Learning From Stories: Using Crowdsourced Narratives to Train Virtual Agents

Brent Harrison and Mark O. Riedl
School of Interactive Computing, Georgia Institute of Technology
Atlanta, Georgia, USA
{brent.harrison, riedl}@cc.gatech.edu

Abstract

In this work we introduce *Quixote*, a system that makes programming virtual agents more accessible to non-programmers by enabling these agents to be trained using the sociocultural knowledge present in stories. Quixote uses a corpus of exemplar stories to automatically engineer a reward function that is used to train virtual agents to exhibit desired behaviors using reinforcement learning. We show the effectiveness of our system with a case study conducted in a virtual environment called *Robbery World* that simulates a bank robbery scenario. In this case study, we use a corpus of stories crowdsourced from Amazon Mechanical Turk to guide learning. We evaluate Quixote under a variety of different conditions to determine the overall effectiveness of the system in Robbery World.

Introduction

Video games are a medium in which virtual agents must coordinate their behavior with people in many different ways. These agents can serve as enemies that the player must interact with to overcome, or they can serve as companions meant to instruct or aid the player in some way. They can also serve as background characters meant to make a game feel more immersive through believable interactions. Despite their importance, creating these agents can prove difficult without a non-trivial amount of programming knowledge. Even using a self-contained machine learning tool to train agents requires some amount of programming/computer science knowledge, which makes it difficult for non-programmers to perform this task. In this work, we make agent training more accessible to non-programmers by enabling them to train virtual agents through natural communication.

Methods such as *apprenticeship learning* (Abbeel and Ng 2004) or *learning from demonstration* (LfD) (Argall et al. 2009) seek to make agent and robot training more accessible to non-programmers by allowing people to provide a corpus of exemplar demonstrations of optimal behavior to aid in learning. This improves on upon traditional learning-based approaches as providing these demonstrations typically does not require extensive programming knowledge. These demonstrations, however, typically come in the form of complete trajectories in the environment that the agent or robot will be acting in, thus making it a requirement that authors have prior knowledge of the agent's environment and how it works.

In this work we introduce the idea of using a natural source of human communication to train machine learning algorithms: stories. Our system, which we call *Quixote*, uses stories told by humans to train virtual agents to exhibit specific behaviors. This task is similar to that of LfD algorithms except that our technique learns from stories told about a task rather than from explicit demonstrations of said task. The primary difference between stories and demonstrations, as well as the primary challenge in dealing with stories, is that stories are more unconstrained than demonstrations since authors have no prior knowledge about specifics of the environment. Also, many different stories could all correctly describe the same task, or authors could skip steps that they feel are obvious or do not need mentioning. Further, storytelling is non-Markovian in that some events that occur are influenced by events that happened far in the past. This can make it especially difficult to utilize story information for training since most agent environments are assumed to be Markovian.

Quixote addresses many of these issues by first cleaning an initial story corpus using the technique outlined by Li *et al.* (2013). This allows for Quixote to reconcile the exemplar stories with each other and fill in any gaps that may exist, which makes learning a more manageable task. From there, Quixote uses this new corpus to define the space of acceptable behaviors that is then turned into a reward function that can be used to train reinforcement learning agents.

To explore the effectiveness of our system, we present a case study in which we use crowdsourced narratives to train a reinforcement learning agent in Robbery World, a virtual environment that is meant to simulate a bank robbery scenario. In this case study, we show how a reinforcement learning agent learning agent would perform in Robbery World without story guidance and then compare this behavior to the behavior produced by Quixote under various conditions.

Related Work

Recently there has been an increased focus on *interactive machine learning*, which seeks to augment machine learning algorithms with the ability to learn from human feedback or demonstrations directly. The type of interactive machine learning that is most closely related to our own work is *Inverse Reinforcement Learning* (IRL). IRL attempts to learn the reward function that best describes a corpus of policy examples (Ng and Russell 2000) or policy trajectories (Abbeel and Ng 2004). Early work in this area required either complete policy examples or complete trajectories in order to learn. This requirement was relaxed through the introduc-

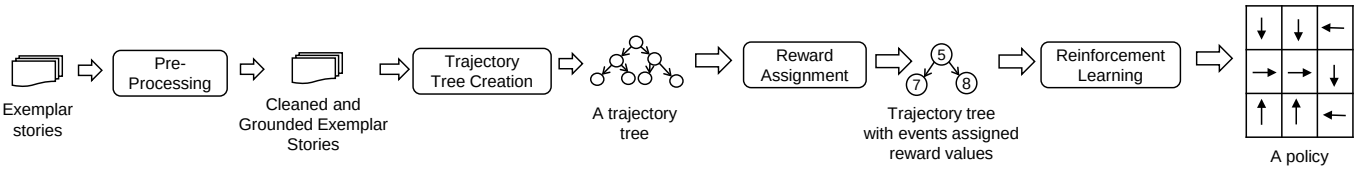

Figure 1: The Quixote system workflow.

tion of techniques such as Bayesian IRL (Ramachandran and Amir 2007) and maximum entropy IRL (Ziebart et al. 2008). There has also been work on relaxing the assumption that all example policies or trajectories are correct (Grollman and Billard 2011). Researchers have also sought to derive behaviors from natural language commands (Lignos et al. 2015; MacGlashan et al. 2015).

The problem that we solve with Quixote is fundamentally different than the problem posed in IRL. While we are attempting to derive reward functions based on a corpus of examples, we make different assumptions about what these examples represent, which leads to a different understanding of how to approach the problem. In this work, we assume that our example stories define a space of believable behaviors. Rather than design a reward function that can reproduce all of these examples, we are trying to create a reward function that produces behaviors that fall within this space.

There is also a branch of research that explores how human feedback about agent learning can be integrated into agent training. These systems use human reward signals to shape agent behavior (Knox and Stone 2009; Judah et al. 2010; Loftin et al. 2014). Our work differs from this work in that we do not seek to dynamically shape agent behavior. Quixote aims to derive what correct behavior is by looking at examples of desired behavior.

Reinforcement Learning Background

The Quixote system uses a set of exemplar stories to construct reward functions that can be used to train reinforcement learning agents. Reinforcement learning (Sutton and Barto 1998) is a technique that is used to solve a Markov decision process (MDP). A MDP is a tuple $M =< S, A, T, R, \gamma >$ where S is the set of possible world states, A is the set of possible actions, T is a transition function $T : S \times A \rightarrow P(S)$, R is the reward function $R : S \times A \rightarrow \mathbb{R}$, and γ is a discount factor $0 \leq \gamma \leq 1$.

Reinforcement learning first learns a policy $\pi : S \rightarrow A$, which defines which actions should be taken in each state. In this work, we use Q-learning (Watkins and Dayan 1992), which uses a Q-value $Q(s, a)$ to estimate the expected future discounted rewards for taking action a in state s. Reinforcement learning allows the agent to fill in any gaps that may exist in the stories due to authors not having prior knowledge about the agent's environment. Thus, the agent is able to take several actions, should it need to, in between plot points. Reinforcement learning also allows the agent to deviate from the stories it has been told if doing so will allow it to more efficiently reach a goal state.

The Quixote System

Quixote learns to coordinate agent behaviors in order to enact a scenario in uncertain environments which are possibly inhabited by a human. A high level flowchart of the Quixote system can be seen in Figure 1. The Quixote system works by first taking in a set of exemplar natural language stories describing the same task and pre-processing them to filter out noise and determine the possible ways that the task can be completed. The resulting cleaned story corpus is then converted into a trajectory tree which encodes every possible story believed, with confidence, to exist. This tree is then used to engineer a reward function which is used to train a reinforcement learning agent to exhibit the desired behaviors. In this section, we will describe each of these steps in greater detail.

Pre-processing

Initial input into the Quixote system is a set of exemplar stories written in natural language about a given task. Since human authors created these stories, it is likely that this initial corpus contains some amount of noise. For example, this set of exemplars may contain different stories that still correctly describe the same scenario, or some authors may skip events in the story or use different language to convey the same event. Some of these stories can also contain errors or events that do not relate to the scenario being described, or they could contain actions that do not exist in the virtual world that the agents will inhabit. This is why Quixote cannot use this initial corpus to train agents directly.

To mitigate these problems, Quixote first pre-processes the initial story corpus by automatically cleaning the dataset, and then determining which story events correspond to labels of the actions that agents can take in the virtual world using natural language processing techniques.

Automatic Story Corpus Cleaning As mentioned previously, the stories contained in the initial training corpus are likely to be noisy and variable. Multiple different stories could all correctly describe the same event, plot events could be skipped all together, and the stories could contain errors. Thus, the first part of the preprocessing step involves using the approach proposed by Li *et al.* (2013) to generate a clean set of stories.

This technique first involves clustering natural language sentences according to semantic similarity. These clusters are referred to as *events*. If any sentences do not cluster into a event then they are discarded. This way the technique helps to reduce the noise caused by errors or variable language in the corpus of exemplars. Second, this technique learns how events can be ordered as well as different ways in which the story can be told. Thus, noise as a result of misordering events is filtered out. To avoid confusion between the initial, human-authored story corpus and this cleaned story corpus, we will refer to the clean story corpus as a *plot corpus* and members of this corpus as *plots* for the remainder of this paper. Whereas stories in the original corpus are made up of

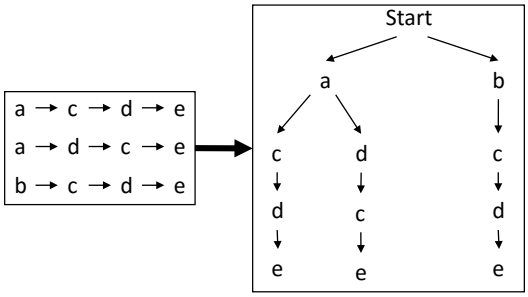

Figure 2: A set of input plots and the resulting trajectory tree.

sentences, plots consist of plot events which are, themselves, clusters of sentences.

Action Correspondence One of the primary contributions of Quixote is the ability to train virtual agents with little prior knowledge of the specifics of their environment. Since authors do not possess this prior knowledge, it is likely that some of the plot events contained in the story corpus will not exist in the agent's environment and vice-versa. In this step, Quixote attempts to identify correspondence between known plot events and some subset of the actions available to the agent. To do this, Quixote calculates the similarity between each event cluster and a natural language action label describing each action available in the environment.

In order to calculate similarity, each plot event sentence and each action label are translated into vectors. These sentence vectors are created by first converting each word in each sentence or action label into a word2vec word vector (Mikolov et al. 2013) and then calculating their sum. Then Quixote measures similarity by calculating the following:

$$sim(E, A) = \frac{1}{n_E} \sum_{e \in E} cos(e, A), \tag{1}$$

where E is an event cluster, A is an action label vector, n_E is the number of sentences contained in a plot event E, e is an individual event sentence vector, and $cos(e, a)$ is the cosine similarity between an event sentence vector e and an the action label vector A. In other words, we calculate similarity between a plot event and an action label by calculating the similarity between each *sentence* in a plot event and an action label and then take the average of those values.

To determine correspondence, Quixote uses a similarity threshold. Any actions that are above the similarity threshold for a given plot event are said to correspond with the plot event in question.

The final step of this process is to remove from every plot all plot events that do not correspond to an environment action. For each story in the training corpus, plot events that do not correspond to any environment actions are removed.

Trajectory Tree Creation

After pre-processing, the Quixote system uses the resulting plot corpus to derive a trajectory tree. This is done to enable the agent to track its progress through a plot. Without this, the agent becomes susceptible to repeatable actions. In stories it is not uncommon for authors to talk about events that are normally repeatable (such as entering or exiting buildings). If the agent cannot monitor its progress through the story and it is rewarded for such actions, then it is possible that it will repeat such actions infinitely (in order to maximize reward).

A trajectory tree is a structure that encodes each plot in the plot corpus. A traversal of the tree from root to leaf is a unique plot that exists in the plot corpus. An example of a plot and its resultant trajectory tree is shown in Figure 2.

Reward Assignment

Using this trajectory tree, we assign rewards to actions or states that exist inside the agent's environment. To do this, we incorporate the tree as part of the agent's world state directly. Thus, as the agent explores its environment it also keeps track of what plot events it has completed and what plot events it needs to complete in the future. Rewards are assigned to those actions that advance the plot as determined by the trajectory tree. When the agent receives a reward, it also advances in the trajectory tree.

Recall that during pre-processing we identified environment actions that corresponded with plot events. Also recall that it is possible that several environment actions may correspond to the same plot event. To account for this, the agent will receive a reward whenever it performs any action that corresponds with plot event that will advance the plot. Each action's reward, however, is weighted by its similarity to each plot event.

Reinforcement Learning

Once the reward function has been specified, we use reinforcement learning to find the optimal policy for a given environment. By using reinforcement learning, the agent is able to fill in any gaps that may still exist in the plot corpus. These gaps can exist because plot events are removed if they do not correspond to an action in the RL environment. Gaps can also exist because it may take several intermediate actions to complete a plot event in the MDP. For example, a plot event for going inside a store may require intermediate actions involving the agent navigating to the store first. Reinforcement learning allows the agent to learn for itself the most efficient sequence of actions to move from one plot event to another.

The trajectory tree also allows the agent to to model non-Markovian behavior that may exist in the plots. Conceptually, encoding the trajectory into the world state breaks up the learning problem into subtasks based on how the tree branches. The subtask that the agent learns is how to optimally get from its current plot event to one of its children in the trajectory tree. The agent only needs to determine at any time what is the optimal policy to get to the next plot event. For example, consider the trajectory tree shown in Figure 2. If the agent has just completed plot event *a*, then its current task is to find the optimal sequence of actions that enable it to complete either plot event *c* or plot event *d*. Here, the reinforcement learning agent is iteratively solving simple MDPs rather than solving a single complex MDP. This allows the agent to learn non-Markovian behaviors (since they will be encoded in the trajectory tree) and avoid infinite rewards due to rewarding repeatable actions (since performing the action will transition the agent into a "different" MDP with different rewards).

Rolling the trajectory tree into the world state does, however, affect the size of the MDP. Since the trajectory tree determines how the MDP task is divided into subtasks, the size of the MDP grows linearly with the number of nodes in the trajectory tree. While learning, the reinforcement learning agent must, essentially, explore each branch of the trajectory tree, which will increase training time. This is acceptable as Quixote is meant to be run as an offline process and should only need to be run once per environment.

Case Study

To show the effectiveness of the Quixote system, we perform a case study in a virtual environment called *Robbery World*. This case study explores how well Quixote is able to train the virtual characters in Robbery World to act out a bank robbery using crowdsourced narratives written in natural language as a guide. We will first examine how reinforcement learning performs using common reward functions without any narrative guidance. Then we will examine performance using a Quixote-engineered reward function using perfect, hand-authored correspondence. These experiments define the upper and lower bounds in terms of performance that we can expect from Quixote in Robbery World. We will then explore part of this spectrum by showing how Quixote performs using natural language similarity thresholds of 0.6, 0.7, and 0.8.

In these experiments, we are concerned with whether or not a policy will result in behavior that is *consistent* with the plot corpus. We say that a policy is consistent with the plot corpus when the optimal sequence of actions produced by the policy corresponds to a plot used for training.

Robbery World Domain

The Robbery World domain is a virtual world meant to simulate a bank robbery. It contains the following three characters: John the bank robber, Sally the bank teller, and the police. Each character can be at one of two possible locations: inside the bank or outside the bank. The police are an exception to this in that they do not begin in either location. Once the police *arrive* outside the bank, however, they are restricted to moving between those two locations like the rest of the characters. John begins the scenario outside the bank while Sally begins the scenario inside the bank.

Each character in Robbery World has an inventory that can contain objects that exist in Robbery World. The items in Robbery World are: a cash register, a bag, a note, a gun, a revolver, and money. The cash register and the bag are special in that they can contain the money in their inventories. Also, the cash register cannot be contained in any character's inventory. It is a static fixture of the world.

There are also many different actions that the characters in Robbery World can take. At a high level, the general actions in Robbery World consist of:

- Movement: a character moving from one location to another
- Give/take item: a character giving or taking an item from another character
- Show gun/revolver: a character shows the gun/revolver to another character
- Pull out gun/revolver: a character pulls out the gun/revolver
- Shooting: a character shooting another character
- Opening the cash register: a character opening the cash register
- Stashing money: a character placing the money in the cash register or the bag

In addition to these general actions, some of the characters can perform special actions based on their role in Robbery World. These special actions are: *Sally presses the alarm, Sally calls the police, The police arrive outside the bank,* and *The police arrest a character*.

In total, there are 97 unique actions that can be performed in Robbery World, but only one action can be performed at a time. The Robbery World scenario ends when either John leaves the bank with the money in his inventory, John leaves the bank with the bag containing the money in his inventory, John is arrested, or John is killed.

Robbery World is notable in that there are many characters present in the environment that each have their own set of actions they can take, making the environment quite complex. In addition, this domain highlights the difference between how a human might expect a robbery would take place and what an agent would do in this environment.

Reinforcement Learning Parameters

We used Q-learning in conjunction with ϵ-greedy exploration for training in each experiment. For this study we define ϵ to be 0.8 and then slowly decay it over $500,000$ learning episodes for all similarity thresholds. We chose this number of learning episodes because it proved a sufficient number of episodes for convergence across all experiments. In addition, we use parameters $\gamma = 0.9$ and $\alpha = 0.5$.

Learning With No Story Guidance

In this first experiment we examine how RL performs in Robbery World without story guidance. This experiment uses the following reward functions: 1) the agent will get rewarded for reaching a terminal state, and 2) the agent will get rewarded if John leaves the bank with the money.

The optimal sequence of actions for the first reward function involves the police immediately arriving and then arresting John before he gets the money. Assuming perfect correspondence between actions and plot events, this policy is inconsistent because the optimal sequence of actions results in John skipping to the end of the trajectory tree without performing any of the preceding plot events.

The second reward function results in John entering the bank, opening the cash register, taking the money, and then exiting the bank. This policy, however, is also inconsistent with the plot corpus because each story in the corpus involves John interacting with either Sally or the police. Since John did not learn to interact with any of the other characters in the scenario, this policy is inconsistent.

Learning From Stories

In the remaining experiments we examine how Quixote performs under varying assumptions about the quality of action correspondence. The first experiment performed assumes that the system was able to perfectly identify correspondence between environment actions and plot events. This was achieved by manually selecting which environment actions correspond to each plot point. The remaining experiments examine how varying natural language correspondence thresholds affect learned behaviors.

Table 1: Comparison of the size of the plot corpus in terms of number of plots for each threshold and the size of the resulting trajectory tree in terms of nodes in the tree.

Threshold	Corpus Size	Trajectory Tree Size
0.6	2936	6047
0.7	52	117
0.8	3	11

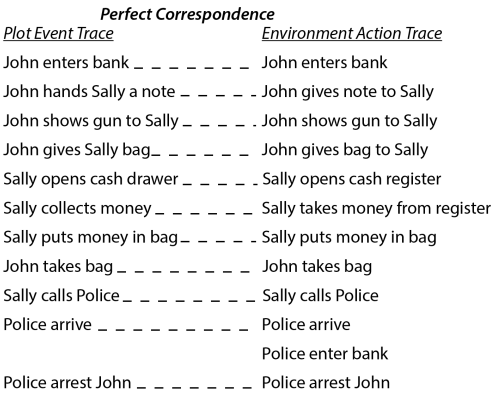

Figure 3: Comparison of the optimal sequence of actions taken and the plot events completed with perfect action correspondence. Dashed lines are drawn between corresponding events and actions.

Constructing an Exemplar Corpus For the following experiments we need to obtain an initial corpus of exemplar stories written in natural language. Here, we use the exemplar corpus collected by Li *et al.* (2013). This corpus was constructed by requesting crowd workers on Amazon Mechanical Turk (AMT) provide stories of how they thought a bank robbery would take place. Crowd workers were instructed to tell the story in natural language, but to limit themselves to simple sentences containing a single verb and no pronouns. Workers were also given two characters to use: John the bank robber and Sally the teller.

In total, this corpus contains 60 stories written in natural language. An example story is given below:

> *John entered the bank. John approached Sally, the bank teller. John showed Sally his gun. John demanded the money. Sally got the money. Sally gave the money to John. John left the bank. Sally called the police. The police arrived. The police caught John. John was arrested. John was convicted of robbery.*

Pre-processing After cleaning, the plot corpus contained approximately $350,000$ plots, the majority of which were small variations of other plots in which the ordering of events is changed. We then determine correspondence between the events in this corpus and the actions available in Robbery World either manually or by using natural language processing with correspondence thresholds of 0.8, 0.7, and 0.6.

Table 1 shows the effect these thresholds had on the number of plots in the resulting plot corpus. As the table shows, higher thresholds resulted in smaller corpora. This is not surprising as the higher thresholds require that a stronger

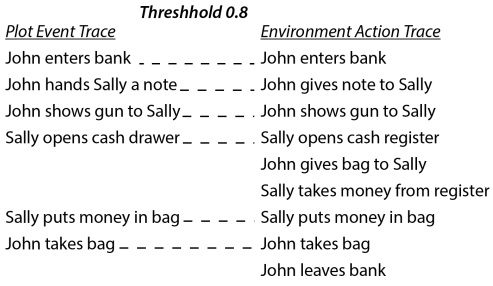

Figure 4: Comparison of the optimal sequence of actions taken and the plot events completed for a threshold of 0.8.

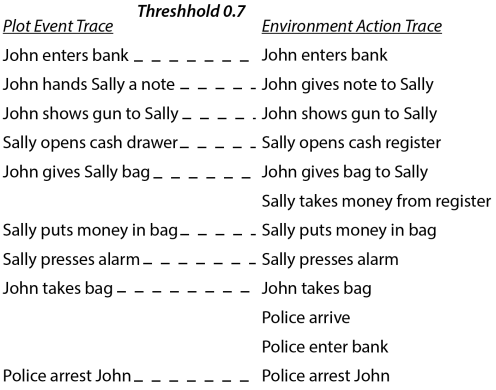

Figure 5: Comparison of the optimal sequence of actions taken and the plot events completed for a threshold of 0.7.

similarity exists between the sentences contained in the plot event clusters and the environment action labels. After determining action correspondence, Quixote constructed the plot corpus and used it to generate the trajectory tree.

Recall that when we integrate the trajectory tree into the world sate, the size of the problem increases with the number of nodes in the trajectory tree. This is especially concerning when one considers the large size of the corpus after initial story cleaning. Examining Table 1, however, shows that the correspondence step has the ability to drastically reduce the size of the resulting plot corpus and, thus, the size of the trajectory tree. While this is not definitive proof, this does provide some evidence that this technique will scale better than initial data cleaning would indicate that in practice, the complexity of the environment has a mitigating effect on scale.

Determining Rewards Using the trajectory tree, Quixote then defines a reward function to use to train a reinforcement learning agent. For Robbery World, we give a base reward value of 2 every time a character performs an action that advances the plot state in the trajectory tree. This base value is weighted by the similarity value between the action being performed and the plot event being completed. Every other action receives a reward value of -1.0. We use these values because they produced acceptable results in practice.

Perfect Correspondence Results The resulting optimal sequence of actions taken by the agent using a reward function generated by Quixote using perfect correspondence is shown in Figure 3. This policy is consistent as the sequence

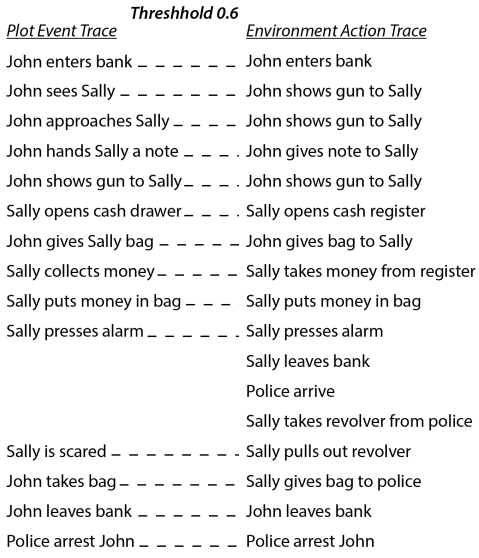

Figure 6: Comparison of the optimal sequence of actions taken and the plot events completed for a threshold of 0.6.

of plot events that the agent completes does exist in the plot corpus used for training. This shows that if Quixote is able to achieve perfect correspondence through either human authoring or perfect natural language, it can teach agents to exhibit behaviors consistent with what it used for teaching.

Varying Thresholds Results While our technique for determining natural language correspondence works reasonably well, though imperfectly, in Robbery World, this experiment aims to identify Quixote's robustness to changes in natural language performance. Lowering thresholds simulates degrading natural language correspondence performance.

Figure 4 shows the path the agent took through the trajectory tree as well as the optimal sequence of actions taken for a correspondence threshold of 0.8. The most important thing to note is that the learned policy for this threshold is consistent. Also notice that the optimal sequence of actions does not involve the police. This is a side-effect of the natural language correspondence process not finding any correspondence between the police's actions and the plot events that contained the police.

Using a correspondence threshold of 0.7 results in a policy that is consistent with the plot corpus (shown in Figure 5).The agent exhibits similar behavior to the agent trained using perfect correspondence (shown in Figure 3). The main difference is that this threshold does not identify correspondence with Sally taking the money from the cash register or the police arriving. Note that the reinforcement learning agent was still able to learn that these action are optimal even though no correspondence was identified earlier.

When the threshold is lowered to 0.6, seen in Figure 6, performance degrades. It is important to note that the optimal sequence of actions taken is technically consistent with the training corpus as per our definition of consistent. The path taken through the tree does correspond to a plot that exists in the training corpus. Inspection of the actions taken in the environment, however, reveals some odd choices. For example, many plot events are rewarded when John shows the Gun to Sally. Although this behavior results in a consistent policy, it would likely be considered unnatural by most people. The problem here is that the threshold identified correspondence between plot events and actions when, in reality, none exists. For example, there is no action that reasonably corresponds to the plot event *John sees Sally*, but the algorithm is forced make a correspondence when one is above the threshold.

Discussion One thing to take from these experiments is that all policies produced by Quixote were consistent with the plot corpora used for training. This provides evidence that Quixote is having an effect on learned behaviors and is biasing learning towards the plots in the plot corpus. These studies also showed, however, the importance of properly identifying correspondence between actions and plot events. The performance of Quixote with a correspondence threshold of 0.6 shows that poor natural language processing led to erratic behavior that, while consistent, would likely appear unnatural to human viewers.

Interactive Narrative

One avenue of future work involves using Quixote for creating interactive simulations in which the human is present or actively controlling one of the characters. This is possible because the policy learned by Quixote dictates the optimal action to take across all characters. The policy learned by Quixote should contain enough information for it to respond to any action that the human player could make. This is especially important for games in which players will interact with NPCs as this allows for the system to respond to the different ways that players could interact with the system.

Reinforcement learning has been successfully used to train virtual characters (Zhao and Szafron 2009) and for drama management before (Nelson and Mateas 2005; Roberts et al. 2006); however complex design knowledge must first be provided that encode the designers intuitions about how characters should behave and scenarios should unfold. The work presented in this paper allows the same knowledge to be automatically acquired from natural language interactions (in this case, from crowdsourced narrative examples) and converted into a form suitable to train virtual characters: a reward function.

Conclusions

In this work we introduce Quixote, a system for training believable agents based on stories. Our system works by using exemplar stories to define a space of acceptable behavior and then using this space to derive reward functions that encourage the agent to exhibit behaviors that fall within it. We have presented a case study in which crowdsourced stories where used to generate behaviors that were consistent with the behaviors contained within these training stories. Using this technique, we allow humans to more naturally communicate with interactive machine learning algorithms which will, ideally, make it easier for non-programmers to use these algorithms.

Acknowledegments

This material is based upon work supported by the U.S. Defense Advanced Research Projects Agency (DARPA) under Grant #D11AP00270 and the Office of Naval Research (ONR) under Grant #N00014-14-1-0003.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 1. ACM.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

Grollman, D. H., and Billard, A. 2011. Donut as i do: Learning from failed demonstrations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 3804–3809. IEEE.

Judah, K.; Roy, S.; Fern, A.; and Dietterich, T. G. 2010. Reinforcement learning via practice and critique advice. In *AAAI*.

Knox, W. B., and Stone, P. 2009. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, 9–16. ACM.

Li, B.; Lee-Urban, S.; Johnston, G.; and Riedl, M. 2013. Story generation with crowdsourced plot graphs. In *AAAI*.

Lignos, C.; Raman, V.; Finucane, C.; Marcus, M.; and Kress-Gazit, H. 2015. Provably correct reactive control from natural language. *Autonomous Robots* 38(1):89–105.

Loftin, R.; MacGlashan, J.; Peng, B.; Taylor, M. E.; Littman, M. L.; Huang, J.; and Roberts, D. L. 2014. A strategy-aware technique for learning behaviors from discrete human feedback. In *Proc. of AAAI*.

MacGlashan, J.; Babes-Vroman, M.; desJardins, M.; Littman, M.; Muresan, S.; Squire, S.; Tellex, S.; Arumugam, D.; and Yang, L. 2015. Grounding english commands to reward functions. In *Proceedings of Robotics: Science and Systems*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Nelson, M. J., and Mateas, M. 2005. Search-based drama management in the interactive fiction anchorhead. In *AIIDE*, 99–104.

Ng, A. Y., and Russell, S. 2000. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*.

Ramachandran, D., and Amir, E. 2007. Bayesian inverse reinforcement learning. In *Proceedings of the 20th international joint conference on Artifical intelligence*, 2586–2591. Morgan Kaufmann Publishers Inc.

Roberts, D. L.; Nelson, M. J.; Isbell, C. L.; Mateas, M.; and Littman, M. L. 2006. Targeting specific distributions of trajectories in mdps. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, 1213. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

Zhao, R., and Szafron, D. 2009. Learning character behaviors using agent modeling in games. In *AIIDE*.

Ziebart, B. D.; Maas, A.; Bagnell, J. A.; and Dey, A. K. 2008. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd national conference on Artificial intelligence-Volume 3*, 1433–1438. AAAI Press.