Learning Sequences of Controllers for Complex Manipulation Tasks

Jaeyong Sung JYSUNG@CS.CORNELL.EDU
Bart Selman SELMAN@CS.CORNELL.EDU
Ashutosh Saxena ASAXENA@CS.CORNELL.EDU
Department of Computer Science, Cornell University, Ithaca, NY 14850 USA

Abstract

Many tasks in human environments require performing a sequence of complex navigation and manipulation tasks. In unstructured human environments, the locations and configuration of the objects can 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 activities. High level activities are expressed as a sequence of *primitive* actions or controllers (with appropriate parameters). Our planning model synthesizes a universal strategy, where the a suitable next action is selected based on the current state of the environment. By expressing the environment using sets of attributes, the approach generalizes well to unseen scenarios. By unfolding our planning strategy into a Markov Random Field approximation, we can effectively train parameters using a maximum margin learning strategy. We provide a detailed empirical validation of our overall framework demonstrating successful plan strategies for a variety of tasks.

1. Introduction

In the area of robotics, carrying out complex manipulation tasks requires piecing together a sequence of controllers that operate in response to the perceived state of the environment. In this work, we call these sequences of controllers (or primitives), a *program*. As an example, consider the task of a robot fetching a magazine from a desk. The method to perform these tasks would vary depending on several properties: for example, the robot's relative distance from the magazine, the robot's relative orientation, thickness of the magazine, and presence or 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 being picked up. Thus, there is a large variety in the way we write a program for this task.

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

Manually writing such programs is not scalable because of the large variety of tasks and situations that can arise. In recent years, there have been significant developments in building low-level controllers for robots (Thrun et al., 2005) as well as in perceptual tasks such as object detection from sensor data (Koppula et al., 2011). In this work, our goal is to enable robots to develop programs (sequences of controllers operating in response to the perceptual input) for new tasks and situations. Learning from previously seen examples of various activities will allow robot to infer correct programs to achieve complex manipulation tasks even if the robot has not seen a particular environment in the context of this particular task. These programs can in turn be sequenced together with just high-level instructions.

We start by representing the objects in the environment with a set of attributes, such their size, shape-related information, presence of handles, and so forth. In other approaches, it is generally assumed that we have an environment that contains only a single possible choice of object or specific object for the robot to interact with. However, for a given task, there are often multiple objects that can be used. For humans, it is natural to reason and choose the most suitable object for the given task (Kemp et al., 2010). Using a attribute representation of a objects in the environment, our model, after a training phase, 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 current environment is represented as a set of attributes. 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 the conditional statements, the corresponding primitive controller is executed bringing the robot one step closer to completing the overall task. We refer to such a dynamic plan as a "Program." For an example program, see Section 3. We will show how to generalize such programs to make them more flexible and robust, by switching to an attribute based representation. We then show how to unwind the loop into a graph-based representation, isomorphic to a Markov Random Field. We can then train the parameters of the resulting model on a series of example controller sequences for high-level task. During the training stage, we are given various environments and a set of available controllers for the robot along with example sequences of different manipulation tasks, and we use a maximum-margin learning approach to train the parameters for sequencing controllers which will enable the robot to carry out a variety of manipulation task autonomously in previously unseen environments.

We have evaluated our model on 127 controller sequences (programs) for five 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 sequence together correct programs in different environments.

2. Related Work

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

Manual Controller Sequencing. Many work manually sequences different types of controllers for specific applications such as baking cookies, making a pancake or laundry folding (Bollini et al., 2011; Beetz et al., 2011; Miller et al., 2012). Most of these work have assumed a controlled environment, which is very different from actual human households where objects of interest can appear on table, on shelf or in refrigerator and the environment may even have a variety of similar objects. More importantly, it cannot scale to large number of tasks when each requires its own complicated rules of sequencing controllers in order to carry out tasks in human environment. Beetz et al. (2011) retrieves sequence of "making pancake" from online websites such as wikihow.com but requires good labelling of the environment but was only tested in an environment with single choices. Nguyen et al. (2013) relies on human experts to generate hierarchical finite state machine but requires user to put AR tags on every object in order to re-use the sequence in new environments.

Symbolic Planning. Planning has long been formalized as a deduction (Green, 1969) or satisfiability problem (Kautz & Selman, 1992). Kaelbling & Lozano-Pérez (2011) generate a plan hierarchically by first planning abstractly and then generating a detailed plan recursively. Such approaches can generate sequence of controllers that can be proven to be correct (Johnson & Kress-Gazit, 2011). Rather than defining precondition for each operation by hand, Yang et al. (2007) learns PDDL representation (variant of STRIPS style) of action and builds weighted proportional satisfiability problem (MAX-SAT) from examples of recorded plans. Though it does not require encoding all intermediate state information, it still requires domain description for each planning domain (e.g., load depot) such as types of each object (e.g., pallet crate x, truck hoist surface y) as well as any relations (e.g., crate x on surface y, hoist z available). Symbolic planners require encoding every precondition for each operation, or encoding domain knowledge which may work well in restricted environments, but will not be able to scale in human environments where there is a large variation in the environments where the task is to be performed.

Such STRIPS-style representation also restricts the environment to be represented as explicit labels. Though there is a substantial body of work on labeling human environments (Koppula et al., 2011), it still remains to be a challenging task. A more reliable way of representing an environment is a representation through attributes (Ferrari & Zisserman, 2007; Farhadi et al., 2009). Lampert et al. (2009) have shown that representation through attributes even allows classification of unseen object classes that does not appear in training data. Similarly in our work, we represent environment as a set of attributes, allowing the robot to search for objects with most suitable attributes rather than looking for the specific object.

Predicting Sequences. Predicting sequence has mostly been studied in a Markov Decision Process (MDP) framework which finds optimal policy given the reward for each state. Because the reward function cannot be easily specified in many applications, Ng & Russell (2000) introduce inverse reinforcement learning (IRL) for an MDP, which tries to learn reward function from the expert's policy. Abbeel & Ng (2004) extend IRL to Apprenticeship Learning, based on the assumption that the expert tries to optimize an unknown reward function. However, such MDP-based approaches require a state representation which is hard to define for a human environment that can contain different number of objects, and stepping through different policies and running a reinforcement learning algorithm would be expensive given a large state space and a large action space.

Most similar to our work, Ratliff et al. (2006) introduces Max Margin Planning, where imitation learning is framed as a structured max margin learning problem. However, it has only been applied to the 2-dimensional path planning problem, which has much smaller and clear set of states and actions compared to our problem of sequencing different controllers.

3. Our Approach

A *program* is a sequence of primitives (low-level controllers in our case). In order to make programs generalizable, primitives should have the following two properties. First, each *primitive* should specialize to an atomic operation such as move_close, pull, grasp, and release. Second, a primitive should not be specific to a single overall activity or 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 easy to adapt to different robots by providing implementing primitives on the new robot.

For illustration, we write a program for "throw garbage away" in Program 1. Most programs could be written in such format, where there are many `if` statements inside the loop. However, even for a simple "throw garbage away" task, the program is quite complex.

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 planning or dynamic planning (Russell & Norvig, 2010; Koenig, 2001). In traditional, deliberative planning, a planning algorithm synthesizes a sequence of steps that start from given state and reaches given goal state. Current symbolic planners can find optimal plan sequences consisting of hundreds of steps. However, in dynamic environments, such long plan sequences often break down because unexpected events may occur during the execution of the plan sequence. A dynamic plan provides a much more robust alternative. At each step, the current state of the world is considered ("sensed') 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 required step in any possible world state bringing the agent closer to the overall goal. (There may be more than one suitable next step. In that case, a simple random selection from the feasible options works.) In complex planning domains, dynamic plans may become too complex. However, we are considering basic human activities, such as following a recipe for cooking. In this setting, dynamic plans are generally quite compact and effectively lead the agent to the goal state. Moreover, as we will demonstrate below, we can often 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 (or attribute) based representation for the entities in the environment. This attribute set represents the state of the world (and hence the program, in a dynamic planning setting). We can extract some features from the environment along with the action that will be executed in the body of `if` statement. With extracted features ϕ and some weight vector $\mathbf{w}$ 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 rewrite Program 1 in the form of Program 2.

Now all the `if` statements have the same form, where the same primitive is used in both `if` 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 the weight vector and joint feature map, as shown in Program 3.

The approach taken in Program 3 also allowed removing $find\_garbage\_can(e)$. Both Program 1 and Program 2 requires $find\_garbage\_can(e)$ which depends

Program 2 "throw garbage away."

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

Program 3 "throw garbage away."

```
Input: environment $e$, trash $a_1$
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 $\hat{p}_t = done$
```

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 explicitly encoding.

Program 3 provides a generic representation of a dynamic plan. We will now discuss an approach to learning a set of weights, which defines the particular overall activity or task. 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 Randon Field, and will use a maximum margin based approach to train the weight vector. Our empirical results show that such framework is effectively trainable with relatively small sets of example sequences. Our feature-based dynamic plan formulation therefore offers a highly effective and general representation to learn and generalize from action sequences accomplishing high-level tasks in a dynamic environment.

3.1. Model Formulation

We are given a set of possible primitives $\mathcal{P}$ to work with (see Section 4), Using these primitives, the robot has to accomplish a manipulation task $g \in \mathcal{T}$. The manipulation task g is followed by the arguments $\{g_{a1}, g_{a2}\} \subset \mathcal{E}$ which give specification of the task. For example, the program "throw garbage away" would have a single argument which would be an object id of the object

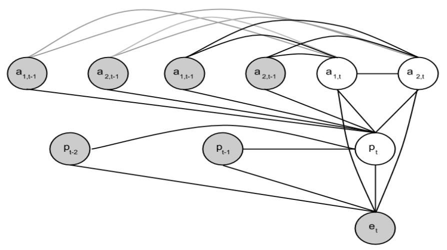


Figure 1. Figure showing Markov Random Field representation of our model at discrete time-step t. The middle layer represents the sequence of primitives, and the top layer represents the arguments associated with each primitive. The bottom node represents the environment. Goal task node consisting of g, g_{a1}, g_{a2} is not explicitly shown but is connected to all of nodes at time t as well as primitives at previous time-steps.

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 we will represent it with a joint set of attributes. These attributes would include information about the physical and semantic properties of the object as well as information about its location in the environment (see Section 4).

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 $S(\cdot)$ over the attributes of the environment, primitives and the arguments that represents the correctness of executing a primitive in the current environment for a task.

$$S(e_t, p_t(a_{1,t}, a_{2,t})) = w^T \phi(e_t, p_t(a_{1,t}, a_{2,t}))$$

In order to incorporate all the affects, the size of the attribute vector above would be quite big. We therefore decompose our score function using a model isomorphic to a Markov Random Field (MRF) shown in Figure 1. This also allows us to capture the dependency on the primitives and arguments executed in the previous time-steps. In the figure, the bottom node represents environment e_t at time t as well as the goal task (g, g_{a1}, g_{a2}) though it is not shown explicitly. The middle layer represents the sequence of primitives and the top layer represents the arguments associated with each primitive. 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:

$$S = S_{ae} + S_{pg} + S_{aet} + S_{aae} + S_{pae} + S_{ppt} + S_{paae}$$

The terms associated with a edge in the graph are defined as a linear function of respective features ϕ

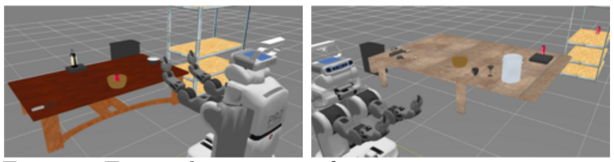


Figure 2. Figure showing two of our 13 environments in our evaluation dataset using 43 objects along with PR2 robot.

and weights w:

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

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

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

Using these edge terms and clique terms, our score function $S(\mathbf{x}, \mathbf{y})$ can be simply written in the following form, which we have seen in Program 3: $S(\mathbf{x}, \mathbf{y}) = \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})$.

3.2. 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 interaction (e.g., grasped) with: (i) the objects of interest given as task arguments g_{a1}, g_{a2}, and (ii) also with the objects of possible interest given as possible primitive argument candidate $a_{1,t}, a_{2,t}$. The robot should be aware of how far it is currently located from objects it will interact with. Therefore we add one feature for the distance from the robot to each primitive arguments and one binary feature which tells whether each primitive argument is already in the robot's manipulator.
- *Arguments-environment-task (ϕ_{aet})*: We add a binary feature vector of length 4 which indicates whether the objects of possible interest (two primitive arguments) is same as the objects given by the two task arguments. Another feature vector of length 4 is added to indicate whether task arguments and primitive arguments are in collision when viewed from the top in order to retrieve some spatial information between these objects.

  Before interacting with the object, it is often important to realize the type of the object (e.g. table, shelf) that is supporting it. Such realization will be relevant depending on what is robot trying to accomplish next in the sequence, which we can get some information by checking whether the robot is holding any object or not. For most of the primitives, the first primitive argument results in a higher level of interaction with the robot compared to the second primitive argument, we extract attributes of object that supports ('is below') first primitive argument. We store this information in a feature vector of size twice the length of attributes ($2l$) where we store extracted attributes in the first half or the second half if the robot is grabbing holding task argument object.

  The set of robot's possibly interested objects (primitive arguments) should be influenced by what task it is trying to accomplish. The binary matrix feature of size $|\mathcal{T}| \times 2l$ stores the object attributes of the primitive arguments $a_{1,t}, a_{2,t}$ on g^{th} row. And a binary vector of size $|\mathcal{T}|$ encodes on g^{th} row whether the robot is currently holding primitive arguments in hand.
- *Primitive-task (ϕ_{pg})*: Depending on the task information provided, the primitive should be selected differently. We create a $|\mathcal{T}| \times |\mathcal{P}|$ binary matrix where each entry indicates whether that primitive is relevant for that task at all. This binary matrix represents a co-occurrence between two terms. Only non-zero entry term in this matrix will be g^{th} row and ${p_t}^{th}$ column.
- *Arguments-environment (ϕ_{aae}).* To capture whether there is any spatial relation between objects of possible interest, we add one binary feature indicating whether primitive arguments $a_{1,t}, a_{2,t}$ are currently in collision with each other.
- *Primitive-arguments-environment (ϕ_{pae}).* Depending on the type of primitive to take, it would sometimes require primitive argument to be already grabbed or sometime requires not to be grabbed. We create a $|\mathcal{P}| \times 2$ where row for 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}).* Depending on the task, the robot would take different transition of primitives. Just like the feature in ϕ_{pg}, we use a binary co-occurrence matrix of size $|\mathcal{T}| \times |\mathcal{P}|^2$ where each cell represents specific transition between primitives and each row forms a transition occurrence for a task. We generate two of these binary matrices where one is concerning the transition from $t-2$ to t for task g

and the other concerning the transition from $t-1$ to t for task g.

- *Primitive-argument-argument(previous)-environment* (ϕ_{paae}). Depending on the type of primitive, the objects could match with previous arguments or sometimes it should not match because the interaction is over. Thus, the matrix of size $|\mathcal{P}| \times 8$ is created to represent for ${p_t}^{th}$ row to contain 8 binary values representing whether the two primitive arguments at time t is same as the two primitive arguments at $t-1$ or the two primitive arguments at $t-2$.

Attributes. Every object in the environment including tables and floors 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 were previously shown to be reliably extracted from RGB or RGBD images and was shown to be useful in different applications (Ferrari & Zisserman, 2007; Farhadi et al., 2009; Lampert et al., 2009; Koppula et al., 2011).

3.3. Learning

We use maximum margin based approach to train a single model for all tasks. The maximum margin approach fits our formulation since it assumes discriminant function is a linear function of a weight vector $\mathbf{w}$ and a joint feature map $\phi(x, y)$, and it has time complexity linear with the number of training examples when solved using the cutting plane method (Joachims et al., 2009). We formalize out problem as "1-slack" structural SVM optimization problem:

$$\min_{\mathbf{w},\xi\geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\xi$$

$$s.t. \quad \forall(\hat{y}^1, \cdots, \hat{y}^n) \in \mathcal{Y}^n, \forall \hat{y}^i_t \in \hat{y}^i$$

$$\frac{1}{n}\boldsymbol{w}^T \sum_{i=1}^{n}\sum_{j=1}^{|\hat{y}^i|}[\phi(x^i_j, y^i_j) - \phi(x^i_j, \hat{y}^i_t)] \geq \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{|\hat{y}^i|}\Delta(y^i_j, \hat{y}^i_t) - \xi$$

with the loss function defined as:

$$\begin{aligned}\Delta(y, \hat{y}) &= \Delta(\{p, a_1, a_2\}, \{\hat{p}, \hat{a}_1, \hat{a}_2\}) \\ &= \mathbb{1}(p \neq \hat{p}) + \mathbb{1}(a_1 \neq \hat{a}_1) + \mathbb{1}(a_2 \neq \hat{a}_2)\end{aligned}$$

With learned $\mathbf{w}$, we choose the next action in sequence by selecting the primitive that gives largest discriminant value: $\arg\max_{p_t\in\mathcal{P}, a_{1,t}, a_{2,t}\in\mathcal{E}} w^T\phi(e_t, p_t(a_{1,t}, a_{2,t}))$.

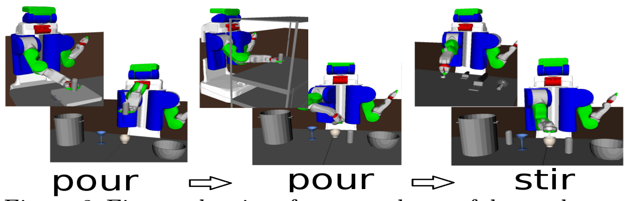


Figure 3. Figure showing few snapshots of learned programs forming higher level task of serving sweet tea which takes the sequence of pouring tea into a cup, pouring sugar into a cup and then stirring it.

4. Experiments

4.1. Data

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 the 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 actual sequence does not directly interact with liquid (obj16) but rather with a container of liquid (obj04), an empty cup (obj02), and an table (obj26).

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

- `stir(A)`: Given a liquid A, robot has to figure out ideal size stirrer (from several), and stir the liquid A on top of the table. The objects may not be on the table but on a shelf. In all of programs, the robot can only interact with the container of the liquid rather than the liquid itself whenever liquid needs to be carried or poured. Therefore our learning algorithm should learn this property.
- `pick_and_place(A,B)`: The robot has to place A on top of B where A could sometimes be under some other object.
- `pour(A)`: The robot has to find a bowl or a pot and pour the liquid A into it.
- `pour_to(A,B)`: The liquid A has to be poured into a container B. (A variant of the previous where the container B was specified.)
- `throw_away(A)`: The robot has to locate the

garbage can in the environment and throw out object A.

In order to learn these *programs*, we collected 127 sequences for 113 unique scenarios. We considered a single-armed mobile manipulator robot for these programs. In order to extract information about environments at each time frame of every sequence, we have implemented each primitive using OpenRAVE simulator (Diankov, 2010). 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 total of 736 instances of primitives. For ensuring variety of sequences, sequences were generated based on the 13 environments shown in Figure 2 using 43 objects.

4.2. Evaluation and Results

We have evaluated our data through 6-fold cross-validation for computing accuracies over primitives (and primitives with arguments). Because we predict next correct primitive and argument pair to be taken, even if a sequence is not correctly predicted at time t, for data at time $t+1$, we assume that the correct sequence was executed up to time t and tries to predict next best sequence. Figure 4(a) shows the confusion matrix for the prediction of our seven primitives. Our model is very robust for most of primitives.

With our dataset, our model was able to correctly predict next primitive along with correct set of arguments 90.0% on average (Table 1). Just considering the primitives, it was able to correctly predict 96.7% on average. For comparison, we have trained multiclass SVM to only learn correct primitives, using the same set of features. Because it only takes $\phi(x)$ rather than the joint feature $\phi(x,y)$, many features had to be adjusted to use previous timestep's primitives and arguments. It predicted well on some of the *primitives* but suffered greatly from *primitives* like `place_above`, `hold_above` and `follow_traj_pour` which will drastically impact constructing overall sequence even with correct argument selected.

The last column of Table 1 shows the performance on whether the complete sequence was correct or not. Over five different *programs*, our model was able to correctly construct complete sequence 69.7% on average. For example, on "pouring" *program*, it has learned not only to bring a cup over to the table but also to pick out the cup when there were multiple other objects like a pot, a bowl and a can that could look similar to a cup. Because of such large search space, if it was predicted at random, none of the sequences would be correct. Also, by varying the set of features, it is evident that without very robust primitive-level accuracies, it is unable to construct a single correct sequence.

These learned *programs* can form higher level tasks such as a recipe found online. For example, for serving sweet tea, it would require following steps of pouring tea into a cup, pouring sugar into a cup, and stirring it (Figure 3). We have tested four tasks in three environments for each task: *serve-sweet-tea*, *serve-coffee-with-milk*, *empty-container-and-throw-away*, and *serve-and-store*. Each four task can be sequenced in following manner by *programs* respectively: `pour_to` $\rightarrow$ `pour_to` $\rightarrow$ `stir`, `pour_to` $\rightarrow$ `pour_to`, `pour` $\rightarrow$ `throw_away`, and `pour` $\rightarrow$ `pick_and_place`. Out of total 12 scenarios, it was able to successfully complete the task 9 times.

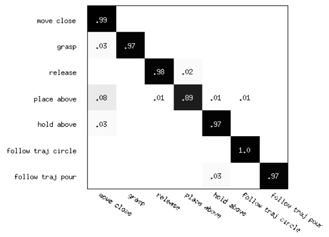
In an assistive robotics settings, the robots will be accompanied by an human observer. With a help from human observer, the performance can be greatly improved. Instead of choosing a primitive and argument pair that maximizes the discriminant function, the robot can present top 2 or top 3 primitive and argument pairs to human observer who can simply give feedback on the best option among those 2 or 3 choices. At initial timestep of the sequence, with only a single feedback given 2 or 3 choices, performance improves to 74.1% and 75.6% from 69.7% respectively (Figure 4(b)). If the feedback was provided through whole sequence with top 2 or 3 choices, it further improves to 76.7% and 81.4%. Furthermore, also for four high-level task (recipe) considered earlier also shows that with a single feedback at initial timestep of a each *program*, the results improves from 75% to 100% (Figure 4(c)).

5. Conclusion

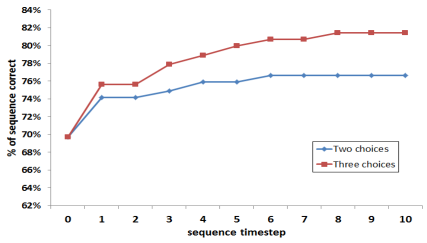
In this paper, we considered the problem of learning sequences of controllers for robots in unstructured human environments. We took a dynamic planning approach, where we represent the current state of the environment using a set of attributes. Conceptually, a dynamic plan can be captured by an loop that selects an appropriate primitive action at each discrete time-step, given the current state of the environment. Our primitive actions (or controllers) represent generic steps and can be used to compose a wide variety of high-level tasks, when appropriately sequenced together. To ensure that our dynamic plans (programs) are as general and flexible as possible, we use a attribute based representation of the environment, and train a set of parameters weighing 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

Table 1. Result of baseline, our model with variations of feature sets, and our full model on out dataset consisting of 127 sequences. The "prim" columns represent percentage of primitive correctly chosen regardless of arguments, and "args" columns represent percentage of a correct pair of primitive and argument. The last column shows average percentage of sequences correct over five *programs* we have evaluated.

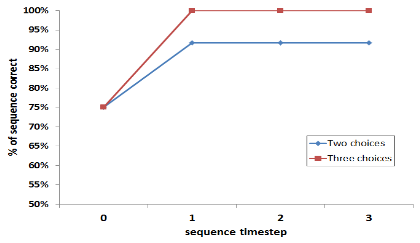
	move_close		grasp		release		place_above		hold_above		traj_circle		traj_pour		**Average**		**Sequence**	
	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg	prim	arg
chance	14.3	1.1	14.3	1.1	14.3	1.1	14.3	0.1	14.3	0.1	14.3	1.1	14.3	0.1	14.3	0.7	0	0
multiclass	99.6	-	90.4	-	95.7	-	68.5	-	79.7	-	100.0	-	14.7	-	78.4	-	-	-
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**



(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. Without any feedback in completely autonomous mode, the accuracy is 75%. With feedback (number of feedbacks on x-axis), the performance increases.

Figure 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 predicting programs.

activities that our approach can effectively learn dynamic plans for various complex high-level tasks.

References

Abbeel, Pieter and Ng, Andrew Y. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

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

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

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

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

Ferrari, Vittorio and Zisserman, Andrew. Learning visual attributes. In *NIPS*, 2007.

Green, Cordell. Application of theorem proving to problem solving. Technical report, DTIC Document, 1969.

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

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

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

Kautz, Henry and Selman, Bart. Planning as satisfiability. In *European conference on Artificial intelligence*, 1992.

Kemp, Charles, Goodman, Noah D, and Tenenbaum, Joshua B. Learning to learn causal models. *Cognitive Science*, 34(7), 2010.

Koenig, Sven. Agent-centered search. In *AI Magazine 22.4: 109*, 2001.

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

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

Miller, S., Van Den Berg, J., Fritz, M., Darrell, T., Goldberg, K., and Abbeel, P. A geometric approach to robotic laundry folding. *IJRR*, 2012.

Ng, Andrew Y and Russell, Stuart. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

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

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

Russell, S. J. and Norvig, P. *Artificial intelligence: a modern approach. Vol. 2.* Prentice Hall, 2010.

Thrun, Sebastian, Burgard, Wolfram, Fox, Dieter, et al. *Probabilistic robotics.* MIT press Cambridge, 2005.

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