Teaching Robots to Execute Verb Phrases

Daniel Hewlett, Thomas J. Walsh, Paul Cohen
Department of Computer Science, University of Arizona, Tucson, AZ 85721, USA
{dhewlett,twalsh,cohen}@cs.arizona.edu

I. Introduction

Humans use verb phrases in a variety of ways, including giving commands ('jump over the hurdle') and relaying descriptions of events to others ('he leapt over it'). These simple verb phrases can actually convey a significant amount of content, including descriptions of events and conditions that should not occur, such as 'avoid the puddle.' Humans have a qualitative understanding of verbs that supports executing the same verb in multiple contexts, and with different arguments. Learning to perform verbs thus presents a challenging problem for imitation learning, requiring a level of generalization across both objects and the potential environments the verb can be enacted in.

Since verb meanings are subjective and their meanings complex, we adopt an imitation learning protocol where a human teacher can both judge and demonstrate a verb phrase to an agent. The protocol alternates between the agent executing a verb command from the teacher and the teacher then possibly enacting the verb phrase in the same environment. The objective of the agent is thus to infer the "meaning" of the verb phrase (essentially the teacher's cost function), and then act accordingly based on new verb commands. While this is certainly some form of inverse reinforcement learning (IRL), the non-Markovian and relational representation we use to model verb phrases is far different from the standard linear cost functions seen in IRL and we show this representation performs empirically better for many natural verb commands.

Our verb model, called the *Verb Finite State Machine* or VFSM, represents sequences of qualitative states, so it supports the natural "stages" of verb completion, the composition of verb meanings, and application to different environments. Specifying an agent's objective as a verb phrase with this representation allows us to learn and enact behaviors that would be difficult to capture using traditional AI encodings such as goals (traditional planning) or reward functions based on the state of an environment (reinforcement learning).

II. Framework Overview

Formally, we consider a set of environments E where each $e \in E$ has a set of objects $\mathcal{O}$ and a starting configuration (or distribution over start states) for the low-level properties (e.g x and y coordinates) of the objects and the agent. In addition, we assume that the agent is equipped with a qualitative understanding of the environment, specifically *relations* between objects using a set of *relational fluents* F where every grounded fluent is either true or false at a given timestep. Together, these attributes and fluents make up the *state* of the environment. This two-level representation (object attributes and relations) provides a qualitative description of the environment, without assuming that the dynamics can be captured at a purely relational level. These relations typically describe spatial relationships such as *LeftOf*(X,Y).

A human teacher communicates objectives to the agent via verb phrases such as 'go around the block', which we will represent formally as *go-around*(Agent, Block). We consider a set of available verb phrases V such that $v \in V$ has a meaning based on a series of configurations of fluents in F that are either true or false on each timestep. For instance, Figure 1 illustrates (as a pair of finite state machines) the verb *fetch*, where different stages of the verb are triggered by changes in the truth values of these fluents.

Learning in our system involves incremental refinement of verb semantics through interactions with a human teacher, a process that extends work from the apprenticeship learning literature [1]. The protocol for each episode is 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 verb model along with a planner to produce and execute a policy that enacts this verb phrase. The human teacher can then label this execution as *successful*, *incomplete* (for partial performance of the verb), or *violation* (of the verb's meaning).
3) If the agent's behavior was judged by the teacher to be incomplete or a violation, the teacher can provide a *demonstration* of the verb in the same environment. Both the agent's and the teacher's labeled trajectories can be used by the agent to refine its verb model.

The goal of the teacher is to teach the verb-phrase semantics to the agent in a complete enough form that the agent's chance of succeeding during subsequent teaching episodes is maximized.

III. Verb Representation

Our representation for verb meanings is a relational finite-state machine we will refer to as a Verb FSM (VFSM) (Figure 1). The VFSM differs from standard FSMs in that each edge is labeled with a set of propositions (relational fluents as described earlier), rather than a single symbol. Importantly, the VFSM is *not* a finite-state controller (FSC), where each state is mapped directly to an action. Rather, the VFSM accepts qualitative traces that match the verb semantics. Additional steps needed to execute the verb described by the VFSM are presented in Section IV. The VFSM is a qualitative representation in that it represents only propositional information, and only information about the ordering of states rather than

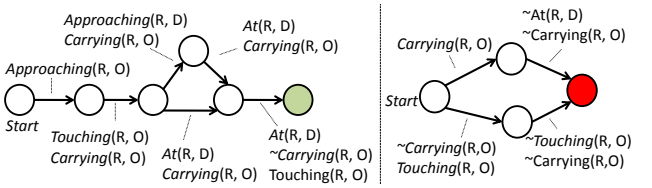

Fig. 1. An example VFSM (shown for clarity as two machines, for acceptance and for rejection) for *deliver*(Robot,Object,Destination). The left machine accepts "successful" examples while the right one accepts "violations".

their duration. It is also underspecified because each transition specifies only a subset of the propositions comprising the environment state. Conceptually, this VFSM is a combination of two simpler FSMs, one representing correct verb behavior and another encoding behavior that violates the semantics of the verb. Also, each intermediate state contains a loop back to itself (omitted from the diagram for clarity), allowing parts of the verb to take varying amounts of time. The utility of using a similar FSM for recognition of activities has been previously established [2], showing that the FSM transitions capture the natural stages and choices involved in verb completion.

A. Learning Verb Meanings from Teachers

The VFSM is created by combining labeled traces of behavior, excluding traces that are subsumed by others. FSM construction is conservative in that only the instances labeled as "success" or "violation" are included in their respective FSMs, a cautious approach inspired by apprenticeship learning results showing that acting based on the most specific hypothesis allows the teacher to safely drive the generalization process [1]. Each agent or teacher trace yields a sequence of sets of relations, which defines a simple VFSM as a single path between the first state and last state. By combining the start states of many such paths, as well as the end states, a single VFSM can be created, as shown in Figure 1.

IV. Executing Verbs

We now describe how the VFSM can be combined with a representation of the robot's environment and modern planning techniques to execute the verb. We begin by describing a model from the reinforcement-learning literature that captures both relational information and low-level robot dynamics.

A. MDP Representation

In order to leverage the VFSM model, a robot needs a relational view of its environment, which could in principle be provided by any relational MDP. However, since the dynamics of complex physical environments cannot be captured by purely relational models (like STRIPS), we will use a two-level model called an object-oriented MDP (OOMDP) from [3]. In an OOMDP, each state consists of a set of objects with attributes, as well as a set of relations between the objects. The set of relations are defined based on object attributes (such as *On*(X, Y) := X.yCoord = Y.yCoord +1). Actions have stochastic affects on object attributes, which in turn may causes changes to the relational state.

Recall that the VFSM does not directly encode a policy for verb execution. Thus, to plan for verb execution, we must combine the dynamics model for the environment M_e (an OOMDP) and the qualitative verb model M_v (a VFSM). This requires combining the state spaces and transition functions of the two models and using the terminal states of the VFSM to indicate reward. Specifically, we build a combined MDP $M_C = \langle S_C, A, T_C, R_C, \gamma \rangle$ where A and γ come from the OOMDP but the states $S_C = S_E \times S_V$ are any pairing of an OOMDP environment state and a VFSM state. The transition function incorporates VFSM transitions. The reward function is based only on the completion of the verb, 0 for accepting states in the VFSM, and otherwise -1.

To mitigate the size of the combined state space and the sparsity of reward, we encode "breadcrumbs" throughout the state space so that small rewards are given for completing each stage of a verb. A simple mechanism for encoding such subgoals is to initialize the values of each state ($V(s_C)$) using a *heuristic function* $\Phi(s_C)$. We use 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 state without violating the verb semantics. This is the minimum number of stages remaining in the VFSM to successfully complete the verb activity. This heuristic draws the planner's search towards areas where it can 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 the current environment.

B. Planning in the Combined Space

We now have a fully specified combined MDP $M_C = \langle S_C, \mathcal{A}, T_C, R_C, \gamma \rangle$, as well as a heuristic function $\Phi(s_C)$. To actually perform the verb, the agent requires a planner that can map states in M_C to actions. When the environment is deterministic, we can use the cost and heuristic functions constructed above with simple A* search [4] to find the optimal policy. Stochastic environments require a more general planner. If the MDP is small enough, then standard MDP planning algorithms like Value Iteration (VI) could be used. However, because OOMDPs usually have a large ground state space, and S_C contains the cross product of all of these states with all of the S_v states, the computational dependence of algorithms like VI on $|S_C|$ is likely to be prohibitive. Instead, our experiments with stochastic environments employ Upper Confidence for Trees (UCT) [5], a sample-based planner that sidesteps a dependence on $|S_C|$ by only guaranteeing to produce a policy for the start state s_0. While UCT is not expressly built to utilize a heuristic function such as our $\Phi(s_C)$ described above, we simply translated this heuristic into a *reward shaping* function [6] that served to guide UCTs search to areas where stages of the verb could be quickly completed.

V. Experiments

We evaluated the performance of the VFSM on a set of verbs, and against baseline methods described later in this section. The verbs tested were the following: *go*(Robot,Target): Travel to the target location; *deliver*(Robot,Object,Target):

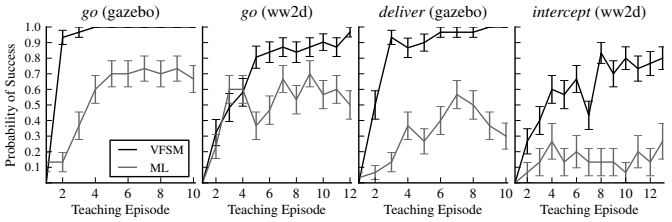

Fig. 2. Experimental results for execution of verb phrases. Error bars show one standard deviation (n $\geq$ 15).

Travel to the object, pick it up, then travel to the target and place the object there; *intercept*(Robot,Enemy,Target): Make contact with the enemy robot before it reaches the target. Our experiments used two simulated mobile robot domains: the Gazebo robot simulator[1], where we ensured that the robot's actions had deterministic effects, and the Wubble World 2D (WW2D) simulator[2], where actions had stochastic effects. Both environments contained similar objects and relations.

As a baseline for execution, we also implemented the Maximum Likelihood verb model of Kollar et al. [7], which we will refer to as ML. ML proceeds by iteratively simulating all possible sequences of actions up to some depth (a breadth-first search), and then executing the sequence that maximizes the likelihood of the verb under a Naive Bayesian model. This process terminates when all possible actions would decrease the likelihood. ML only models the percentage of time that each relation is true, and assumes all relations are independent of each other. We also implemented the Inverse Reinforcement Learning (IRL) method of Abbeel and Ng [8] as a baseline.

During each teaching episode, the robot is asked to perform the verb in a situation it has not encountered before, which is a form of hold-one-out evaluation. From many learning trajectories, we can estimate the probability of success after a given number of teaching episodes. This measure of performance for the VFSM and the ML baseline at each teaching episode is shown in Figure 2. The order of presentation by the teacher was randomized for each learning trajectory. Success is determined by an automatic validation procedure.

The simplest verb, *go*, was tested in both domains, Gazebo and WW2D. In each case, the VFSM was able to master the verb, achieving a high rate of success after only a few training examples, as shown in Figure 2. However, the ML baseline was not able to match the performance of the VFSM, reaching a plateau lower than the 90% success rate reported by Kollar et al. [7]. While the VFSM explicitly models the completion of a verb, ML instead relies on a decrease in the likelihood function for termination, meaning that it does not always stop at the goal, reducing its success rate.

On the more complex verb *deliver*, VFSM significantly outperforms ML because it explicitly represents the sequence of stages involved in delivering, and provides incremental feedback to the planner based on the VFSM state. The failure of ML to master *deliver* is not unexpected, as Kollar et al. reported a low rate of success (29%) for the similar verb *bring* [7]. However, the reasons for this failure underscore the importance of the sequential nature of the VFSM for modeling verbs. ML struggles with *deliver* because the features it is trained on are order-independent averages, making it difficult to model the sequential semantics of *deliver*. Since ML attempts to match these averages, it frequently performs behaviors inappropriate for the current stage of verb execution, such as briefly dropping the object so that the *Holding*(R,O) relation will be true "less often."

The verb *intercept*, in which the enemy is another agent moving along a known path to a target location, is perhaps the most difficult verb because the agent must act effectively in a constantly changing stochastic environment. As shown in Figure 2, the VFSM representation outperformed the ML baseline significantly on this verb. The ML method does not learn a model of verb constraints, and relies on a full BFS, limited look-ahead and thus planning precision.

A. Comparison to Inverse Reinforcement Learning

Our examination of the Inverse Reinforcement Learning (IRL) method of Abbeel and Ng [8] revealed properties not well suited to a general verb representation. Because IRL makes a linear cost assumption, its behavior on a "goal oriented" verb like *go* was highly dependent on the amount of "padding" at the end of an example showing the agent sitting at the destination. With a large amount of padding the agent successfully completed all test instances with one teacher trace, but without padding, even with all the training data, it failed in all the test examples because it preferred *approaching* the goal to actually *reaching* it. For a verb like *deliver*, IRL had trouble finding a weight for *Carrying*, since this was a "good" relation as long as the agent was not at the destination (a non-linear relationship). These results indicate that while IRL can learn certain verbs very quickly, it cannot capture the full range of complicated (non-linear) verb definitions. Our VFSM captures such relationships (the stages of verb completion) and can be seen as a step towards generalizing IRL techniques to more complex teacher cost functions.

[1]http://playerstage.sourceforge.net/

[2]http://code.google.com/p/wubbleworld2d/

REFERENCES

[1] T. J. Walsh, M. L. Littman, and C. Diuk, "Generalizing Apprenticeship Learning across Hypothesis Classes," in *ICML*, 2010.

[2] W. Kerr, A. Tran, and P. Cohen, "Activity Recognition with Finite State Machines," in *IJCAI*, 2011.

[3] C. Diuk, A. Cohen, and M. L. Littman, "An object-oriented representation for efficient reinforcement learning," in *ICML*, 2008.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2003.

[5] L. Kocsis and C. Szepesvari, "Bandit Based Monte-Carlo Planning," in *ECML*, 2006.

[6] E. Wiewiora, "Potential-Based Shaping and Q-Value Initialization are Equivalent," *Journal of Artificial Intelligence Research*, vol. 19, pp. 205–208, 2003.

[7] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Grounding Verbs of Motion in Natural Language Commands to Robots," in *International Symposium on Experimental Robotics*, 2010.

[8] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML*. New York, New York, USA: ACM Press, 2004.