of planning operators, $\mathcal{E}$ is a set of plan-based robot activity experiences, and $\mathcal{M}$ is a set of methods in the form of activity schemata[2].

Planners usually either retain the previously generated world states or have a method to reproduce them later as needed. For complex planning domains that have a large number of planning operators and a large set of predicates, retaining or reproducing the whole of world states during planning will affect the performance of the planners. A distinction between static and transient world information, already explored in (Seabra Lopes 1999; 2007), allows a planner to only retain or reproduce the transient information in the world states during planning, and retrieve the static information from a global storage when it is required. This idea improves the efficiency of planning in terms of both time and memory.

Definition 2 *An* abstract operator *$a \in \mathcal{A}$ is a class of planning operators, which is described by a* head. *A head takes the form $n(x_1, ..., x_{k\geq 0})$, in which n is the name, and $x_1, ..., x_k$ are the arguments, e.g.* `move(from, to)`.

Definition 3 *A* planning operator *$o \in \mathcal{O}$ is a tuple,*

$$o = (h, a, S, P, E),$$

where h is the operator's head, a is an abstract operator which is the superclass or parent of o, S is the static world information, and P and E are respectively the preconditions and effects of o. A ground instance of an operator is called an action.

[2]We have extended PDDL to represent the proposed EBPDs. To fully grasp the notions of EBPDs, an instance of the definitions and the concepts in these domains are available in https://github.com/mokhtarivahid/icaps2016

Table 1: Primitive and abstract operators in this project.

Abstract operators	**Primitive operators**
move(from, to)	**move_base**(oat, from, gat, to) ; *move a robot to a goal area*
	move_base_blind_to_ma(oat, from, gat, to) ; *move a robot to a goal manipulation area*
	move_base_blind_to_pma(oat, from, gat, to) ; *move a robot to a goal pre-manipulation area*
pick(obj, arm)	**pick_up_object**(obj, arm, on, at, ma, pa, g, gh, tp, op, gp) ; *pick an object with an arm*
place(obj, arm, pa)	**place_object**(obj, arm,at, ma, pa, g, tp, h, on, ogp, ggp) ; *place an object with an arm in a placing area*
nil	**tuck_arms**(olpt, orpt, olp, orp, glpt, grpt, glp, grp) ; *tuck both arms into goal postures*
	move_arms_to_carryposture(olpt, orpt, olp, orp, glp, grp) ; *move both arms to carry posture*
	move_arm_to_side(arm, oapt, oap, gap) ; *move an arm to a goal posture*
	move_torso(torso, otpt, otp, gtpt, gtp) ; *move robot's torso to a goal posture*

Table 1 shows the implemented abstract and planning operators in this work. One planning operator represented in EBPDs is shown in Listing 1.

The process of learning starts with experience gathering. Experiences are episodic descriptions of plan-based robot activities including environment perception, sequences of applied actions and achieved tasks.

Definition 4 *A* plan-based robot activity experience *$e \in \mathcal{E}$ is a triple of ground structures,*

$$e = (t, K, \pi),$$

where t is the head of a task, taught by a human user to a robot, e.g., `serve_coffee(guest1)`, *K is a set of key fluents, i.e., propositions in the form of first-order binary predicates with qualitative timestamps, and π is a plan, i.e., a sequence of applied actions to achieve t.*

The timestamps specify the temporal extent of the predicates in an experience. Three types of timestamps are used in the representation of fluents, *at_start* (true at the initial state, e.g. `at_start(on(mug1,paelc1))`), *throughout* (always true during the experience, e.g. `throughout(at(guest1,sawt1))`) and *at_end* (true in the final state, e.g. `at_end(on(mug1,pawrt1))`).

Listing 2 shows a plan-based robot activity experience for the "serve a coffee" task. In (Mokhtari et al. 2016) an approach is proposed to extract plan-based robot activity experiences. Extracted experiences are used to acquire activity schemata. An activity schema is a task model obtained from a plan-based robot activity experience. The knowledge stored in an activity schema is about actions, goal to be achieved and features of objects involved in the experience:

```
(:action place_object
 :parameters    (?obj ?arm ?at ?ma ?pa ?grip ?tp ?h
                 ?on ?ogp ?gap)
 :parent        (place (?obj ?arm ?pa))
 :static        (and
                   (instance arm ?arm)
                   (instance gripper ?grip)
                   (hasmanipulationarea ?pa ?ma)
                   (hasgripper ?arm ?grip))
 :precondition (and
                   (instance robotat ?at)
                   (hasarea ?at ?ma)
                   (instance armtosideposture ?ap)
                   (hasarmposture ?arm ?ap)
                   (instance torsoupposture ?tp)
                   (hasgripperposture ?grip ?ogp)
                   (instance ?ogpt ?ogp)
                   (instance holding ?h)
                   (hasgripper ?h ?grip)
                   (haspassiveobject ?h ?obj)
                   (= ?ogpt gripperholdingposture))
 :effect        (and
                   (not (instance holding ?h))
                   (not (hasgripper ?h ?grip))
                   (not (haspassiveobject ?h ?obj))
                   (not (hasgripperposture ?grip ?ogp))
                   (not (instance ?ogpt ?ogp))
                   (new_constant on ?on)
                   (instance on ?on)
                   (hasarea ?on ?pa)
                   (hasphysicalentity ?on ?obj)
                   (new_constant gripperopenposture ?ggp)
                   (instance gripperopenposture ?ggp)
                   (hasgripperposture ?grip ?ggp)))
```

Listing 1: PDDL-like representation of a planning operator. With respect to the standard PDDL, *parent* and *static* are new added properties to the domain.

Definition 5 *An* activity schema $m \in \mathcal{M}$ *is a triple,*

$$m = (h, G, \Omega),$$

where h *is the head of the target task (e.g.* `serve_coffee(guest)`*),* G *is a set of generalized propositions called* goal propositions *that are inferred from the possible relationships between arguments of* h *in an experience, and* Ω *is an abstract plan, i.e., a sequence of abstract operators enriched with features.*

Definition 6 *An* enriched abstract operator ω *is a pair,*

$$\omega = (a, F),$$

where $a \in \mathcal{A}$ *is an abstract operator, and* F *is a set of features, in the form of reified fluents, that documents the arguments of* a*.*

Definition 7 *A* task planning problem *is a triple,*

$$\mathcal{P} = (\sigma, s_0, t),$$

where $\sigma \subseteq \Sigma$ *is the static world information,* $s_0 \in \mathcal{S}$ *is the initial state (i.e., transient world information), and* t *is a task to be planned (e.g.* `serve_coffee(guest1)`*).*

A plan solution π is generated for task t, iff there is a learned activity schema $m \in \mathcal{M}$ in which the head of m matches task t and π can be derived from m in s_0.

Conceptualizing experiences

The term *conceptualization* in this work refers to the process of learning an activity schema from a robot activity experience. The conceptualization approach is a combination of different techniques including deductive generalization, different forms of abstraction, goal inference and feature extraction.

```
(:task serve_coffee
 :parameters       (mug1 guest1)
 :key-propositions (....) ; omitted
 :actions
   ((tuck_arms aunp aunp aunp1 aunp0 atp atp atp7 atp13)
    (move_base at0 fatr1 at7 pmaec1)
    (move_torso torso1 tdp tdp0 tup tup2)
    (tuck_arms atp atp atp7 atp13 atp atp atp1001 atp17)
    (move_arm_to_side rightarm1 atp atp17 asp19)
    (move_base_blind_to_ma at7 pmaec1 at9 maec1)
    (pick_up_object mug1 rightarm1 on4 at9 maec1 paerc1
                    rg1 h1 tup2 asp19 asp27)
    (move_base_blind_to_pma at9 maec1 at11 pmaec1)
    (move_torso torso1 tup tup2 tdp tdp4)
    (move_arms_to_carryposture atp asp atp1001 asp27
                               acp31 acp33)
    (move_base at11 pmaec1 at13 pmast1)
    (move_torso torso1 tdp tdp4 tup tup6)
    (move_arm_to_side rightarm1 acp acp33 asp35)
    (move_base_blind_to_ma at13 pmast1 at16 mast1)
    (place_object mug1 rightarm1 at16 mast1 on3 pawrt1
                  rg1 h1 tup6 asp35 asp42)
    (move_base_blind_to_pma at16 mast1 at19 pmast1)))
```

Listing 2: A plan-based robot activity experience for serve a coffee task. It contains 16 primitive actions and 122 key propositions.

Generalization and abstraction

Generalization is the first step in conceptualizing a plan-based robot activity experience. Through deductive generalization, it is possible to formulate general concepts from single training examples and domain knowledge. A *goal regression algorithm*, as in explanation-based generalization (EBG) (Mitchell, Keller, and Kedar-Cabelli 1986), is employed to: (*i*) build an explanation of how a plan-based robot activity experience is solved with domain operators; and (*ii*) generalize the obtained explanation. The generalization is carried out by variabilizing the observed constants in an experience and propagating the substitution of constants for variables in the whole experience.

To reduce the level of detail in a generalized experience, an operator abstraction hierarchy was developed, which results in more widely applicable task knowledge. This hierarchy is specified in an EBPD using the parent property of planning operators which links to an abstract operator (Table 1). In this abstraction hierarchy, some operators are mapped onto abstract operators, some others are mapped onto *nil*, meaning they are excluded from the learned activity schema, and some arguments of operators are excluded in the respective abstract operators. The *nil* class of operators contains auxiliary operators that are filled later during instantiation of a learned activity schema for a given problem.

Generalization and abstraction allow a planner to generate plans with different objects, different sequences of actions, and different plan lengths.

Goal inference

The ability to recognize, formulate, select, and manage goals/objectives is of great interest to intelligent robotics. Recognizing the goal of a task in an experience enables to better understand the experience and to better capture the essential aspects of the applied solution. We have developed a goal inference procedure which extracts a set of goal propositions

Algorithm 1 Inferring goal propositions in an experience

```
Input:
    Task arguments $A$ and key propositions $K$ in a generalized experience
Output:
    A set of generalized goal propositions $G$
 1: procedure GOALINFERENCE($A, K$)
 2:     if $A = \phi$ then
 3:         return $\phi$
 4:     $G \leftarrow$ GOALINFERENCE($tail(A)$,$K$)
 5:     for all $b$ in $tail(A)$ do
 6:         $P \leftarrow$ FINDPATHS($head(A)$,$b$,$K$,$\phi$)
 7:         $G \leftarrow G \cup$ SHORTESTPATH($P$)
 8:     return $G$

 1: procedure FINDPATHS($a, b, K, p$)
 2:     if $a = b$ then
 3:         return $\{p\}$
 4:     $F \leftarrow$ fluents in $K$ containing $a$ as argument
 5:     $P \leftarrow \phi$
 6:     for all $f$ in $F$ do
 7:         $x \leftarrow$ the other argument of $f$          ▷ $x \neq a$
 8:         if $x$ does not appear in $p$ then
 9:             $P \leftarrow P \cup$ FINDPATHS($x$,$b$,$K$,$p \cup \{f\}$)
10:     return $P$
```

from the final state of a generalized experience (i.e., from the set of key propositions with timestamps `throughout` and `at_end`). This inferred goal will be useful in solving different instances of the same problem. The arguments of a taught task, identified by an instructor (e.g., `mug1`, `guest1` etc.), define the main targets for the task, and are used as cues for inferring goal propositions. Algorithm 1 describes the process of goal inference. The GOALINFERENCE procedure takes as input the set of generalized arguments and the set of generalized key propositions in an experience. For every pair of task arguments, key propositions with timestamps `throughout` and `at_end` are explored to find all possible paths between the arguments. Each path is a set of propositions that reveals a relation between two task arguments in the final state of the experience. For each pair of arguments, the shortest connecting path is selected. Finally, the union of the shortest paths computed for all pairs of arguments is used as a description of the goal. Figure 2 shows some of the possible paths between two task arguments `mug` and `guest` in an experience. For the purpose of feature extraction, the `at_end` key propositions in the generalized experience are replaced by the extracted goal propositions with `at_goal` timestamps.

Feature extraction

Concepts derived by generalization and abstraction are enriched through feature extraction. Features are properties of objects involved in an experience. During planning, features define preferences for finding the closest instantiation of an activity schema to a given task problem. In the current conceptualization approach, a *feature* is a fluent or a set of fluents, which describes a relation between an argument of an action in the experience and an argument of the taught task. The key propositions in a generalized and abstracted robot

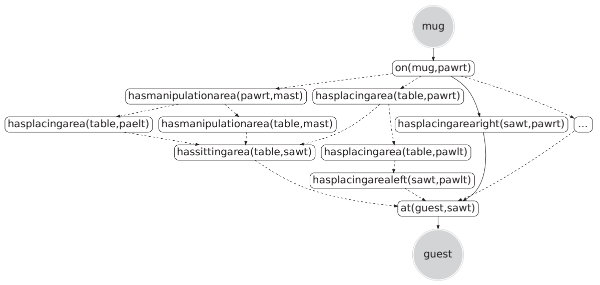
mug
on(mug,pawrt)
hasmanipulationarea(pawrt,mast)
hasplacingarea(table,pawrt)
hasplacingarea(table,paelt)
hasmanipulationarea(table,mast)
hasplacingarearight(sawt,pawrt)
...
hassittingarea(table,sawt)
hasplacingarea(table,pawlt)
hasplacingarealeft(sawt,pawlt)
at(guest,sawt)
guest

Figure 2: A small part of the paths between the task arguments `mug` and `guest` in a generalized experience `serve_coffee(mug, guest)`. A total of 62 paths were found in this experience. The shortest path identifies the set of propositions, which describes the goal (solid lines).

activity experience are used for discovering potentially relevant features. For this purpose, all possible one-step and two-step relationships between the arguments of abstract operators and the arguments of the taught task are extracted.

The set of features, F, for an abstract operator a in a generalized and abstracted experience of a task t with key propositions K is computed as:

$$F(a, t, K) = F_1(a, t, K) \cup F_2(a, t, K) \qquad (1)$$

where F_1 and F_2 are sets of one-step and two-step features respectively:

$$\begin{aligned} F_1(a, t, K) = \Big\{ \tau\big(p(x, y)\big) \in K \;\Big|\; p \in \mathcal{L}, \\ (x \in A \wedge y \in B) \vee (x \in B \wedge y \in A) \Big\}, \end{aligned} \qquad (2)$$

$$\begin{aligned} F_2(a, t, K) = \bigg\{ \Big(\tau_1\big(p(x, z)\big), \tau_2\big(q(z, y)\big) \Big) \in K^2 \bigg| \\ p, q \in \mathcal{L}, (x \in A \wedge y \in B) \vee (x \in B \wedge y \in A) \bigg\}, \end{aligned} \qquad (3)$$

where τ, τ_1 and τ_2 are qualitative timestamps (i.e., *at_start*, *throughout* and *at_goal*), A is the set of arguments of a, and B is the set of arguments of t.

For every abstract operator in an experience, a set of possible features is discovered and added to the corresponding abstract operator. During planning, features help a planner to select actions that best match the abstract operators in a used activity schema. They also reduce the probability of backtracking in a search tree, thus speeding up planning. During planning, the percentage of features of the abstract operators in the adopted activity schema that are verified in the actions applicable in the target problem determines the extent of similarity of the produced plan to the used activity schema. Listing 3 shows an example of a learned activity schema after generalization, abstraction, goal inference and feature extraction.

Planning based on activity schemata

An adapted A^* heuristic search planner, *Schema-Based Planner* (SBP), was developed to find a solution for a given

```
1.(:method serve_coffee
2. :parameters  (?mug ?guest)
3. :goal   (and (on ?mug ?pawrt)
4.              (hasplacingarearight ?sawt ?pawrt)
5.              (at ?guest ?sawt))
6. :abstract-plan
7.  ((!move ?fatr ?pmaec) (()-())
8.   (!move ?pmaec ?maec) (()-
9.       ((at_start(on ?mug ?paerc))
10.       (throughout(hasmanipulationarea ?paerc ?maec))))
11.   (!pick_up ?mug ?paerc)
12.       (((at_goal(on ?mug ?pawrt))
13.         (at_start(on ?mug ?paerc)))-
14.        ((throughout(hasplacingarearight ?sawt ?pawrt))
15.         (at_goal(on ?mug ?pawrt))))
16.  (!move ?maec ?pmaec) (()-())
17.  (!move ?pmaec ?pmast) (()-())
18.  (!move ?pmast ?mast) ((....)-())          ; omitted
19.  (!place ?mug ?pawrt) ((....)-             ; omitted
20.      ((throughout(hasplacingarearight ?sawt ?pawrt))
21.       (throughout(at ?guest ?sawt))))
22.  (!move ?mast ?pmast) (()-(....))))        ; omitted
```

Listing 3: A learned activity schema for serving a coffee to a guest containing 8 abstract operators. Lines 3-5 show a generalized goal which is instantiated based on a given task problem and should be achieved during planning. Some abstract operators are enriched with features. During planning, these features give preferences to find the best matched action to the one in the activity schema. For instance, features on lines 20-21 help the SBP planner to prefer an action, among all applicable actions belonging to the class of *place*, that closely matches these features in a given task problem.

Algorithm 2 Schema-Based Planner (SBP)

```
Input:
    a planning domain $\mathcal{D} = (\mathcal{L}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E}, \mathcal{M})$    ▷ Def. 1
    a planning problem $\mathcal{P} = (\sigma, s_0, t)$    ▷ Def. 7
Output:
    a plan $\pi$
1:  procedure SBP($\mathcal{D}, \mathcal{P}$)
2:      if there is no schema in $\mathcal{M}$ for task $t$ then
3:          return failure
4:      $(h, G, \Omega_0) \leftarrow$ select a schema from $\mathcal{M}$ for task $t$
5:      $instantiate(h, G, \mathcal{P})$    ▷ instantiate $h$ and $G$ based on $\mathcal{P}$
6:      $Open \leftarrow \{(s_0, \phi, \Omega_0, 0, 0, \alpha \cdot length(\Omega_0))\}$
7:      while $Open \neq \phi$ do
8:          $(s, \pi, \Omega, f, g, h) \leftarrow$ pick a node with lowest $f$ from $Open$
9:          if ($\Omega = \phi \land G$ is satisfied in $s$) then
10:             return $\pi$
11:         $\omega \leftarrow head(\Omega)$
12:         $A \leftarrow$ {all applicable actions from classes $nil$ and $\omega$}
13:         for all $a \in A$ do
14:             $s_n \leftarrow \gamma(s, a)$    ▷ transition function
15:             $\pi_n \leftarrow \pi.a$
16:             $\Omega_n \leftarrow \Omega$
17:             if $a$ belongs to the class of $\omega$ then
18:                 $k \leftarrow$ number of features in $\omega$
19:                 $v \leftarrow$ number of features in $\omega$ verified for $a$
20:                 $g_n \leftarrow g + c_{real} \cdot (k+1)/(v+1)$    ▷ $c_{real} = 1$
21:                 $\Omega_n \leftarrow tail(\Omega)$
22:             else    ▷ $a$ belongs to the class of $nil$
23:                 $g_n \leftarrow g + c_{real}$
24:             $h_n \leftarrow \alpha \cdot length(\Omega_n)$    ▷ $\alpha = 2.5$
25:             $f_n = g_n + h_n$
26:             $n \leftarrow (s_n, \pi_n, \Omega_n, f_n, g_n, h_n)$
27:             if no node with same $s$, $\Omega$ and lower $f$ is in $Open$ then
28:                 $Open \leftarrow$ add $n$
29:     return failure
```

problem. This algorithm searches forward from the initial state of the world by following an activity schema and tries to reach the end of the activity schema where a given task is achieved. Searching by following an activity schema is the key difference with respect to standard search.

The SBP planner takes as input a planning domain $\mathcal{D} = (\mathcal{L}, \Sigma, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E}, \mathcal{M})$ and a task planning problem $\mathcal{P} = (\sigma, s_0, t)$. If there is an activity schema for task t, the planner attempts to generate a plan for t; otherwise it fails. The SBP planner is presented in Algorithm 2. SBP begins by selecting an activity schema $(h, G, \Omega_0) \in \mathcal{M}$ corresponding to a given task t (line 4). The task signature h and the goal G are instantiated based on the given problem P (line 5), and this instantiation is also propagated into the abstract plan Ω_0. Each node in the search tree retains a state, s, the plan built so far, π, the remaining part of the abstract plan, Ω, and the costs f, g and h. Line 6 creates the initial node. In each planning iteration, a leaf node with the lowest f cost is retrieved (line 8). If the current plan is not yet complete, the planner selects actions belonging to the class of the first abstract operator, ω, in the abstract plan, Ω, as well as auxiliary actions from the nil class (line 12). If action a belongs to the class of ω (lines 17-21), the accumulated cost g_n is computed taking into account the features in ω verified and not verified in a. Suppose k is the total number of features in ω and v is the number of these features that are verified for action a. In this case, g_n is given by:

$$g_n = g + c_{real} \cdot (k+1)/(v+1), \tag{4}$$

where c_{real} is the real cost of a (e.g. $c_{real} = 1$). This means that applicable actions belonging to the class of ω which verify all features in ω, gain the real cost. The lower the percentage of verified features, the higher will be the cost. If action a belongs to the nil class, the cost is always c_{real} for this action (lines 22-23). This means if there is no applicable action belonging to the class of ω and verifying all features, actions in the nil class may be preferred for expansion. In all cases, the heuristic h_n is estimated as the length of the remaining abstract plan Ω_n multiplied by a factor α that estimates the average number of actions in a plan per operator in the abstracted version of the plan (line 24). Based on empirical tests, we use $\alpha = 2.5$ for our test domain. Finally, the cost function is computed as $f_n = g_n + h_n$ for each new node, n. Based on the selected actions and resulting states and costs, new nodes are created and added to the search tree (lines 27-28). The planner stops when it retrieves a node with an empty Ω and a state s satisfying the goal G (lines 9-10). Figure 3 illustrates how an activity schema guides a planner to progress towards generating a plan.

Experimental results

The learning approach was demonstrated and evaluated in two different tasks, “serve a coffee” and “clear a table”. To learn an activity schema, a plan-based robot activity experience is generated through human-robot interaction. This

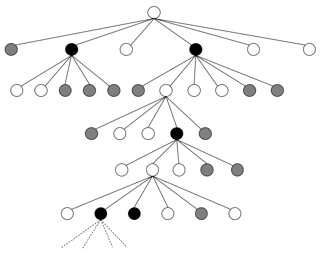

Figure 3: Illustration of how the SBP planner follows an activity schema. Black and gray nodes are generated by operators which have as parents the abstract operators in the activity schema. Gray nodes are generated by operators that don't verify all features, while black nodes verify all features. White nodes are generated by operators that belong to the *nil* class. The planner always expands the black nodes in the highest priority. White nodes are preferred for expansion when there is no black node to expand. Gray nodes are expanded in the lowest priority.

interaction involves a sequence of instructions to carry out and teach a complex task. An infrastructure for teaching a robot how to perform a task, and extracting a robot activity experience from working memory was presented in (Mokhtari et al. 2016). A "serve a coffee" experience is shown above in Listing 2. The conceptualization is carried out immediately after a plan-based robot activity experience is generated. Listing 3 showed a learned activity schema after generalization, abstraction, goal inference and feature extraction. The proposed conceptualizer was tested with a *PR2* robot in a real environment. An online video of the experience gathering and conceptualization process for the "serve a coffee" task in a real robot *PR2* is available in https://youtu.be/5Z6PJX6Ucfg.

Since an activity schema is learned, it can be followed by the SBP to make plans for similar task problems. The measures of *penetrance*, *effective branching factor* and *precision* are used to evaluate the performance of the planner over the learned activity schemata.

The penetrance ratio, P, of a search is,

$$P = L/X, \tag{5}$$

where L is the length of the plan, and X is the total number of expanded nodes. The penetrance is the extent to which the search has focused toward a goal (Russell and Norvig 2010). It shows how the extracted features guide the planner to avoid expanding irrelevant nodes during the search.

The average branching factor, B, is the average number of successors generated for each node in a search problem,

$$B = (N - 1)/X, \tag{6}$$

where N is the total number of nodes generated during the search.

The effective branching factor is a measure of the heuristic's usefulness. If a method generates N nodes to find a solution of depth d, the effective branching factor of β that

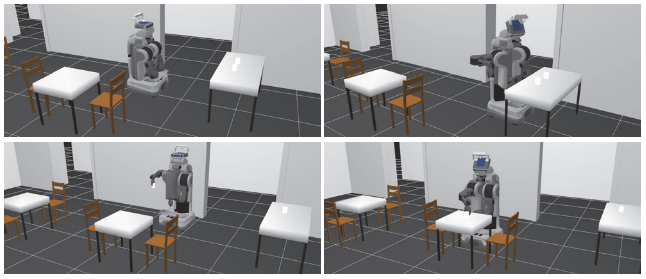

Figure 4: Snapshots of "serve a coffee" scenario with *PR2* robot in gazebo simulated environment. In this scenario, (top-left) robot moves to the `counter1`, (top-right) picks up `mug1` from the counter, (bottom-left) moves to the `table1`, and (bottom-right) puts the mug on the table in front of a guest.

a uniform tree of depth d requires to contain N nodes is,

$$N = (\beta^{(L+1)} - 1)/(\beta - 1), \tag{7}$$

where L is the length of the plan, and N is the total number of nodes generated during the search.

The *precision* metric is used to evaluate the quality of the produced plans, and is defined as,

$$Precision = ca/(ca + wa), \tag{8}$$

where ca is the total number of correct actions, and wa is the total number of wrong actions in the produced plan.

The learning approach was evaluated in the tasks "serve a coffee to a guest" and "clear a table". Since the arguments of the taught task are the cues for inferring the goal as well as for extracting features during conceptualization, we taught several variations of these tasks with different sets of arguments: `srv(guest,counter)`, `srv(mug,guest)`, `srv(mug,guest,counter)`, `clr(table)`, `clr(table,counter)`, and `clr(mug,table,counter)`. For each task, one scenario is set up to teach the robot and three other scenarios are used to evaluate the performance of the planner using that learned task. The test scenarios differ in the initial location and configuration of the robot and in the number of objects and their positions in the world state. Listing 4 sketches the plan executed in the teaching scenario of `srv(mug,guest)`, as well as the plans generated for the three test scenarios.

The performance metrics of the SBP planner for "serve a coffee" and "clear a table" scenarios are presented in Tables 2 and 3 respectively. The penetrance ratio increased and the effective branching factor decreased with respect to the number of task arguments. The activity schemata `srv(g,c)` and `clr(t)` showed lack of precision due to lack of features as well as due to lack of goal propositions (i.e., no goal propositions were inferred for these activity schemata). Since the activity schemata `srv(m,g)`, `srv(m,g,c)` and `clr(m,t,c)` have goal, the obtained plan solutions are the real solutions to the given task problems. It was observed that the extracted features are sufficient for the abstract operators in the activity schema

Table 2: Planner's performance in `serve_coffee`.

Activity schema*	srv(g, c)†			srv(m, g)			srv(m, g, c)		
Experiment	#1	#2	#3	#1	#2	#3	#1	#2	#3
Plan length	15	16	17	15	16	17	15	16	17
Search tree size	2862	5240	16035	1211	1717	2851	877	1451	2557
Nodes expanded	1139	2173	6801	356	448	697	241	345	585
Penetrance (%)	1.32	0.74	0.25	4.21	3.57	2.44	6.22	4.64	2.91
Average branching factor	2.51	2.41	2.36	3.40	3.83	4.09	3.63	4.20	4.37
Effective branching factor	1.51	1.60	1.67	1.49	1.48	1.49	1.45	1.46	1.48
Precision	0.73	0.75	0.94	1.00	1.00	1.00	1.00	1.00	1.00

* *srv* stands for *serve_coffee* task, and *m*, *g* and *c* stand for *mug*, *guest* and *counter*.

† No goal propositions were inferred for these activity schemata.

Table 3: Planner's performance in `clear_table`.

Activity schema*	clr(t)†			clr(t, c)†			clr(m, t, c)		
Experiment	#1	#2	#3	#1	#2	#3	#1	#2	#3
Plan length	15	16	17	15	16	17	15	16	17
Search tree size	2814	2874	3121	1001	1061	1308	1068	1096	1375
Nodes expanded	1136	1145	1216	299	308	379	292	283	372
Penetrance (%)	1.32	1.40	1.40	5.02	5.19	4.49	5.14	5.65	4.57
Average branching factor	2.48	2.51	2.57	3.34	3.44	3.45	3.65	3.87	3.69
Effective branching factor	1.58	1.54	1.50	1.46	1.43	1.42	1.47	1.43	1.42
Precision	0.73	0.75	0.76	1.00	1.00	1.00	1.00	1.00	1.00

* *clr* stands for *clear_table* task, and *m*, *t* and *c* stand for *mug*, *table* and *counter*.

† No goal propositions were inferred for these activity schemata.

`clr(t,c)`. Hence this activity schema fully achieved the given task problems, despite the fact that it was not possible to infer its goal. This system was tested with a *PR2* robot in both a gazebo simulated environment (Figure 4) and a real environment. An online video showing the operation of the proposed SBP planner in a "serve a coffee" scenario in a real *PR2* robot is available in https://youtu.be/mjrP3hiMRnw. The original experiences and the given task problems used in these experiments can be obtained from https://github.com/mokhtarivahid/icaps2016/.

A more detailed description of this work will appear in (Mokhtari, Seabra Lopes, and Pinho 2016).

Conclusion and future work

This paper proposed: (*i*) a formalization of experience-based planning domains; (*ii*) a unified framework for learning and problem solving in these domains; and (*iii*) a goal inference approach that allowed to improve the previously developed conceptualization and planning algorithms. The terms *experience*, *activity schema* and *schema-based planning* are tightly integrated in the proposed framework. The activity schemata are learned task models enriched with features and goal propositions. During planning, features give preferences to achieve the most similar interpretation of the original experience while the inferred goal propositions help to guarantee that the given task is fully achieved. An interesting issue for future work is detecting and analyzing loops of actions in an experience. Identifying loops of actions during conceptualization enables the planner to deal with similar tasks with different number of objects.

```
Original experience of serve_coffee(mug1,guest1) task.

1.  (tuck_arms aunp aunp aunp0 aunp1 atp atp atp7 atp13)
2.  (move_base fatr1 pmaec1)
3.  (move_torso torso1 tdp tdp0 tup tup2)
4.  (tuck_arms atp atp atp7 atp13 autp atp autp5 atp1002)
5.  (move_arm_to_side leftarm1 autp autp5 atsp21)
6.  (move_arm_to_side rightarm1 autp autp9 atsp23)
7.  (move_base_blind_to_ma pmaec1 maec1)
8.  (pick_up_object mug1 rightarm1)
9.  (move_base_blind_to_pma maec1 pmaec1)
10. (move_arms_to_carryposture atsp atsp atsp21 ...)
11. (move_torso torso1 tup tup2 tmp tmp4)
12. (move_base pmaec1 pmast)
13. (move_torso torso1 tmp tmp4 tup tup6)
14. (move_arm_to_side rightarm1 acp acp39 atsp53)
15. (move_base_blind_to_ma pmast mast1)
16. (place_object mug1 rightarm1 pawrt1)
17. (move_base_blind_to_pma mast1 pmast)

-----------------------------------------------------
Experiment #1

1.  (move_base fatr1 pmaec1)
2.  (move_base_blind_to_ma pmaec1 maec1)
3.  (move_torso torso1 tdp tdp0 tup tup89387)
4.  (tuck_arms atp atp atp0 atp1 autp autp autp89401 ...)
5.  (move_arm_to_side leftarm1 autp autp89401 atsp89591)
6.  (pick_up_object mug1 leftarm1)
7.  (move_base_blind_to_pma maec1 pmaec1)
8.  (move_torso torso1 tup tup89387 tmp tmp90693)
9.  (move_arms_to_carryposture atsp autp atsp89591 ...)
10. (move_base pmaec1 pmast)
11. (move_base_blind_to_ma pmast mast1)
12. (move_torso torso1 tmp tmp90693 tup tup90949)
13. (move_arm_to_side leftarm1 acp acp90703 atsp90960)
14. (place_object mug1 leftarm1 pawrt1)
15. (move_base_blind_to_pma mast1 pmast)

-----------------------------------------------------
Experiment #2

1.  (tuck_arms aunp aunp aunp0 aunp1 atp atp atp1981 ...)
2.  (move_base fatr1 pmaec1)
3.  (move_base_blind_to_ma pmaec1 maec1)
4.  (move_torso torso1 tdp tdp0 tup tup2104)
5.  (tuck_arms atp atp atp1981 atp1982 autp autp ...)
6.  (move_arm_to_side leftarm1 autp autp2131 atsp2399)
7.  (pick_up_object mug2 leftarm1)
8.  (move_base_blind_to_pma maec1 pmaec1)
9.  (move_torso torso1 tup tup2104 tmp tmp3906)
10. (move_arms_to_carryposture atsp autp atsp2399 ...)
11. (move_base pmaec1 pmant1)
12. (move_base_blind_to_ma pmant1 mant1)
13. (move_torso torso1 tmp tmp3906 tup tup4162)
14. (move_arm_to_side leftarm1 acp acp3916 atsp4173)
15. (place_object mug2 leftarm1 paert1)
16. (move_base_blind_to_pma mant1 pmant1)

-----------------------------------------------------
Experiment #3

1.  (move_torso torso1 tup tup0 tdp tdp84998)
2.  (tuck_arms aunp aunp aunp0 aunp1 atp atp ...)
3.  (move_base fatr1 pmaec1)
4.  (move_base_blind_to_ma pmaec1 maec1)
5.  (move_torso torso1 tdp tdp84998 tup tup85273)
6.  (tuck_arms atp atp atp85020 atp85021 autp autp ...)
7.  (move_arm_to_side leftarm1 autp autp85326 atsp85750)
8.  (pick_up_object mug3 leftarm1)
9.  (move_base_blind_to_pma maec1 pmaec1)
10. (move_torso torso1 tup tup85273 tmp tmp88067)
11. (move_arms_to_carryposture atsp autp atsp85750 ...)
12. (move_base pmaec1 pmaet2)
13. (move_base_blind_to_ma pmaet2 maet2)
14. (move_torso torso1 tmp tmp88067 tup tup88323)
15. (move_arm_to_side leftarm1 acp acp88077 atsp88334)
16. (place_object mug3 leftarm1 pasrt2)
17. (move_base_blind_to_pma maet2 pmaet2)
```

Listing 4: Sketches of the generated plans for an experience of the `srv(m,g)` task and for three given task problems of the same class. Depending on the given task problems, plans with different lengths and different sequences of actions were generated. Some arguments of actions are omitted due to lack of space.

References

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.

Billard, A.; Calinon, S.; Dillmann, R.; and Schaal, S. 2008. Robot programming by demonstration. In Siciliano, B., and Khatib, O., eds., *Springer Handbook of Robotics*. Springer Berlin Heidelberg. 1371–1394.

Chrpa, L. 2010. Generation of macro-operators via investigation of action dependencies in plans. *The Knowledge Engineering Review* 25(03):281–297.

Ekvall, S., and Kragic, D. 2008. Robot Learning from Demonstration: A Task-level Planning Approach. *International Journal of Advanced Robotic Systems* 5(3).

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial intelligence* 3:251–288.

Hertzberg, J.; Zhang, J.; Zhang, L.; Rockel, S.; Neumann, B.; Lehmann, J.; Dubba, K.; Cohn, A.; Saffiotti, A.; Pecora, F.; Mansouri, M.; Konečný, Š.; Günther, M.; Stock, S.; Seabra Lopes, L.; Oliveira, M.; Lim, G.; Kasaei, H.; Mokhtari, V.; Hotz, L.; and Bohlken, W. 2014. The race project. *KI - Künstliche Intelligenz* 28(4):297–304.

Hogg, C.; Muñoz-Avila, H.; and Kuter, U. 2014. Learning hierarchical task models from input traces. *Computational Intelligence*.

Hu, Y., and De Giacomo, G. 2011. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, 918–923.

Ingrand, F., and Ghallab, M. 2015. Deliberation for autonomous robots: A survey. *Artificial Intelligence*.

Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27:433–467.

Konečný, Š.; Stock, S.; Pecora, F.; and Saffiotti, A. 2014. Planning domain + execution semantics: a way towards robust execution? In *Qualitative Representations for Robots, AAAI Spring Symposium*.

Lim, G. H.; Oliveira, M.; Mokhtari, V.; Hamidreza Kasaei, S.; Chauhan, A.; Seabra Lopes, L.; and Tome, A. 2014. Interactive teaching and experience extraction for learning about objects and robot activities. In *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*, 153–160.

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

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*, HRI '15, 205–212. New York, NY, USA: ACM.

Mokhtari, V.; Seabra Lopes, L.; Pinho, A. J.; and Lim, G. H. 2015. Planning with activity schemata: Closing the loop in experience-based planning. In *Autonomous Robot Systems and Competitions (ICARSC), 2015 IEEE International Conference on*, 9–14.

Mokhtari, V.; Lim, G.; Seabra Lopes, L.; and Pinho, A. 2016. Gathering and conceptualizing plan-based robot activity experiences. In Menegatti, E.; Michael, N.; Berns, K.; and Yamaguchi, H., eds., *Intelligent Autonomous Systems 13*, volume 302 of *Advances in Intelligent Systems and Computing*. Springer International Publishing. 993–1005.

Mokhtari, V.; Seabra Lopes, L.; and Pinho, A. 2016. Experience-based planning domains: An integrated learning and deliberation approach for intelligent robots. *Journal of Intelligent & Robotic Systems (to appear)*.

Mugan, J., and Kuipers, B. 2012. Autonomous learning of high-level states and actions in continuous environments. *IEEE Transactions on Autonomous Mental Development (TAMD)* 4(1):70–86.

Nicolescu, M. N., and Mataric, M. J. 2003. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 241–248. ACM.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.

Rybski, P.; Yoon, K.; Stolarz, J.; and Veloso, M. 2007. Interactive robot task training through dialog and demonstration. In *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, 49–56.

Seabra Lopes, L. 1999. Failure recovery planning in assembly based on acquired experience: Learning by analogy. In *Assembly and Task Planning, 1999.(ISATP'99) Proceedings of the 1999 IEEE International Symposium on*, 294–300. IEEE.

Seabra Lopes, L. 2007. Failure recovery planning for robotized assembly based on learned semantic structures. In *IFAC Workshop on Intelligent Assembly and Disassembly (IAD'2007)*, 65–70.

She, L.; Yang, S.; Cheng, Y.; Jia, Y.; Chai, J. Y.; and Xi, N. 2014. Back to the blocks world: Learning new actions through situated human-robot dialogue. In *15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 89.

Wood, R.; Baxter, P.; and Belpaeme, T. 2012. A review of long-term memory in natural and synthetic systems. *Adaptive Behavior* 20(2):81–103.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine* 24(2):73–96.