Synthesizing Manipulation Sequences for Under-Specified Tasks using Unrolled Markov Random Fields

Jaeyong Sung, Bart Selman and Ashutosh Saxena

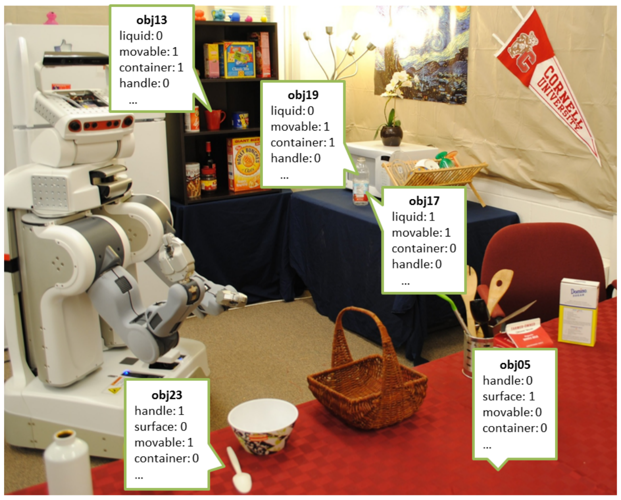
***Abstract*— Many tasks in human environments require performing a sequence of navigation and manipulation steps involving objects. In unstructured human environments, the location and configuration of the objects involved often change in unpredictable ways. This requires a high-level planning strategy that is robust and flexible in an uncertain environment. We propose a novel dynamic planning strategy, which can be trained from a set of example sequences. High level tasks are expressed as a sequence of primitive actions or controllers (with appropriate parameters). Our score function, based on Markov Random Field (MRF), captures the relations between environment, controllers, and their arguments. By expressing the environment using sets of attributes, the approach generalizes well to unseen scenarios. We train the parameters of our MRF using a maximum margin learning method. We provide a detailed empirical validation of our overall framework demonstrating successful plan strategies for a variety of tasks.**[1]

I. Introduction

When interacting with a robot, users often under-specify the tasks to be performed. For example in Figure 5, when asked to `pour` something, the robot has to infer which cup to pour into and a complete sequence of the navigation and manipulation steps—moving close, grasping, placing, and so on.

This sequence not only changes with the task, but also with the perceived state of the environment. As an example, consider the task of a robot fetching a magazine from a desk. The method to perform this task varies depending on several properties of the environment: for example, the robot's relative distance from the magazine, the robot's relative orientation, the thickness of the magazine, and the presence or the absence of other items on top of the magazine. If the magazine is very thin, the robot may have to slide the magazine to the side of the table to pick it up. If there is a mug sitting on top of the magazine, it would have to be moved prior to the magazine being picked up. Thus, especially when the details of the manipulation task are under-specified, the success of executing the task depends on the ability to detect the object and on the ability to sequence the set of *primitives* (navigation and manipulation controllers) in various ways in response to the environment.

In recent years, there have been significant developments in building low-level controllers for robots [34] as well as in perceptual tasks such as object detection from sensor data [20, 11, 35]. In this work, our goal is to, given the environment and the task, enable robots to sequence the navigation



Task: pour (obj17)
Inferred Sequence:
```
move_close(obj13) → grasp(obj13) → move_close(obj05)
→ place_above(obj13,obj05) → release(obj13) → move_close(obj19)
→ grasp(obj19) → move_close(obj13) → hold_above(obj19,obj13)
→ follow_traj_pour(obj19,obj13)
```

Fig. 1. Figure showing our Kodiak PR2 in a kitchen with different objects labeled with attributes. To accomplish the under-defined task of `pour(obj17)`, it has to first find the mug (obj13) and carry it to the table (obj05) since it is dangerous to pour liquid in a tight shelf. Once the mug is on the table, it has to bring the liquid by the container (obj19) and then finally pour it into the mug.

and manipulation primitives. Manually sequencing instructions is not scalable because of the large variety of tasks and situations that can arise in unstructured environments.

In this work, we take an attribute-based representation of the environment, where each object is represented with a set of attributes, such as their size, shape-related information, presence of handles, and so forth. For a given task, there are often multiple objects with similar functions that can be used to accomplish the task, and humans can naturally reason and choose the most suitable object for the given task [17]. Our model, based on attribute representation of objects, is similarly capable of choosing the most suitable object for the given task among many objects in the environment.

We take a dynamic planning approach to the problem of synthesizing, in the right order, the suitable primitive controllers. The best primitive to execute at each discrete time step is based on a score function that represents the appropriateness of a particular primitive for the current state of the environment. Conceptually, a dynamic plan consists of a loop containing a sequence of conditional statements each with an associated primitive controller or action. If the current environment matches the conditions of one of

Jaeyong Sung, Bart Selman and Ashutosh Saxena are with the Department of Computer Science, Cornell University, Ithaca, NY. Email: {jysung,selman,asaxena}@cs.cornell.edu

[1]A preliminary version of this work was presented at ICML workshop on Prediction with Sequential Models, 2013 [33].

the conditional statements, the corresponding primitive controller is executed, bringing the robot one step closer to completing the overall task (example in Section III). We will show how to generalize sequencing of primitives to make them more flexible and robust, by switching to an attribute-based representation. We then show how to unroll the loop into a graph-based representation, isomorphic to a Markov Random Field. We then train the parameters of the model by maximum margin learning method using a dataset comprising many examples of sequences.

We evaluated our model on 127 controller sequences for five under-specified manipulation tasks generated from 13 environments using 7 primitives. We show that our model can predict suitable primitives to be executed with the correct arguments in most settings. Furthermore, we show that, for five high-level tasks, our algorithm was able to correctly sequence 70% of the sequences in different environments.

The main contributions of this paper are:

- using an attribute-based representation of the environment for task planning,
- inferring the sequence of steps where the goals are under-specified and have to be inferred from the context,
- a graph-based representation of a dynamic plan by unrolling the loop into a Markov Random Field.

II. Related Work

There is a large body of work in task planning across various communities. We describe some of them in the following categories.

Manual Controller Sequencing. Many works manually sequence different types of controllers to accomplish specific types of tasks. Bollini et al. [4] develop an end-to-end system which can find ingredients on a tabletop and mix them uniformly to bake cookies. Others used pre-programmed sequences for tea serving and carrying humans in healthcare robotics [25, 24]. These approaches however cannot scale to large number of tasks when each task requires its own complicated rules for sequencing controllers and assumes a controlled environment, which is very different from actual human households, where objects of interest can appear anywhere in the environment with a variety of similar objects.

Beetz et al. [2] retrieve a sequence for “making a pancake” from online websites but assumes an environment with correct labels and a single choice of object for the task. Human experts can generate finite state machines for robots but this again requires explicit labels (e.g. AR tags) [27]. Our work addresses these problems by representing each object in the environment as a set of attributes which is more robust than labeling the individual object [7, 6, 22]. In our recent work [23], we learn a sequence given a natural language instruction and object labels, where the focus is to learn the grounding of the natural language into the environment.

Learning Activities from Videos. In the area of computer vision, several works [37, 38, 32, 19] consider modeling the sequence of activities that humans perform. These works are complementary to ours because our problem is to infer the sequence of controllers and not to label the videos.

Symbolic Planning. Planning problems often rely on symbolic representation of entities as well as their relations. This has often been formalized as a deduction [9] or satisfiability problem [16]. A plan can also be generated hierarchically by first planning abstractly, and then generating a detailed plan recursively [15]. Such approaches can generate a sequence of controllers that can be proven to be correct [14, 3]. Symbolic planners however require encoding every precondition and effect of each operation, which will not scale in human environments where there are large variations. Such planners also require domain description for each planning domain including the types of each object (e.g., pallet crate - surface, hoist surface - locatable) as well as any relations (e.g., on x:crate y:surface, available x:hoist). The preconditions and effects can be learned directly from examples of recorded plans [36, 39] but this method suffers when there is noise in the data [39], and also suffers from the difficulty of modeling real world situations with the PDDL representation [36].

Such STRIPS-style representation also restricts the environment to be represented with explicit labels. Though there is a substantial body of work on labeling human environments [20, 21], it still remains a challenging task. A more reliable way of representing an environment is representing through attributes [7, 6]. An attribute-based representation even allows classification of object classes that are not present in the training data [22]. Similarly, in our work, we represent the environment as a set of attributes, allowing the robot to search for objects with the most suitable attributes rather than looking for a specific object label.

Predicting Sequences. Predicting sequences has mostly been studied in a Markov Decision Process framework, which finds an optimal policy given the reward for each state. Because the reward function cannot be easily specified in many applications, inverse reinforcement learning (IRL) learns the reward function from an expert’s policy [26]. IRL is extended to Apprenticeship Learning based on the assumption that the expert tries to optimize an unknown reward function [1]. Most similar to our work, the Max-Margin Planning frames imitation learning as a structured max-margin learning problem [28]. However, this has only been applied to problems such as 2D path planning, grasp prediction and footstep prediction [29], which have much smaller and clearer sets of states and actions compared to our problem of sequencing different controllers. Co-Active Learning for manipulation path planning [10], where user preferences are learned from weak incremental feedback, does not directly apply to sequencing different controllers.

Both the model-based and model-free methods evaluate state-action pairs. When it is not possible to have knowledge about all possible or subsequent states (*full backup*), they can rely on *sample backup* which still requires sufficient sample to be drawn from the state space [8]. However, when lots of robot-object interactions are involved, highly accurate and reliable physics-based robotic simulation is required along with reliable implementation of each manipulation controllers. Note that each of the manipulation primitives such as grasping are still not fully solved problems. For example, consider the scenario where the robot is grasping the edge of the table and was given the instruction of `follow_traj_pour(table,shelf)`. It is unclear what should occur in the environment and becomes challenging to have reliable simulation of actions. Thus, in the context of reinforcement learning, we take a maximum margin based approach to learning the weight for $\mathbf{w}^T\phi(s,a)$ such that it maximizes the number of states where the expert outperforms other policies, and chooses the action that maximizes $\mathbf{w}^T\phi(s,a)$ at each time step. The key in our work is representing task planning as a graph-based model and designing a score function that uses attribute-based representation of environment for under-specified tasks.

III. Our Approach

We refer to a sequence of *primitives* (low-level navigation and manipulation controllers) as a *program*. To model the sequencing of primitives, we first represent each object in the environment with a set of attributes as described in Section IV-B. In order to make programs generalizable, primitives should have the following two properties. First, each primitive should specialize in an atomic operation such as moving close, pulling, grasping, and releasing. Second, a primitive should not be specific to a single high-level task. By limiting the role of each primitive and keeping it general, many different manipulation tasks can be accomplished with the same small set of primitives, and our approach becomes easily adaptable to different robots by providing implementation of primitives on the new robot.

For illustration, we write a program for "throw garbage away" in Program 1. Most tasks could be written in such a format, where there are many `if` statements inside the loop. However, even for a simple "throw garbage away" task, the program is quite complex. Writing down all the rules that can account for the many different scenarios that can arise in a human environment would be quite challenging.

Program 1 "throw garbage away."

```
Input: environment $e$, trash $a_1$
$gc = find\_garbage\_can(e)$
repeat
    if $a_1$ is in hand & $gc$ is close then
        release($a_1$)
    else if $a_1$ is in hand & far from $gc$ then
        move_close($gc$)
    else if $a_1$ is close & $a_1$ not in hand
            & nothing on top of $a_1$ then
        grasp($a_1$)
            ⋮
    else if $a_1$ is far then
        move_close($a_1$)
    end if
until $a_1$ inside $gc$
```

Program 1 is an example of what is commonly referred to as reactive or dynamic planning [31, 18]. In traditional deliberative planning, a planning algorithm synthesizes a sequence of steps that starts from the given state and reaches the given goal state. Although current symbolic planners can find optimal plan sequences consisting of hundreds of steps, such long sequences often break down because of unexpected events during the execution. A dynamic plan provides a much more robust alternative. At each step, the current state of the environment is considered and the next appropriate action is selected by one of the conditional statements in the main loop. A well-constructed dynamic plan will identify the next step required to bring the robot closer to the overall goal in any possible world state. In complex domains, dynamic plans may become too complicated. However, we are considering basic human activities, such as following a recipe, where dynamic plans are generally quite compact and can effectively lead the robot to the goal state. Moreover, as we will demonstrate, we can learn the dynamic plan from observing a series of action sequences in related environments.

In order to make our approach more general, we introduce a feature based representation for the conditions of `if` statements. We can extract some features from both the environment and the action that will be executed in the body of `if` statement. With extracted features ϕ and some weight vector **w** for each `if` statement, the same conditional statements can be written as $\mathbf{w}^T\phi$, since the environment will always contain the rationale for executing certain primitive. Such a feature-based approach allows us to re-write Program 1 in the form of Program 2.

Program 2 "throw garbage away."

```
Input: environment $e$, trash $a_1$
$gc = find\_garbage\_can(e)$
repeat
    $e_t$ = current environment
    if $w_1^T\phi(e_t$,release$(a_1)) > 0$ then
        release($a_1$)
    else if $w_2^T\phi(e_t$,move_close$(gc)) > 0$ then
        move_close($gc$)
            ⋮
    else if $w_n^T\phi(e_t$,move_close$(a_1)) > 0$ then
        move_close($a_1$)
    end if
until $a_1$ inside $gc$
```

Now all the `if` statements have the same form, where the same primitive along with same arguments are used in both the condition as well as the body of the `if` statement. We can therefore reduce all `if` statements inside the loop further down to a simple line which depends only on a single weight vector and a single joint feature map, as shown in Program 3, for finding the most suitable pair of primitive $\hat{p}_t$ and its arguments ($\hat{a}_{1,t}$, $\hat{a}_{2,t}$).

Program 3 "throw garbage away."

```
Input: environment $e$, trash $g_{a1}$
repeat
    $e_t$ = current environment
    $(\hat{p}_t, \hat{a}_{1,t}, \hat{a}_{2,t}) := \arg\max_{p_t \in \mathcal{P}, a_{1,t}, a_{2,t} \in \mathcal{E}} w^T\phi(e_t, p_t(a_{1,t}, a_{2,t}))$
    execute $\hat{p}_t(\hat{a}_{1,t}, \hat{a}_{2,t})$
until $\hat{p}_t = done$
```

The approach taken in Program 3 also allowed removing the function $find\_garbage\_can(e)$. Both Program 1 and Program 2 require $find\_garbage\_can(e)$ which depends on semantic labeling of each object in the environment. The attributes of objects will allow the program to infer which object is a garbage can without explicit encoding.

Program 3 provides a generic representation of a dynamic plan. We will now discuss an approach to learning a set of weights. To do so, we will employ a graph-like representation obtained by "unrolling" the loop representing discrete time steps by different layers. We will obtain a representation that is isomorphic to a Markov Random Field (MRF) and will use a maximum margin based approach to training the weight vector. Our MRF encodes the relations between the environment, primitive and its arguments. Our empirical results show that such a framework is effectively trainable with a relatively small set of example sequences. Our feature-based dynamic plan formulation therefore offers an effective and general representation to learn and generalize from action sequences, accomplishing high-level tasks in a dynamic environment.

IV. Model Formulation

We are given a set of possible primitives $\mathcal{P}$ (navigation and manipulation controllers) to work with (see Section V) and an environment $\mathcal{E}$ represented by a set of attributes. Using these primitives, the robot has to accomplish a manipulation task $g \in \mathcal{T}$. The manipulation task g is followed by the arguments $g_{a1}, g_{a2} \in \mathcal{E}$ which give a specification of the task. For example, the program "throw garbage away" would have a single argument which would be the object id of the object that needs to be thrown away.

At each time step t (i.e., at each iteration of the loop in Program 3), our environment e_t will dynamically change, and its relations with the primitive is represented with a joint set of features. These features include information about the physical and semantic properties of the objects as well as information about their locations in the environment.

Now our goal is to predict the best primitive $p_t \in \mathcal{P}$ to execute at each discrete time step, along with its arguments: $p_t(a_{1,t}, a_{2,t})$. We will do so by designing a score function $S(\cdot)$ that represents the correctness of executing a primitive in the current environment for a task.

$$\begin{aligned} S(g(g_{a1}, g_{a2}), e_t, & p_t(a_{1,t}, a_{2,t})) = \\ & w^T \phi(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t})) \end{aligned}$$

In order to have a parsimonious representation, we decompose our score function using a model isomorphic to a Markov Random Field (MRF), shown in Figure 2. This allows us to capture the dependency between primitives, their arguments, and environments which are represented by set of attributes. In the figure, the top node represents the given task and its arguments (g, g_{a1}, g_{a2}). The second layer from the top represents the sequence of primitives, and the layer below represents the arguments associated with each primitive. And, the bottom node represents the environment which is represented with set of attributes. Note that we also take into account the previous two primitives in the past, together with their arguments: $p_{t-1}(a_{1,t-1}, a_{2,t-1})$ and $p_{t-2}(a_{1,t-2}, a_{2,t-2})$.

Now the decomposed score function is:

$$S = \underbrace{S_{ae}}_{\text{args-env}} + \overbrace{S_{pt}}^{\text{prim-task}} + \underbrace{S_{aet}}_{\text{args-env-task}} + \overbrace{S_{pae}}^{\text{prim-args-env}} + \underbrace{S_{ppt}}_{\text{prim-prim(prev)-task}} + \overbrace{S_{paae}}^{\text{prim-args-args(prev)-env}}$$

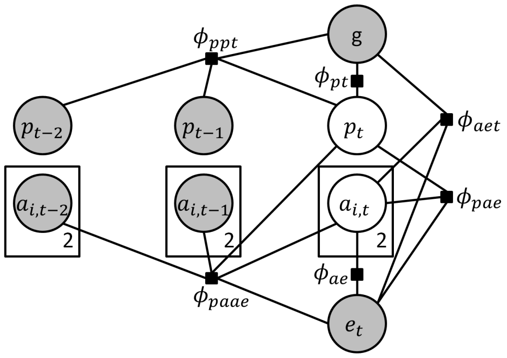


Fig. 2. **Markov Random Field representation of our model** at discrete time step t. The top node represents the given task g, g_{a1}, g_{a2}. The second layer from the top represents the sequence of primitives, and the layer below represents the arguments associated with each primitive. And, the bottom node represents the environment represented with set of attributes.

The terms associated with an edge in the graph are defined as a linear function of its respective features ϕ and weights w:

$$\begin{aligned} S_{ae} &= {w_{ae1}}^T \phi_{ae}(a_{1,t}, e_t) + {w_{ae2}}^T \phi_{ae}(a_{2,t}, e_t) \\ S_{pt} &= {w_{pt}}^T \phi_{pt}(p_t, g) \end{aligned}$$

Similarly, the terms associated with a clique in the graph are defined as a linear function of respective features ϕ and weights w:

$$\begin{aligned} S_{aet} &= {w_{aet1}}^T \phi_{aet}(a_{1,t}, e_t, g) + {w_{aet2}}^T \phi_{aet}(a_{2,t}, e_t, g) \\ S_{pae} &= {w_{pae1}}^T \phi_{pae}(p_t, a_{1,t}, e_t) + {w_{pae2}}^T \phi_{pae}(p_t, a_{2,t}, e_t) \\ S_{ppt} &= {w_{ppt1}}^T \phi_{ppt}(p_{t-1}, p_t, g) + {w_{ppt2}}^T \phi_{ptt}(p_{t-2}, p_t, t) \\ S_{paae} &= \sum_{i,j \in (1,2), k \in (t-2,t-1)} {w_{paae_{ijk}}}^T \phi_{paae}(p_t, a_{i,k}, a_{j,t}, e_t) \end{aligned}$$

Using these edge and clique terms, our score function S can be simply written in the following form, which we have seen in Program 3 with an extra term g for the task: $S(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t})) = w^T \phi(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t}))$.

A. Features

In this section, we describe our features $\phi(\cdot)$ for the different terms in the previous section.

Arguments-environment (ϕ_{ae}): The robot should be aware of its location and the current level of its interaction with objects (e.g., grasped), which are given as possible primitive arguments $a_{1,t}, a_{2,t}$. Therefore, we add two binary features which indicate whether each primitive argument is already grasped and two features for the centroid distance from the robot to each primitive arguments.

For capturing spatial relation between two objects $a_{1,t}$ and $a_{2,t}$, we add one binary feature indicating whether primitive arguments $a_{1,t}, a_{2,t}$ are currently in collision with each other.

Arguments-environment-task (ϕ_{aet}): To capture relations between the objects of interest (task arguments) and objects of possible interest (primitive arguments), we build a binary vector of length 8. First four represents the indicator values of whether the objects of interest are identical as the objects of possible interest, and the last four represents spatial relation of whether they overlap from top view.

It is important to realize the type of object that is below the objects of interests, and the desired property (e.g., bowl-like object or table-like object) may differ depending on the situation. We create two feature vectors, each of length l. If the robot is holding the object, we store its extracted attributes in the first vector. Otherwise, we store them in the second vector. If the primitive has two arguments, we use the first primitive argument since it often has higher level of interaction with the robot compared to the second argument.

Finally, to capture correlation between the high-level task and the types of object in primitive argument, we take a tensor product of two vectors: an attribute vector of length $2l$ for two objects and a binary occurrence vector of length $|\mathcal{T}|$. The matrix of size $2l \times |\mathcal{T}|$ is flattened to a vector.

Primitive-task (ϕ_{pt}): The set of primitives that are useful may differ depending on the type of the task. We create a $|\mathcal{T}| \times |\mathcal{P}|$ binary co-occurrence matrix between the task g and the primitive p_t that has a single non-zero entry in the current task's (g^{th}) row and current primitive's (${p_t}^{th}$) column.

Primitive-arguments-environment (ϕ_{pae}): Some primitives such as `hold_above` require one of the objects in arguments to be grasped or not to be grasped to execute correctly. We create a $|\mathcal{P}| \times 2$ matrix where the row for the current primitive (${p_t}^{th}$ row) contains two binary values indicating whether each primitive argument is in the manipulator.

Primitive-primitive(previous)-task (ϕ_{ppt}): The robot makes different transitions between primitives for different tasks. Thus, a binary co-occurrence matrix of size $|\mathcal{T}| \times |\mathcal{P}|^2$ represents transition occurrence between the primitives for each task. In this matrix, we encode two transitions for the current task g, from $t-2$ to t and from $t-1$ to t.

Primitive-arguments-arguments(previous)-environment (ϕ_{paae}): For a certain primitive in certain situations, the arguments may not change between time steps. For example, `pour(A,B)` would often be preceded by `hold_above (A,B)`. Thus, the matrix of size $|\mathcal{P}| \times 8$ is created, with the ${p_t}^{th}$ row containing 8 binary values representing whether the two primitive arguments at time t are the same as the two arguments at $t-1$ or the two arguments at $t-2$.

B. Attributes.

Every object in the environment including tables and the floor is represented using the following set of attributes: height h, max(width(w),length(l)), min(w, l), volume($w * l * h$), min(w, l, h)-over-max(w, l, h), median(w, l, h)-over-max(w, l, h), cylinder-shape, box-shape, liquid, container, handle, movable, large-horizontal-surface, and multiple-large-horizontal-surface. Attributes such as cylinder-shape, box-shape, container, handle, and large-horizontal-surface can be reliably extracted from RGB or RGBD images, and were shown to be useful in several different applications [7, 6, 22, 20]. We study the effects of attribute detection errors on our model in Section V.

C. Learning

We use a max-margin approach to train a single model for all tasks. This maximum margin approach fits our formulation, since it assumes that the discriminant function is a linear function of a weight vector $\mathbf{w}$ and a joint feature map $\phi(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t}))$, and it has time complexity linear with the number of training examples when solved using the cutting plane method [13]. We formalize our problem as a "1-slack" structural SVM optimization problem:

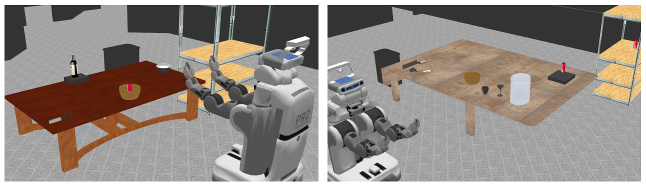


Fig. 3. Figure showing two of our 13 environments in our evaluation dataset using 43 objects along with PR2 robot.

$$\min_{\mathbf{w},\xi \geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{l}\sum_{i=1}^{n}\sum_{t=1}^{l^i}\xi_t^i$$

$$s.t. \quad \text{for } 1 \leq i \leq n, \text{for each time step } t: \quad \forall \hat{p} \in \mathcal{P}, \forall \hat{a}_1, \hat{a}_2 \in \mathcal{E}:$$
$$\boldsymbol{w}^T[\phi(g^i(g^i_{a1}, g^i_{a2}), e^i_t, p^i_t(a^i_{1,t}, a^i_{2,t})) - \phi(g^i(g^i_{a1}, g^i_{a2}), e^i_t, \hat{p}(\hat{a}_1, \hat{a}_2))]$$
$$\geq \Delta(\{p^i_t, a^i_{1,t}, a^i_{2,t}\}, \{\hat{p}, \hat{a}_1, \hat{a}_2\}) - \xi^i_t$$

where n is the number of example sequences, l^i is the length of the i^{th} sequence, and l is the total length combining all sequences. The loss function is defined as:
$\Delta(\{p, a_1, a_2\}, \{\hat{p}, \hat{a}_1, \hat{a}_2\}) = \mathbb{1}(p \neq \hat{p}) + \mathbb{1}(a_1 \neq \hat{a}_1) + \mathbb{1}(a_2 \neq \hat{a}_2)$

With a learned $\mathbf{w}$, we choose the next action in sequence by selecting a pair of primitive and arguments that gives the largest discriminant value:

$$\operatorname*{arg\,max}_{p_t \in \mathcal{P}, a_{1,t}, a_{2,t} \in \mathcal{E}} w^T\phi(g(g_{a1}, g_{a2}), e_t, p_t(a_{1,t}, a_{2,t}))$$

V. Experiments

Dataset. We considered seven primitives (low-level controllers): `move_close (A)`, `grasp (A)`, `release (A)`, `place_above (A,B)`, `hold_above (A,B)`, `follow_traj_circle (A)` and `follow_traj_pour (A,B)`. Depending on the environment and the task, these primitives could be instantiated with different arguments. For example, consider an environment that contains a bottle (obj04) containing liquid (obj16) and an empty cup (obj02) placed on top of the shelf, among other objects. If, say from a recipe, our task is to pour the liquid, then our program should figure out the correct sequence of primitives with correct arguments (based on the objects' attributes, etc.):

$\{$`pour(obj16)`$; env2\} \rightarrow$
$\{$`move_close(obj02); grasp(obj02); move_close(obj04); place_above(obj02,obj26); release(obj02); grasp(obj04); hold_above(obj04,obj02); follow_traj_pour(obj04,obj02)`$\}$

Note that the actual sequence does not directly interact with the liquid (obj16)—the only object specified by the task—but rather with a container of liquid (obj04), an empty cup (obj02), and a table (obj26), while none of these objects are specified in the task arguments. As seen in this example, the input for our planning problem is under-specified.

For evaluation, we prepared a dataset where the goal was to produce correct sequences for the following tasks in different environments:

- `stir(A)`: Given a liquid A, the robot has to identify a stirrer of ideal size (from several) and stir with it. The

TABLE I

RESULT OF BASELINES, OUR MODEL WITH VARIATIONS OF FEATURE SETS, AND OUR FULL MODEL ON OUR DATASET CONSISTING OF 127 SEQUENCES. THE "PRIM" COLUMNS REPRESENT PERCENTAGE OF PRIMITIVES CORRECTLY CHOSEN REGARDLESS OF ARGUMENTS, AND "ARGS" COLUMNS REPRESENT PERCENTAGE OF A CORRECT PAIR OF PRIMITIVE AND ARGUMENTS. THE LAST COLUMN SHOWS AVERAGE PERCENTAGE OF SEQUENCES CORRECT OVER THE FIVE PROGRAMS EVALUATED.

	move_close		grasp		release		place_above		hold_above		traj_circle		traj_pour		**Average**		**Sequence**	
	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg
chance	14.3	1.1	14.3	1.1	14.3	1.1	14.3	0.1	14.3	0.1	14.3	1.1	14.3	0.1	14.3	0.7	0	0
multiclass	99.6	-	90.4	-	95.7	-	68.5	-	79.7	-	100.0	-	14.7	-	78.4	-	-	-
symb-plan-svm	99.6	82.5	94.2	72.4	67.4	63.0	60.9	43.5	76.6	73.4	96.7	76.7	97.1	91.2	84.6	71.8	58.4	49.6
symb-plan-manual	99.6	85.4	94.2	76.3	67.4	63.0	60.9	50.0	76.6	76.6	96.7	96.7	97.1	97.1	84.6	77.9	58.4	54.9
Only edge features	23.5	15.3	56.4	45.5	93.5	93.5	0.0	0.0	18.8	9.4	100.0	100.0	50.0	44.1	48.9	44.0	0	0
Only clique features	99.6	1.9	96.8	82.7	90.2	90.2	72.8	15.2	87.5	15.6	96.7	96.7	100.0	97.1	91.9	57.0	45.0	0
Ours - full	99.3	82.8	96.8	84.0	97.8	97.8	89.1	79.3	96.9	92.2	100.0	100.0	97.1	94.1	96.7	**90.0**	91.6	**69.7**

liquid may be located on a tight shelf where it would be dangerous to stir the liquid, and the robot should always stir it on top of an open surface, like a table. The robot should always only interact with the container of the liquid, rather than the liquid itself, whenever liquid needs to be carried or poured. Our learning algorithm should learn such properties.

- `pick_and_place(A,B)`: The robot has to place A on top of B. If A is under some other object C, the object C must first be moved before interacting with object A.
- `pour(A)`: The robot has to identify a bowl-like object without object labels and pour liquid A into it. Note again that liquid A cannot be directly interacted with, and it should not be poured on top of a shelf.
- `pour_to(A,B)`: The liquid A has to be poured into the container B. (A variant of the previous task where the container B is specified but the model should be able to distinguish two different tasks.)
- `throw_away(A)`: The robot has to locate a garbage can in the environment and throw out object A.

In order to learn these programs, we collected 127 sequences for 113 unique scenarios by presenting participants the environment in simulation and the task to be done. We considered a single-armed mobile manipulator robot for these tasks. In order to extract information about the environment at each time frame of every sequence, we implemented each primitive using OpenRAVE simulator [5]. Though most of the scenarios had a single optimal sequence, multiple sequences were introduced when there were other acceptable variations. The length of each sequence varies from 4 steps to 10 steps, providing a total of 736 instances of primitives. To ensure variety in sequences, sequences were generated based on the 13 different environments shown in Figure 3, using 43 objects each with unique attributes.

Baseline Algorithms. We compared our model against following baseline algorithms:

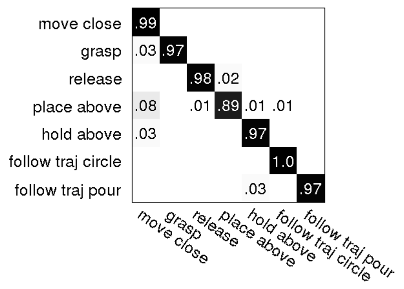
- *chance*: At each time step, a primitive and its arguments are selected at random.
- *multiclass*: A multiclass SVM [13] was trained to predict primitives without arguments, since the set of possible arguments changes depending on the environment.
- *symbolic-plan-svm*: A PDDL-based symbolic planner [36, 39] requires a domain and a problem definition. Each scenario was translated to symbolic entities and relations. However, the pre-conditions and effects of each action in domain definition were hand-coded, and each object was labeled with attributes using predicates. Unlike our model that works on an under-specified problem, each symbolic planning problem requires an explicit goal state. In order to define these goal states, we have trained ranking SVMs [12] in order to detect a 'stirrer', an 'object to pour into' and a 'garbage can' for `stir`, `pour`, and `throw_away`, respectively. Each symbolic planning instance was then solved by reducing to a satisfiability problem [16, 30].
- *symbolic-plan-manual*: Based on the same method as *symbolic-plan-svm*, instead of training ranking SVMs, we provided ground-truth goal states. Even after providing lots of hand-coded rules, it is still missing some rules due to the difficulty of representation using PDDL [36, 39], These missing rules include the fact that liquid needs to be handled through its container and that objects should not be manipulated on top of the shelf.

Evaluation and Results. We evaluated our algorithm through *6-fold cross-validation*, computing accuracies over primitives, over primitives with arguments, and over the full sequences. Figure 4(a) shows the confusion matrix for prediction of our seven primitives. We see that our model is quite robust for most primitives.

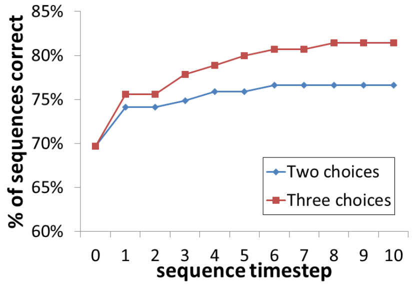
With our dataset, our model was able to correctly predict pairs of primitives and arguments 90.0% of the time and full sequences 69.7% of the time (Table I). Considering only the primitives without arguments, it was able to predict primitive 96.7% of the time and full sequence 91.6% of the time. The last column of Table I shows the performance with respect to whether the complete sequence was correct or not. For example, for "pouring", our model has learned not only to bring a cup over to the table, but also to pick out the cup when there are multiple other objects like a pot, a bowl, or a can that may have similar properties.

How do baselines perform for our under-specified planning problem? The results of various baseline algorithms are shown in Table I. If the primitive and arguments pairs are predicted at random, none of the sequences would be correct because of the large search space of arguments. *Multiclass* predicted well for some of the primitives but suffered greatly on primitives like `place_above`, `hold_above` and `follow_traj_pour`, which drastically impacts constructing overall sequences, even with correct arguments selected.

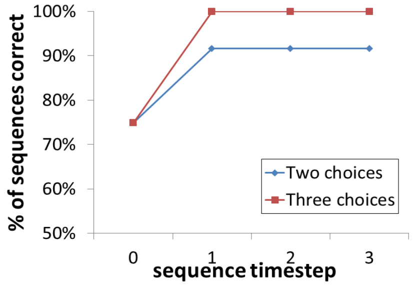
The symbolic planner based approaches, *symbolic-plan-*



(a) **Confusion matrix** for the seven primitives in our dataset. Our dataset consist of 736 instances of seven primitives in 127 sequences on five manipulation tasks.



(b) **Percentage of programs correct.** Without any feedback in completely autonomous mode, the accuracy is 69.7%. With feedback (number of feedbacks on x-axis), the performance increases. This is on full 127 sequence dataset.



(c) **Percentage of programs correct for 12 high-level tasks** such as making sweet tea. In completely autonomous mode, the accuracy is 75%. With feedback (number of feedbacks on x-axis), the performance increases.

Fig. 4. **Results with cross-validation.** (a) On predicting the correct primitive individually. (b) On predicting programs, with and without user intervention. (c) On performing different tasks with the predicted sequences.

svm and *symbolic-plan-manual*, suffered greatly from under-specified nature of the problem. The planners predicted correctly 49.6% and 54.9% of the times, respectively, compared to our model's performance of 69.7%. Even though both planners made use of heavily hand-coded domain definitions of the problem, due to the nature of the language used by symbolic planners, rules such as that liquid should not be handled on top of shelves were not able to be encoded. Even if the language were capable of encoding these rules, it would require a human expert in planning language to carefully encode every single rule the expert can come up with.

Also, by varying the set of features, it is evident that without very robust primitive-level accuracies, the models are unable to construct a single correct sequence.

How important is attribute representation of objects? For 113 unique scenarios in our dataset, we have randomly flipped binary attributes and observed the effects of detection errors on correctness for the full sequence (Figure 6). When there is no error in detecting attributes, our model performs at 69.7%. With 10% detection error, it performs at 55.8%, and with 40% detection errors, it performs at 38.1%. Since the attribute detection is more reliable than the object detection [7, 6, 22], our model will perform better than planners based on explicit object labels.

How can the robot utilize learned programs? These learned programs can form higher level tasks such as making a recipe found online. For example, serving sweet tea would require the following steps: pouring tea into a cup, pouring sugar into a cup, and stirring it (Figure 5). We have tested each of the four tasks, *serve-sweet-tea*, *serve-coffee-with-milk*, *empty-container-and-throw-away*, and *serve-and-store*, in three environments. Each of the four tasks can be sequenced in following manner by programs respectively: `pour` → `pour_to` → `stir`, `pour_to` → `pour_to`, `pour` → `throw_away`, and `pour` → `pick_and_place`. Out of total 12 scenarios, our model was able to successfully complete the task for 9 scenarios.

Does the robot need a human observer? In an assistive robotics setting, a robot will be accompanied by a human observer. With help from the human, performance can be greatly improved. Instead of choosing a primitive and argument pair that maximizes the discriminant function, the robot

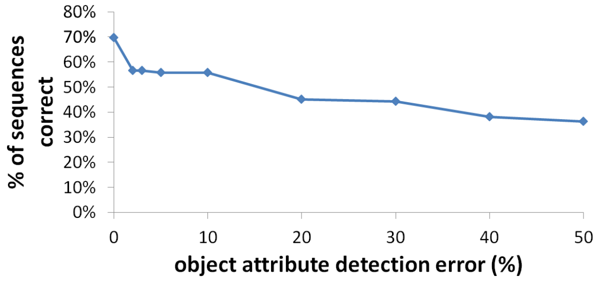


Fig. 6. **Effect of attribute perception error.** Figure showing percentage of programs correct with attribute labeling errors for binary attributes. For 113 unique scenarios, binary attributes were randomly flipped.

can present the top 2 or 3 primitive and argument pairs to the observer, who can simply give feedback on the best option among those choices. At the initial time step of the sequence, with only a single piece of feedback, given 2 or 3 choices, performance improves to 74.1% and 75.6% respectively from 69.7% (Figure 4(b)). If feedback was provided through whole sequence with the top 2 or 3 choices, it further improves to 76.7% and 81.4%. Furthermore, the four higher level tasks (recipes) considered earlier also shows that with a single feedback at the initial time step of each program, the results improve from 75% to 100% (Figure 4(c)).

Robotic Experiments. Finally, we demonstrate that our inferred programs can be successfully executed on our Kodiak PR2 robot for a given task in an environment. Using our implementation of the primitives discussed in Section V, we show our robot performing the task of "serving sweet tea." It comprises executing three programs in series – `pour`, `pour_to` and `stir` – which in total required sequence of 20 primitives with correct arguments. Each of these programs (i.e., the sequence of primitives and arguments) is inferred for this environment. Figure 5 shows a few snapshots and the full video is available at: `http://pr.cs.cornell.edu/learningtasksequences`

VI. Conclusion

In this paper, we considered the problem of learning sequences of controllers for robots in unstructured human environments. In an unstructured environment, even a simple task such as pouring can take variety of different sequences of controllers depending on the configuration of the environment. We took a dynamic planning approach, where we

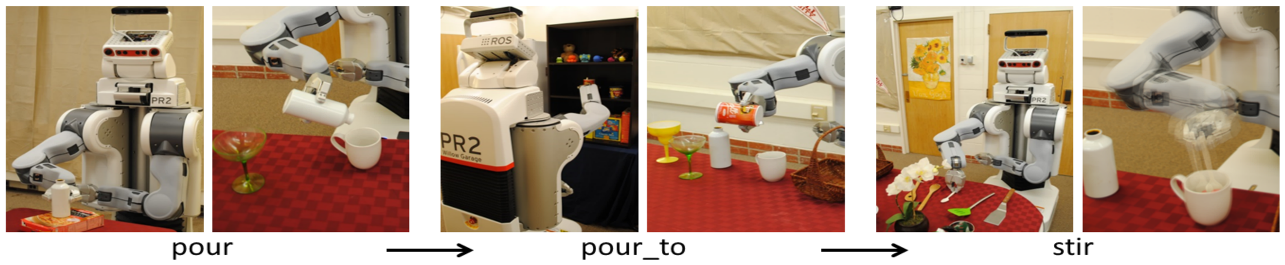


Fig. 5. **Few snapshots of learned sequences** forming the higher level task of serving sweet tea, which takes the sequence of pouring tea into a cup, pouring sugar into a cup, and then stirring it.

represent the current state of the environment using a set of attributes. To ensure that our dynamic plans are as general and flexible as possible, we designed a score function that captures relations between task, environment, primitives, and their arguments, and we trained a set of parameters weighting the various attributes from example sequences. By unrolling the program, we can obtain a Markov Random Field style representation, and use a maximum margin learning strategy. We demonstrated on a series of example sequences that our approach can effectively learn dynamic plans for various complex high-level tasks.

Acknowledgements

This work was supported in part by ONR Grant N00014-14-1-0156, and Microsoft Faculty Fellowship and NSF Career award to Saxena.

References

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[2] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr, and M. Tenorth. Robotic roommates making pancakes. In *Humanoids*, 2011.

[3] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion. *Robotics & Automation Magazine*, 2007.

[4] M. Bollini, J. Barry, and D. Rus. Bakebot: Baking cookies with the pr2. In *The PR2 Workshop, IROS*, 2011.

[5] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, 2010.

[6] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. Describing objects by their attributes. In *CVPR*, 2009.

[7] V. Ferrari and A. Zisserman. Learning visual attributes. In *NIPS*, 2007.

[8] L. Frommberger. *Qualitative Spatial Abstraction in Reinforcement Learning*. Springer, 2010.

[9] C. Green. Application of theorem proving to problem solving. Technical report, DTIC Document, 1969.

[10] A. Jain, B. Wojcik, T. Joachims, and A. Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *NIPS*, 2013.

[11] Y. Jiang, H. Koppula, and A. Saxena. Hallucinated humans as the hidden context for labeling 3d scenes. In *CVPR*, 2013.

[12] T. Joachims. Training linear svms in linear time. In *KDD*, 2006.

[13] T. Joachims, T. Finley, and C.-N. J. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[14] B. Johnson and H. Kress-Gazit. Probabilistic analysis of correctness of high-level robot behavior with sensor error. In *RSS*, 2011.

[15] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *ICRA*, 2011.

[16] H. Kautz and B. Selman. Planning as satisfiability. In *European conference on Artificial intelligence*, 1992.

[17] C. Kemp, N. D. Goodman, and J. B. Tenenbaum. Learning to learn causal models. *Cognitive Science*, 34(7), 2010.

[18] S. Koenig. Agent-centered search. *AI Magazine*, 2001.

[19] H. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *RSS*, 2013.

[20] H. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3d point clouds for indoor scenes. *NIPS*, 2011.

[21] K. Lai, L. Bo, X. Ren, and D. Fox. Detection-based object labeling in 3d scenes. In *ICRA*, 2012.

[22] C. H. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *CVPR*, 2009.

[23] D. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *RSS*, 2014.

[24] T. Mukai, S. Hirano, H. Nakashima, Y. Kato, Y. Sakaida, S. Guo, and S. Hosoe. Development of a nursing-care assistant robot riba that can lift a human in its arms. In *IROS*, 2010.

[25] H. Nakai, M. Yamataka, T. Kuga, S. Kuge, H. Tadano, H. Nakanishi, M. Furukawa, and H. Ohtsuka. Development of dual-arm robot with multi-fingered hands. In *RO-MAN*, 2006.

[26] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

[27] H. Nguyen, M. Ciocarlie, J. Hsiao, and C. Kemp. Ros commander (rosco): Behavior creation for home robots. In *ICRA*, 2013.

[28] N. Ratliff, J. A. Bagnell, and M. Zinkevich. Maximum margin planning. In *ICML*, 2006.

[29] N. D. Ratliff, D. Silver, and J. A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 2009.

[30] J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 2012.

[31] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach.* Prentice Hall, 2010.

[32] J. Sung, C. Ponce, B. Selman, and A. Saxena. Unstructured human activity detection from rgbd images. In *ICRA*, 2012.

[33] J. Sung, B. Selman, and A. Saxena. Learning sequences of controllers for complex manipulation tasks. In *ICML workshop on Prediction with Sequential Models*, 2013.

[34] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*. MIT press Cambridge, 2005.

[35] C. Wu, I. Lenz, and A. Saxena. Hierarchical semantic labeling for task-relevant rgb-d perception. In *RSS*, 2014.

[36] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171, 2007.

[37] W. Yang, Y. Wang, and G. Mori. Recognizing human actions from still images with latent poses. In *CVPR*, 2010.

[38] B. Yao and L. Fei-Fei. Modeling mutual context of object and human pose in human-object interaction activities. In *CVPR*, 2010.

[39] H. H. Zhuo, Q. Yang, D. H. Hu, and L. Li. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 2010.