A Framework for Teaching and Executing Verb Phrases

Daniel Hewlett and **Thomas J. Walsh** and **Paul Cohen**
Department of Computer Science, University of Arizona, Tucson, AZ
{dhewlett,twalsh,cohen}@cs.arizona.edu

Abstract

This paper describes a framework for an agent to learn verb-phrase meanings from human teachers and combine these models with environmental dynamics so the agent can enact verb commands from the human teacher. This style of human/agent interaction allows the human teacher to issue natural-language commands and demonstrate ground actions, thereby alleviating the need for advanced teaching interfaces or difficult goal encodings. The framework extends prior work in apprenticeship learning and builds off of recent advancements in learning to recognize activities and modeling domains with multiple objects. In our studies, we show how to both learn a verb model and turn it into reward and heuristic functions that can then be composed with a dynamics model. The resulting "combined model" can then be efficiently searched by a sample-based planner which determines a policy for enacting a verb command in a given environment. Our experiments with a simulated robot domain show this framework can be used to quickly teach verb commands that the agent can then enact in new environments.

Introduction

Verb phrases (like "walk over the hill") are the method humans most naturally use to communicate about actions with other human agents. However, the capability of artificial agents to learn and understand verbs is underdeveloped. Instead, agents designed in the Artificial Intelligence community often encode objectives in terms of goals (traditional planning) or reward functions (reinforcement learning, or RL). The difference is more than just diction. When judging whether an agent has performed a verb phrase such as "walk over", one must consider not just whether it reached a goal location, but *how* the agent got there. In addition, verbs have many desirable properties not shared by standard objective encodings, including the ability to inherit meaning from, or compose new meanings with, other verbs. Critically, verb meanings should also be *executable* by an agent in a given environment. The problem we consider in this work is that of learning verb models that support all of these properties with the help of a human teacher. Our solution to this problem provides a natural teacher-student interaction between a human and an agent where the human can communicate commands and label demonstrations via natural language and make use of the properties of verbs.

Our verb-learning and execution protocol extends work in apprenticeship learning (Abbeel and Ng 2005), where agents learn low-level dynamics from human teachers for a single domain class and task. Instead, our agents learn verb meanings that can be used to define completely new tasks. It also contrasts with work on verb-phrase recognition or classification (Tellex et al. 2010) because our system not only learns a model of the verb-phrase, but is able to plan and execute policies to enact it.

The specific contributions of this work are the following. We describe a method for representing and learning (from a teacher) verb-phrase meanings based on work in *activity recognition* (Kerr 2010), but unlike that line of research, we link the learned models to actual verbs in a way that maintains several desirable properties of verbs. We then compose these models with an environment's dynamics model in a manner that allows a planner to build a policy that will enact the verb in a given environment. This gives us an end-to-end system for learning verb meanings from human teachers and executing verb commands. As a proof of concept, we describe a specific instance of this combination using object-oriented MDPs to model environments, finite state machines to model verbs, and a sample-based planner (LRTDP) to do the planning. We empirically demonstrate the success of our verb learning/executing system in some simple test cases, including problems where a human teaches verbs that help bootstrap other verb meanings through composition.

Problem Overview

Formally, we will be considering a set of environments E at two levels: low-level object dynamics and abstract verb descriptions. We will discuss the complete data structures used to capture each of these in subsequent sections, but for now it suffices to consider the following components. The low-level object dynamics capture the changing physical state of an environment $e \in E$, specifically the configuration of a set of objects $\mathcal{O}$. Each object has a set of attributes (for instance its x and y coordinates). There is also a set of relational fluents F (such as *Above*(X,Y)) that are either true or false at each step for given object parameters. Together, these object attributes and fluents make up the *state* of the environment at every step. While the individual environments may

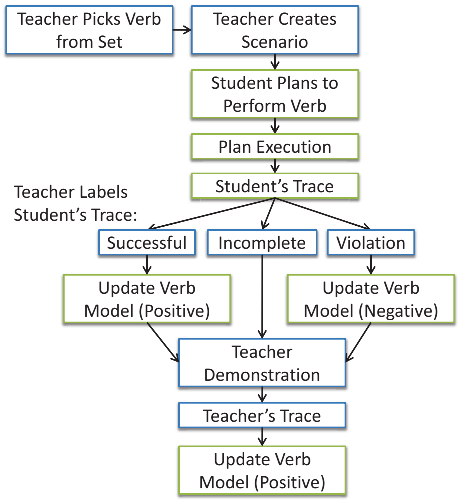
Teacher Picks Verb from Set
Teacher Creates Scenario
Student Plans to Perform Verb
Plan Execution
Student's Trace
Teacher Labels Student's Trace:
Successful
Incomplete
Violation
Update Verb Model (Positive)
Update Verb Model (Negative)
Teacher Demonstration
Teacher's Trace
Update Verb Model (Positive)

Figure 1: Our verb-teaching and execution protocol.

contain different numbers of objects with different attributes (different obstacles in different places), the fluents in each environment are considered to be consistent. Thus, each e is a specific instantiation of the *domain class* E. Algorithms exist for learning the dynamics of similar models (Diuk, Cohen, and Littman 2008), but they are always used in concert with a reward function or goal specification.

We consider a different paradigm where goals are communicated via verb phrases such as "Go around the block." While these phrases are not as concrete as goals or rewards, they are far more amenable to natural human-machine interaction. We consider a set of available verb phrases V such that $v \in V$ has a meaning based on a series of configurations of a set of fluents F_v that are either true or false on each timestep. For instance, Figure 2 illustrates (as a finite state machine) the verb *go*.

Since meanings are subjective, human guidance (in some form) is a necessary component for learning verbs. In our system, learning is done through incremental refinement of the verb semantics from interactions with a human teacher. The role of this teacher is twofold. First, the teacher can provide positive or negative demonstrations of a verb phrase being executed in an environment. This demonstration comes in the form of a sequence of states and actions (taken by the demonstrator). The second channel for human teaching is labeling examples, including both the teacher's own demonstration and executions performed by the agent. The full learning architecture we propose is illustrated in Figure 1. Formally, the protocol for every episode goes as follows.

1. The teacher can choose an environment $e \in E$, and a verb $v \in V$ and ask the agent to execute v in e.
2. The agent then uses its semantic model of the verb phrase along with a planner to produce and execute a policy that enacts this verb phrase. The human teacher can then label this execution *successful*, *incomplete* (for partial performance of the verb), or *violation* (for actions incompatible with performing the verb).
3. If the agent's behavior was judged by the teacher to be incomplete or a violation of the verb phrase, the teacher can provide a demonstration of the verb in the same environment.

The overall goal of the teacher is to teach the verb-phrase semantics to the agent in a complete enough form that for any $\langle e, v \rangle$ chosen adversarially (in a later testing phase), the agent will be able to enact v to the teacher's satisfaction. In the rest of this work, we provide details of the design choices, data structures, and algorithms we have used to implement this system and present some experimental results.

Learning Verb Meanings from Teachers

We begin by describing our representation and learning algorithm for verb phrases, starting with the properties of verbs we would like to maintain. Humans communicate a variety of content with verbs, including descriptions of events, and conditions that should not occur, such as "avoid the painted area" (see, e.g., (Levin 1993) for a discussion). Thus, an ideal verb representation should, at a minimum, support both recognition and execution, as well as the specification of constraints on execution, such as undesirable outcomes. It should also be specified at a qualitative level, since the same verb can be applied to different physical (or even metaphorical) events. To facilitate teaching verbs, a verb representation should support the sorts of operations on verbs that humans intuitively understand. For example, many verbs may be described in terms of other verbs, whether by specialization (e.g., "run" vs. "go") or composition (e.g., "fetch" is a sequence of simpler actions). There are likely several combinations of representations and learning algorithms that can accomplish this task. Here, we propose a representation of verb meanings based on finite-state machines that we argue possesses many of these characteristics.

Representation

Our representation for verb meanings is a type of finite-state machine we will refer to as a Verb FSM (VFSM). A VFSM differs from standard DFAs in two respects: each edge is labeled with a set of propositions, rather than a single symbol, and accepting states are partitioned into positive and negative states. Each intermediate (non-start and non-terminal) state contains a loop back to itself, allowing parts of the verb to take varying amounts of time. An example VFSM with abbreviated proposition names is shown in Figure 2. The utility of similar FSMs for recognition of activities, as well as a method for generalizing such FSMs, has been previously established (Kerr 2010), so we will not explore these issues here.

The VFSM is a qualitative representation in that it represents only propositional information, and only information about the ordering of states rather than their duration. It is also underspecified because each transition specifies only a subset of the propositions comprising the environment state s_e. Also, all of the propositions in the VSFM are expressed as relations over the arguments of the verb rather than specific objects (e.g., *InFrontOf*(Agent,Obstacle) rather than *InFrontOf*(robot1,bluebox2)). The full verb representation,

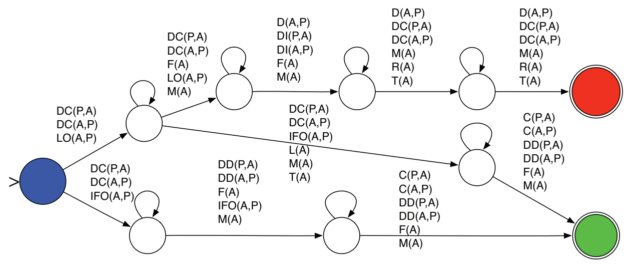

Figure 2: Example VFSM for the verb *go*(Agent,Place). The two paths ending in positive (green) terminals represent different ways to perform the verb, depending on the initial configuration. The upper path is incompatible with the verb.

then, is composed of the lexical form of the verb and its argument structure (e.g., *pick-up*(Agent,Object)), together with the VFSM.

Formally, the VFSM can be defined by a tuple (S, Σ, s_0, δ, A_p, A_n), where S is the set of propositional states, $\delta : S, \Sigma \mapsto S$ is the transition function , s_0 is the start state and A_p and A_n are the sets of positive and negative accepting states, respectively. Σ, the "alphabet" of the VFSM, is theoretically all the possible sets of true fluents, though we do not have to enumerate all of these to define the transition function. Because the transitions in the VFSM are labeled with sets rather than individual symbols, non-determinism can result because a given set of fluents may be compatible with more than one outgoing transition. To ensure that the VFSM operates deterministically, we define a function δ' to select a unique transition given an input set of fluents F. Let Σ_s denote the set of outgoing transitions from s, and σ_0 denote a transition back to the start state. Then, we can define a function δ' for the VFSM as:

$$\delta'(s, F) = \begin{cases} \text{argmax}_{\sigma \in \Sigma_s} |\sigma|, \text{ such that } \sigma \subseteq F \\ \sigma_0 \text{ if no such } s' \text{ exists.} \end{cases} \tag{1}$$

Ties are broken in a consistent manner. By always taking the most specific transition matching F, we ensure that the VFSM operates deterministically. The agent will progress through the VFSM until it reaches one of two kinds of terminal states: Positive accepting states indicate that the agent has successfully completed the verb, while negative accepting states indicate that the agent's actions were incompatible with performing the verb.

Learning Algorithm

The student constructs the VFSM from stored compressed traces of previous performances of the verb and teacher demonstrations. At the level of propositions, an episode can be viewed as a multivariate time series: each proposition is a variable and its value is either true or false. Such a time series is referred to as a propositional multi-variate time series, or PMTS. A PMTS can be compressed by collapsing together sequences of states that are identical to one another; this process is analogous to dynamic time-warping of real-valued time series. For brevity, we defer to Kerr (2010) for a detailed discussion of this compressed PMTS representation. To create the VFSM, we first build a DFA (specifically, a trie) out of these compressed sequences, where each leaf/terminal is an accepting state. Then, we minimize this DFA. In addition to reducing the number of states, minimization of a trie-like DFA ensures that the resulting DFA has precisely one accepting state, since the algorithm merges states with the same set of outgoing transitions. Thus, in a VFSM, which has two types of accepting states, minimization results in one positive accepting state and one negative accepting state. This property will prove useful for composition.

Each teaching episode is structured so as to provide the student with multiple opportunities to update the VFSM. First, the student's attempt to perform the verb in the environment the teacher has configured will result in a trace for the teacher to label. If the teacher labels the trace *successful* or *violation*, the compressed sequence of the student's performance will be added as a new (positive or negative) path to the VFSM. Next, the teacher may perform a demonstration, which will be added to the VFSM as a positive instance. Thus, the student will elaborate the VFSM throughout each teaching episode.

Composition of VFSMs

Since each VFSM has one start state and one positive accepting state, concatenation of VFSMs is simple, which allows for sequential verb composition. Concatenation is achieved simply by merging the positive accepting state of the first VFSM with the start state of the second VFSM. Note that, while at a formal level this combined VFSM accepts a language that is simply the concatenation of the languages accepted by each VFSM, planning with the combined VFSM is not equivalent to planning with the two VFSMs independently. This is because there are multiple paths through the VFSM and also because transitions in the VFSM are underspecified, meaning that some of the ways of executing the first verb may be incompatible with executing the second verb. Planning with the combined VFSM will ensure that no such paths are taken, while planning with the two VFSMs independently does not.

Object Oriented MDPs

In order to plan and execute a verb phrase in an environment, we need to model the environment's dynamics. In this work, we will employ a representation called an object-oriented MDP (OOMDP), which has been used previously to model video games (Diuk, Cohen, and Littman 2008) and robot dynamics as taught by human teachers (Walsh et al. 2010). We have chosen OOMDPs because their object-attribute dynamics mesh well with our domains of interest, but any relational fluent language, including STRIPS (Fikes and Nilsson 1971) could be used in our architecture.

In a standard (not OO) MDP formalism (Puterman 1994), a model $M = \langle S, \mathcal{A}, T, R, \gamma \rangle$ is comprised of states, actions, a transition function, rewards, and a discount factor. The long-term *value* of a state can be described by the optimal value function: $V^*(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')V(s')$. The policy corresponding to these values is $\pi^* : S \mapsto \mathcal{A}$. An OOMDP model compactly describes MDP-style dynamics for environments with many objects, each with attributes as described earlier. The set of relational fluents F are deterministically *defined* (such as *On*(X, Y) := X.yCoord = Y.yCoord +1) in terms of formulas over the object attributes. OOMDPs model the dynamics of actions ($T(s, a, s')$) using a special form of action schemas (templates for parameterized actions like *pick-up*(A, X)). OOMDP action schemas have conditional effects based on conjunctions c_i over the relational fluents (e.g. *Clear*(X) $\wedge$ *EmptyHand*(A)). Each c_i is associated with a set of possible stochastic effects (an object might be picked up or the arm might miss). These effects describe changes to the object attributes (e.g., a changing x coordinate). Thus at each time step, the state changes conditionally based on the current fluent configuration and action, which affect the object attributes, which in turn changes the currently true fluents.

While efficient algorithms have been developed for learning OOMDP dynamics with or without teachers (Walsh et al. 2010; Diuk, Cohen, and Littman 2008), our current work focuses on learning verbs and using the learned meanings to execute commands. Therefore, in this work we will assume that the full OOMDP model is provided with the environment chosen by the teacher.

Algorithm 1 Planning, Executing, and Updating with an OOMDP and VFSM

```
1:  Input: An OOMDP $M_e$, a VFSM $M_v$, Environment $e$ with initial state $s_{e0}$, and horizon $H$
2:  Create combined MDP $M_C$ with $S_C = (S_e \times S_v)$ and $T_C$ from (2) and $R_C$ from (3)
3:  Create the heuristic function $\Phi(\langle s_e, s_v \rangle) = -\rho(s_v)$
4:  $s_t = \langle s_{e0}, s_{v0} \rangle$
5:  while the $s_v$ component of $s_t$ is not terminal and $t < H$ do
6:      $a_t$ = Planner.recommend($M_C$, $\Phi$, $s_T$)
7:      Execute $a_t$ in $e$, observe $s'_e$
8:      Query $M_v(s_v, a_t, s'_e)$ for $s'_v$
9:      $s_{t+1} = \langle s'_e, s'_v \rangle$
10:     $t = t + 1$
11: Use the teacher's label $l$ of $\tau$ to update $M_v$.update($\tau$,$l$)
12: If the teacher provides a demonstration trace and label $\langle \tau', l' \rangle$, $M_v$.update($\tau'$,$l'$)
```

Combining the Representations

Now that we have representations for the low-level object dynamics and the high-level meanings of verbs, we need to combine the models in a way that facilitates efficient planning of a policy to execute the verb phrase in a given environment. An outline of the algorithm for the construction of the combined model and the planning and execution of the verb phrase is given in Algorithm 1, which we will refer to throughout this section. The first step in this process is to combine the state spaces of the two models (line 2). In the OOMDP representation of an environment e, a state s_e is fully determined (because of the derivable fluents) by the attribute values of each of the objects. In the VFSM representation, a state is simply the current node in the VFSM, s_v. To cover the full set of contingencies, we can construct a combined state $s_C = \langle s_e, s_v \rangle$ for all possible combinations of the two state components. The transition dynamics are then describable as:

$$T_C(s_C, a, s'_C) = T(s'_e|a, s'_e)\mathbb{I}(\delta(s_v, f[s'_e]) = s'_v) \qquad (2)$$

where $f[s]$ denotes the fluents that are true in s and $\mathbb{I}$ is an indicator function. While this state space is potentially quite large, it is unlikely that all or even most of the naive states are reachable because a combined state can only be valid if the fluents for the transition to s'_v are true in s_e. We now show how to construct reward and heuristic functions that will facilitate efficient planning in this combined state space.

Using the VFSM for Reward and Heuristic Functions

As mentioned earlier, executing a verb is different from goal-based planning tasks in that a sequence of stages must be completed to perform as the human teacher has requested. To that end, we have constructed a state space above that encodes the state of the verb execution, so we can define the reward function as:

$$R_C(s_C) = -\mathbb{I}(s_v \notin A_p) \qquad (3)$$

That is, the reward for a step is 0 if the agent is in an accepting terminal state in the VFSM, otherwise -1. Note that the reward function here depends solely on the VFSM, not the ground environment. Incorporating the environment's reward function R_e is possible, but potentially complicated, as we discuss in the Future Work section. Unfortunately, because of the size of the combined state space and the sparsity of reward, planners may have difficulty finding a policy to perform the verb in a reasonable amount of time. Instead, we would like to leave some "breadcrumbs" throughout the state space so that small rewards are given for completing each stage of a verb, without hindering the completion of the verb itself.

A simple mechanism for encoding such subgoals is to initialize the values of each state ($V(s_C)$) using a heuristic function $\Phi(s_C)$, which is equivalent to the reward shaping described above (Wiewiora 2003). We use (line 3) the heuristic function $\Phi(s_C) = -\rho(s_v)$, where $\rho(s_v)$ is the shortest distance in the VFSM from s_v to an accepting terminal state. This is the minimum number of stages remaining in the VFSM to complete the verb activity. This heuristic is admissible because it always takes at least one step (reward of -1) to transition between stages.

Using this heuristic will draw the planner towards areas of the state space where it is able to progress through stages of the verb, but will not stop it from backtracking if the currently explored branch does not allow for the verb's completion in this specific environment. For instance, if the verb "go around" is to be performed and an obstacle has a short way around and a long way (in terms of the VFSM states), the planner will search the short way first, but if there is no actual path to complete the verb (say because of a pit in this specific environment), the planner will be able to backtrack and find a (longer) way to complete the verb.

Planning in the Combined Space

We now have a fully specified combined MDP $M_C = \langle S_C, \mathcal{A}, T_C, R_C, \gamma \rangle$, where the actions $\mathcal{A}$ and discount factor γ come from the original environment MDP M_e (in our case an OOMDP). However, to actually perform the behavior consistent with the verb, we need to use a planner that can map M_C to a policy. In this work, we will be employing a sample-based planner, which plans by simulating trajectories through its learned model to determine the values of states and then chooses actions to maximize the discounted return. We chose such a planner over "all state" planners like Value Iteration (VI) (Puterman 1994) for two reasons. First, the combined state space may be very large and contain many states that are either not reachable from the initial state, or that do not need to be considered because they are far away. Planners like VI have a computational dependence on $|S|$, and therefore can become essentially unusable in such state spaces. Sample-based planners sidestep this difficulty by only guaranteeing to produce a policy for the start state s_0 (hence they may need to be called at every timestep, as in line 6). Second, sample-based planners tend to see a considerable performance boost from heuristic functions (Bonet and Geffner 2003), such as the one we constructed above.

A number of sample-based planners have been suggested in the literature including several variants of Real Time Dynamic Programming (RTDP) (Barto et al. 1995). In our experiments we used the LRTDP algorithm (Bonet and Geffner 2003), which is designed specifically for minimum-cost path planning problems, which is exactly the type of problem we have because the cost function we have specified (Equation 3) has only negative rewards for steps and terminal states. Using a more complex reward function (for instance one that takes into account a general R_e) may require more advanced sample-based planners.

Experiments

We conducted our experiments with a simulated mobile robot in the Gazebo robot simulator (http://playerstage.sourceforge.net/). The environment contained three types of objects: the robot, blocks, and locations. Each of these corresponds to an object type in the OOMDP. Example OOMDP attributes include x and y locations for all objects, orientations for the robot and blocks, and size and shape for blocks. Evaluation followed a train/test protocol, where the set of test environments was fixed but we varied the number of teaching episodes the student was exposed to. To assess the student's performance, we measured the average success rate across the scenarios, average planning time, and average plan length. In all experiments, the student's performance was deemed successful if it was a valid and complete execution of the verb, even if it was suboptimal.

Learning Verbs

The first verb we examine is very simple: *go*(Agent,Place). The agent was able to perform this verb in all our tests after essentially two episodes, which is to be expected given

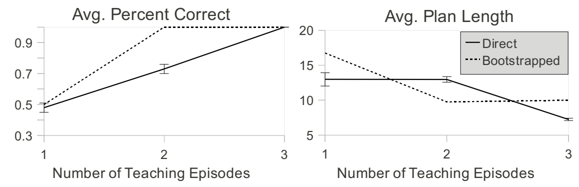

Figure 3: Average percent correct and average plan length for the verb phrase "go to X via Y."

the simplicity of the verb. Even so, the decline in planning time (1.91 to 0.39 seconds) and average plan length (9.03 to 5.22 steps) after the second episode is illustrative of how adding new paths through the VFSM improves the reward and heuristic functions, resulting in better plans and shorter planning time.

Earlier, we noted that verbs describe the manner in which an action should be performed rather than simply specifying a goal state. We chose "go around" as a simple example of this kind of verb. To perform *go-around*(Agent,Obstacle), the agent must move to the other side of the obstacle without touching it. To challenge the agent, we created test scenarios where one side of the object was blocked by a wall. After seeing two examples of this verb, one going around an object's right side, and the other its left side, the agent was able to perform *go-around* in all of our tests, with an average planning time of 37.83 seconds.

Combining Verbs

Our final experiment tests the performance of the robot on a verb phrase corresponding to "go to X via Y" that we will refer to as *go-via*. We compare two ways of arriving at this verb: Teaching the verb directly, and defining *go-via* in terms of the simpler verb *go*. Figure 3 compares the performance of the agent when taught *go-via* directly and bootstrapping *go-via* from *go*. When taught *go-via* directly, the agent requires three examples to perform all four tests reliably. By contrast, the teacher can simply define *go-via*(X,Y) as follows: *go*(Y) then *go*(X). With *go-via* defined this way, the teacher can teach the simpler verb *go* and the agent's ability to perform *go-via* will improve. The curve labeled "Bootstrapped" in Figure 3 shows the performance on the *go-via* tests after varying numbers of exposures to *go*. The agent can perform the tests reliably after only two episodes, but with higher average planning time (54.16 vs. 26.67 seconds) and plan length (9.75 vs. 7.24 steps) than the direct version. This example demonstrates how combining smaller verbs can speed up learning, but also shows how a VFSM trained directly on the target may better optimize behavior.

Related Work

A number of previous works have considered the problem of learning classifiers or recognizers for verb phrases from situated language data. Typically, these are created by mining a parallel corpus of episodes and linguistic utterances or labels. This approach has been applied to such problems as answering natural language queries against a video corpus of human activities (Tellex et al. 2010). Our work differs from these approaches because we provide a framework for executing the verbs, not just learning to recognize them. There has also been work on humans teaching noun phrases (like "stapler") for use in tasks, such as fetching (Saxena et al. 2007; Hewlett et al. 2007). Our work can be seen as complementary to theirs in that we are learning models of activities (like "fetch") that could be combined with such a system to handle the full semantics of verb phrases.

The teaching protocol we have used is similar to the apprenticeship learning framework (Abbeel and Ng 2005; Walsh et al. 2010). However, where those works focused on learning the dynamics of particular domains (in the latter case including an OOMDP representation), here we have focused on learning general verb phrases that can serve as human instructions in any number of environments, as long as the fluents for tracing a verb are the same in each setting. Our verb learning component and its translation to a reward function can also be thought of as a form of inverse reinforcement learning (Abbeel and Ng 2004), but the structure of our problems differs greatly from the assumptions usually employed in such work. Imposing a reward structure on an environment based on a command also has a history in hierarchical RL (HRL) (Dietterich 2000). There, upper levels of the hierarchy impose a reward function on lower levels during learning. While there are some similarities between our composed model and HRL, especially when verbs are being combined, their approach uses background knowledge (the hierarchy) to impose a policy bias while learning low-level controllers to achieve a goal. In contrast, our agents know how the low-level dynamics work, but attempt to learn a policy bias itself (the verb).

Conclusions and Future Work

In this work we have shown how to combine a teaching protocol for learning verbs from humans with dynamics models to create an interactive system for teaching and executing verb phrases. We have provided a method that learns a verb model, uses it to expand an environment's state space, and overlays both a cost and heuristic function. We presented early results demonstrating the success of a sample-based planner in the resulting model, and showed verb phrases can be efficiently taught and executed in this manner.

From these initial results, there are a number of avenues we plan to pursue. First, we assumed throughout this work that the low-level action dynamics (the OOMDP operators) were known and that the reward function of the ground environment R_e could be replaced by a uniform step-cost function. These decisions were made to highlight the verb learning and execution portion of our work, but neither of these is critical to our system. In fact, prior work (Walsh et al. 2010) has established that teaching by demonstration in a very similar protocol is a powerful tool for efficiently learning OOMDP operators; so a version of our system that learns both the verb semantics and low-level dynamics is a natural next step. In addition, incorporating R_e into M_C (in Equation 3) could allow the agent to reason about physical costs (like walking through dangerous terrain versus a smooth road) when it has the choice in performing a verb, though special care has to be taken when combining two different criteria functions.

Given the rich diversity of verb meanings, a system capable of representing the semantics of all or even most verbs remains a long-term research goal. However, characterizing the subset of verb meanings that VFSMs can represent is an immediate goal. While VFSMs can naturally represent verbs with primarily sequential structure (e.g., *fetch*), verbs with complex looping/conditional structures (e.g., *patrol*) may pose a challenge to the current system. A full study of verb composition is another area of future work.

References

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

Abbeel, P., and Ng, A. Y. 2005. Exploration and apprenticeship learning in reinforcement learning. In *ICML*.

Barto, A. G.; Bradtke, S. J.; Singh, S. P.; Yee, T. T. R.; Gullapalli, V.; and Pinette, B. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Bonet, B., and Geffner, H. 2003. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *ICAPS*, 12–31.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* 13(1):227–303.

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *ICML*.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 5:189–208.

Hewlett, D.; Hoversten, S.; Kerr, W.; and Cohen, P. 2007. Wubble World. In *AIIDE*.

Kerr, W. N. 2010. *Learning to Recognize Agent Activities and Intentions*. Ph.D. Dissertation, University of Arizona, Tucson, AZ, USA.

Levin, B. 1993. *English Verb Classes and Alternations: A Preliminary Investigation*. Chicago, IL: University of Chicago Press.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley.

Saxena, A.; Wong, L.; Quigley, M.; and Ng, A. Y. 2007. A vision-based system for grasping novel objects in cluttered environments. In *International Symposium of Robotics Research*.

Tellex, S.; Kollar, T.; Shaw, G.; Roy, N.; and Roy, D. 2010. Grounding spatial language for video search. In *International Conference on Multimodal Interfaces*.

Walsh, T. J.; Subramanian, K.; Littman, M. L.; and Diuk, C. 2010. Generalizing apprenticeship learning across hypothesis classes. In *ICML*.

Wiewiora, E. 2003. Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research* 19:205–208.