The Workshops of the Thirty-Second
AAAI Conference on Artificial Intelligence

Towards Explainable NPCs: A Relational Exploration Learning Agent

Matthew Molineaux,[1] **Dustin Dannenhauer,**[2] **David W. Aha**[3]
[1]Knexus Research Corporation; Springfield, VA
[2]NRC Postdoctoral Fellow; NRL; Navy Center for Applied Research in AI; Washington, DC
[3]Naval Research Laboratory; Navy Center for Applied Research in AI; Washington DC
matthew.molineaux@knexusresearch.com, {dustin.dannenhauer.ctr, david.aha}@nrl.navy.mil

Abstract

Non-player characters (NPCs) in video games are a common form of frustration for players because they generally provide no explanations for their actions or provide simplistic explanations using fixed scripts. Motivated by this, we consider a new design for agents that can learn about their environments, accomplish a range of goals, and explain what they are doing to a supervisor. We propose a framework for studying this type of agent, and compare it to existing reinforcement learning and self-motivated agent frameworks. We propose a novel design for an initial agent that acts within this framework. Finally, we describe an evaluation centered around the supervisor's satisfaction and understanding of the agent's behavior.

Introduction

Assistant non-player characters (NPCs) can be a major source of frustration to video game players. Commonly, players can give tasks or policy instructions to an NPC, who use human-written behaviors to act within the rules of the game world to assist the player and accomplish tasks. Frustration stems primarily from the opacity of their reasoning processes. They may be responding to internal needs the player is unaware of, or encountering obstacles the player cannot see, but they do not communicate these problems. As a result, a player can assume incompetence and not take advantage of the NPC's skills. Furthermore, such NPCs are reliant on pre-existing hand-coded behaviors; they can't respond to novel requests, nor adapt to changes in their environment. Scripted NPCs with fixed models of the world can be reasonably used in games where the NPC can be expected to take a constrained role in well-defined environments, but as games increase in complexity to include growing or changing worlds (e.g., user created content or computer-generated levels) the need for an adaptive agent increases. We focus on NPCs in the player-assistant role and posit that an explainable agent who explores when not tasked would be more useful and less frustrating.

For NPCs to explain themselves to human players, it's important that they represent concepts important to the player: *objects* and *relationships* in the game world, and *goals* and *plans* that the user understands. These capabilities are common to many planning and execution agents; therefore, we seek to extend such agents to perform exploration-based learning, and interact with a human supervisor. In addition to acquiring knowledge about the state of the environment, we are interested in allowing the agent to update a model of the environment's transitions based on self-directed exploration. This will allow an agent to improve its understanding of how the environment and other agents develop. To facilitate goal-oriented behavior, we consider representations for this transition model drawn from automated planning research.

This paper is composed as follows: in Section 2, we describe a framework for explainable autonomous agents that can be used to describe exploratory and goal-directed behaviors as well as interaction with a supervisor. In Section 3, we describe how this framework relates to prior research in reinforcement learning and self-motivated learning. Section 4 presents the design of an initial explainable exploratory goal-driven agent. Section 5 discusses a notional study with such an agent. Section 6 reviews plans for future work.

Framework for Explainable Autonomous Agents

Existing frameworks such as the reinforcement learning paradigm are inadequate to describe the information needs of explainable autonomous agents. Instead, we present a new framework (Figure 1), based on the idea that a supervisor, rather than the environment, provides an agent with motivation. As in other agent frameworks, the agent receives an **observation** (o_t) on each time step t that reflects information about the true environment state (s_t, not depicted), and interacts with the environment by taking an **action** (a_t). Unlike in other frameworks, the agent also interacts with a **supervisor**. The supervisor makes **requests** (r_t) of the agent, updated at each time step, that reflect what the supervisor would like the agent to accomplish. In return, the agent is expected to provide an **explanation** (x_t) to the supervisor at each time step describing why it takes a particular action.

Supervisors will observe the world and be more or less satisfied with how the agent is doing; we represent this with a satisfaction function, *satisfaction*: $R^N \times S^N \rightarrow \mathbb{R}$, that indicates how a supervisor's past requests and the environment's state impact how satisfied the supervisor becomes. The satisfaction function can encode, for example, that the supervisor's satisfaction decreases based on the latency be-

Figure 1: Explainable Autonomous Agent Framework

tween their first request and the time step when the agent actually accomplishes a goal. Other complex constraints and preferences can also be encoded.

We represent the environment as producing observations deterministically based on states via an observation function $obs : S \rightarrow O$, and transitioning deterministically (in future work we will explore non-deterministic transitions) between states via the transition function $\lambda : S \times A \rightarrow S$.

This framework allows for explanations and requests to take many different forms; the user could specify a request as a goal state, a temporal expression, or as a set of preferences and constraints. In Section 4, we discuss some choices we make about the design of a particular autonomous agent.

Related Work

Work on learning action models has a long history in the area of reinforcement learning. The primary difference of this work from RL is that our agent is goal-driven (notice in Figure 1 that the environment only provides an observation and not a reward). Yet since we also have an exploration vs exploitation problem, we expect to draw on ideas from RL.

Carmel and Markovitch (1999) contrast reinforcement learning with model-based learning in order to show how RL strategies can be transferred. Their work focuses on adversarial domains such as the Iterate Prisoners' Dilemma. They learn models of their opponents as finite automata using a utility. A primary contribution is a novel lookahead based exploration strategy to attempt to avoid the dangers implicit in exploration through a Bayesian prior, which helps the system avoid low-utility absorbing paths in an opponent model. In our work we do not focus exclusively on adversarial domains given that we also model a state of the environment. Avoiding dangers is something we consider for our exploratory planner; during exploration it should avoid states that are undesirable even if the agent would gain a novel update to its action model.

Georgeon, Morgan, and Ritter (2010) discuss an algorithm for an "intrinsically motivated" agent, similar to a reinforcement learning agent, whose "rewards" (referred to as "satisfaction") are based on actions and their effects, but not the world state. Work by Georgeon et al. differs from RL because the focus is on trying to learn hierarchies that achieve a good reward rather than taking the best action at a particular state. The paper does not describe clearly whether there is an existing transition model within the schemas. It is clear that observations are different from RL observations, as they assume a non-Markovian environment. The major difference from this work is the lack of state information. Here we are concerned with a supervisor that is able to give goals in the form of future states for the agent to achieve, and that the agent is able to learn a state transition function for its actions.

Hester and Stone (2017) discuss multiple exploration reward schemes and reward metrics in intrinsically motivated RL systems. Two exploration-based intrinsic reward schemes are presented: a variance-based scheme and a novelty-based scheme. The variance-based scheme rewards an agent for taking an action a in a state s proportionally to the disagreement among its learned team of experts about the outcome distribution of the next state $P(s, a)$; that is to say, the agent becomes more likely to take an action if it is uncertain of that action's outcome. The novelty-based scheme rewards an agent for taking an action a proportionally to the distance between the current state s and the nearest state s' in which action a was previously taken. That is to say, an action is preferred to the extent that it has not yet been tried in a similar state. Their work focuses only on the exploration period and do not consider the exploration-exploitation tradeoff, which we explicitly account for in our agent design (within the Controller shown in Figure 2). We propose that our agent will often need to choose whether to explore to learn its action model or to achieve a goal. The novelty-based scheme is directly applicable to the exploration planner; the notion of novelty could serve as a heuristic to find a state in which taking an action would yield considerable new observations.

Baranes and Oudeyer (2009) discuss learning of forward and inverse robot kinematic and dynamic models for robot control. Exploration is governed in three ways: *uniform random*, *learning process maximization exploration*, and *error maximization learning*. *Learning process maximization exploration* selects a (continuous) action from the region where the best learning progress is currently being made (referred to by Hester and Stone as "competence progress"). *Error maximization learning* (which is based on *learning process maximization exploration*) selects an action with the maximum predicted error within the region where the best learning progress is currently being made. No exploitation is performed in that work; tradeoff between various exploration methods is achieved with simple constant probabilities.

Sequeira, Melo, and Paiva (2011) discuss emotions as a structure for intrinsic motivations which are balanced against extrinsic motivations. Numeric intrinisic motivations are based on appraisal dimensions including novelty, motivation, control, and valence. Novelty is proportional to the number of times an action has been used in a given situation before. Motivation has to do with relevance, and is inversely proportional to the (estimated) distance to a "goal" state that provides maximal reward. Control is simply the inverse of novelty, as the agent has more control when it's in a more well understood situation. Valence seems to deal with how well the agent's basic needs are currently met.

Since we are interested in learning actions in a relational structure, we describe related work for learning relational action models in automated planning research. For example, actions may have preconditions and effects. This would be more closely related to model-based RL where an agent can predict what state it will be in if it takes a specific action in

a specific state (as opposed to just knowing the reward value for taking an action in a state, like in Q-learning). However, most RL problems are formulated using a feature-vector of the state while we are considering a more ontological structure of the state that has objects and predicates specifying relations among the objects.

(Haussler 1989) shows that to find the maximally specific common generalization MSCG of preconditions (or effects) for an operator o given m examples over an instance space defined by n attributes is NP-complete. Inductive approaches that use heuristics have been effective (Vere 1980; Hayes-Roth and McDermott 1978; Watanabe and Rendell 1990) but may not apply to all learning problems. FOIL (Quinlan 1990) is a greedy algorithm that, like the heuristic approaches just mentioned, requires both positive and negative examples for training. (Wang 1995) developed a system called OBSERVER that uses expert traces of actions and a simulator for running practice problems. OBSERVER does not need approximate action models, it learns action models from scratch that are as effective as human-expert coded action models assuming expert traces are available. One requirement of OBSERVER is the ability to learn with positive-only examples prior to execution. In our work we are considering an online approach, where the agent will generate negative samples whenever an action fails. Thus we plan to use an approach similar to OBSERVER and FOIL for deterministic, fully observable environments.

A survey on machine learning for automated planning (Jiménez et al. 2012) noted that, for learning deterministic models in fully observable environments, collecting a good observation sample of planning actions remains an open problem. (Walsh and Littman 2008) show that by bounding the number of preconditions and effects for a given action, the number of executions needed to guarantee learning the action is reduced to polynomial complexity (polynomial in the number of actions and predicates in the domain).

Besides only focusing on operator learning for planning with complete action models, (Weber and Bryce 2011) gives a planning algorithm that reasons about the domain model's incompleteness in order to avoid failure. Incorporating such a technique into our agent for the goal-driven planner would allow the agent to perform planning prior to the learning of a complete action model (which may never happen).

Finally, learning models for actions alone may not be enough in environments with exogenous events. (Molineaux and Aha 2014) learn models of unknown exogenous events in their FoolMeTwice system using DiscoverHistory. DiscoverHistory is an algorithm for discovering explanations given a series of partial observations, and by generalizing over these explanations using an adaptation of FOIL (Quinlan 1990), a model of the event can be formulated. Incorporating events into the planner produce better plans that are able to account for the newly discovered events. Our exploratory planner will need to be able to direct behavior in order to learn models of not only actions but also exogenous events, should they be present in the environment.

Explainable Exploratory Goal-Driven Agent

Figure 2 shows the architecture of an initial explainable autonomous agent capable of taking actions in order to explore as well as to achieve goals. This agent assumes a relational representation of states, observations, and actions are provided to it, as well as the environment's observation function, but not the transition function. The agent attempts to learn the transition function over time. The supervisor is assumed to represent user requests via a set of goals, and explanations via the agent's current plan and the goal that plan is intended to achieve; as a common representation has been provided to the agent, we expect that these plans and goals will be understandable to the supervisor.

We separate the functionality of this agent into four submodules: the **exploratory planner**, responsible for taking actions to obtain new information with which to update the action model, the **goal-directed planner**, responsible for achieving goals given by the supervisor, the **transition model learner**, responsible for updating the agent's model of the world, and the **controller**, responsible for determining when to explore, achieve goals, and update the model, as well as communicating with the supervisor. We separate exploration planning from goal achievement planning due to typical goal-based planners seek states, not novelty, and therefore standard heuristics and other optimizations are likely to not be useful for exploration. Each submodule is intended to work in a domain-independent fashion, with data represented in a relational fashion. We now describe the current design of each submodule.

Exploratory Planner: Given an initial state s, an interaction history $(\langle s_0, a_0, s_1\rangle, \langle s_1, a_1, s_2\rangle, \ldots)$, and a domain model of the transition function λ , the exploratory planner is expected to find a plan π that seeks novel observations of the environment. In prior work by, e.g., (Sequeira, Melo, and Paiva 2011; Hester and Stone 2017), novelty is recognized as an important intrinsic motivation that rewards exploration. In each work, novelty encourages taking actions unlike other actions taken in the same or similar states. We would like to build on this notion to encourage exploration in a relational context. As relational environments can have a very large number of states, we consider the novelty of taking an action with respect to individual literals; we will build a map tracking how many times each ground action has been taken in the presence of each ground literal that shares parameters with it. All literals that share parameters with an action will vote based on how often the action was taken in the presence of that literal. In this manner, we will determine actions with high novelty. We will also use the novelty of a literal-action pair, combined with the cost of reaching a state with that literal, to encourage **exploratory plans** that achieve high novelty levels.

Goal Achievement Planner: This module will be provided by a traditional automated planner: given an initial state s, a goal g and a domain model of the transition function λ, it provides a plan π expected to lead to a state that satisfies g (i.e., $s \models g$). We will initially consider Metric-FF (Hoffmann 2003), for its ability to plan based on action knowledge containing metrics.

Controller: At each time step, the controller receives a state

s_t from the environment and a request from the supervisor in the form of a set of goals $[g]_t$; it generates a new action a_t to act in the environment and explains that action to the supervisor via a current goal g_t and a plan to achieve it π_t. It calls upon the exploratory and goal planners to create the plans, but must determine the goal itself. This requires the controller to manage the exploration-exploitation tradeoff, as well as choose a goal that best satisfies the supervisor's current request. When taking exploratory actions, the controller's explanation consists of the symbol $exploration$ in lieu of a goal, and its exploratory plan.

The exploration-exploitation tradeoff is characterized as follows: In the initial stages of acting in an environment, an agent can not be expected to have sufficient knowledge to exploit; furthermore, in some cases, the supervisor's request may be an empty set of goals. In these cases, exploration is an obvious choice. Similarly, if the agent has repeatedly fulfilled a particular goal successfully, exploration is unlikely to be necessary before fulfilling that goal in a request. In this situation, goal achievement is the obvious choice. To satisfy these needs, the controller will simply choose to pursue a supervisor's goal *as long as a plan can be constructed to do so*. At other times, exploratory planning will be performed.

When multiple supervisor goals are achievable, the controller will use domain independent motivation-based goal selection techniques to choose among them, as described in work by Wilson et al 2013. In this strategy, an **urgency value** between 0 and 1 is calculated for each motivation (including social, exploratory, and opportunity) at each time step. Then, each goal available is assigned a **fitness value** for each motivation. A goal is selected that maximizes urgency-weighted fitness.

Transition Model Learner: Given a history of interactions $(\langle s_0, a_0, s_1\rangle, \langle s_1, a_1, s_2\rangle, \ldots)$, the transition model learner must find a model of the transition function λ that minimizes error over the interaction history. While the agent is learning the effects of actions, a human is watching and may ask the agent to explain itself. One rationale behind learning an action model is to reduce the knowledge engineering behavior of the system designer. It may also be true that the designer could make a mistake in the action model, which the agent could overcome by learning the correct model.

Each time the agent executes an action, we save the pre and post states as a case corresponding to that action. In this way, we can begin to estimate what the preconditions and effects of actions are over time. Every time an action is executed and a new case is added, if the case is not identical to a previous case, we re-compute the preconditions and effects of that action. In our initial studies we will use approaches similar to OBSERVER (Wang 1995) and FOIL (Quinlan 1990).

We expect that future versions of this agent will relax assumptions including that of a deterministic, fully observable environment, and initial knowledge of the supervisor's satisfaction function. We hope to extend to more complex explanation and request representations. We expect that the goal-directed planning will become more difficult given a partially observable domain and probabilistic actions (see Figure 2 in Jimeńez et al. (2012) for a summary of models under

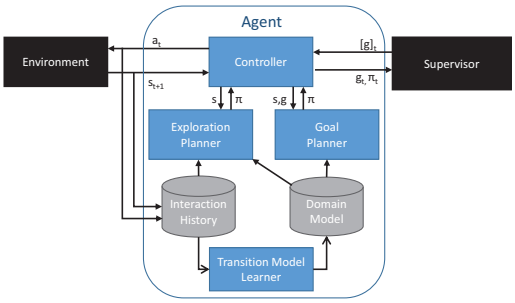

Figure 2: Architecture for Exploratory and Goal-Driven Autonomous Agent

different domain conditions). Our design represents only an initial agent; there is a great deal of room for expansion.

Evaluating Explainable Autonomous Agents

We are interested in evaluating agents in complex game worlds, to provide challenge in terms of exploration, diverse goals, and plan complexity. In initial experiments, we expect to model the supervisor using an oracle with a constant supervision function that deterministically introduces new goals at predetermined times or based on a state of the environment.

Appropriate metrics for evaluation of these agents revolve around the supervisor's satisfaction and understanding. We would like to evaluate the change in the supervisor's satisfaction over time, as more exploration allows an agent to better satisfy goals more quickly. The degree to which a human supervisor understands why an agent is failing to achieve a goal is also important; fundamentally, we are interested in explanations to make the interaction with the agent less frustrating, and we expect this frustration to decrease as the human understands the agent better. This can be measured subjectively through direct questions to a group of human players, but also (possibly) objectively through the requests the supervisor makes. Under the hypothesis that a supervisor who understands an agent will change their request behavior to suit the agent's capabilities, the quality of explanations can be measured based on the percentage of user requests that are actually achievable for the agent.

Discussion

Video games provide a great testbed, as well as application domain, for research on explainable autonomous agents. The opportunities for exploration and challenges for developing and communicating plans provide for strong experimentation. Research on explainable autonomous agents provides new challenges not encountered by reinforcement learning agents, planning and execution agents, or self-motivated agents; however, research from each is applicable. The explainable autonomous agent framework provides lots of room for growth; we have described a novel agent designed for initial research, but expect to grow far beyond that. Many research problems must be solved to create competent explainable autonomous agents.

Acknowledgments

This research was developed with funding from DARPA. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the DoD or the U.S. Government.

References

Baranes, A., and Oudeyer, P.-Y. 2009. R-IAC: Robust intrinsically motivated exploration and active learning. *IEEE Transactions on Autonomous Mental Development* 1(3):155–169.

Carmel, D., and Markovitch, S. 1999. Exploration strategies for model-based learning in multi-agent systems: Exploration strategies. *Autonomous Agents and Multi-agent systems* 2(2):141–172.

Georgeon, O. L.; Morgan, J. H.; and Ritter, F. E. 2010. An Algorithm for Self-Motivated Hierarchical Sequence Learning.

Haussler, D. 1989. Learning Conjunctive Concepts in Structural Domains. *Machine Learning* 4:7–40.

Hayes-Roth, F., and McDermott, J. 1978. An interference matching technique for inducing abstractions. *Communications of the ACM* 21(5):401–411.

Hester, T., and Stone, P. 2017. Intrinsically motivated model learning for developing curious robots. *Artificial Intelligence* 247:170–186.

Hoffmann, J. 2003. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(04):433–467.

Molineaux, M., and Aha, D. W. 2014. Learning unknown event models. In *AAAI*, 395–401.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine learning* 5(3):239–266.

Sequeira, P.; Melo, F. S.; and Paiva, A. 2011. Emotion-based intrinsic motivation for reinforcement learning agents. In *International Conference on Affective Computing and Intelligent Interaction*, 326–336. Springer.

Vere, S. A. 1980. Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial intelligence* 14(2):139–164.

Walsh, T., and Littman, M. 2008. Efficient Learning of Action Schemas and Web-Service Descriptions. *Aaai* 714–719.

Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. *Icml*.

Watanabe, L., and Rendell, L. A. 1990. Effective generalization of relational descriptions. In *AAAI*, 875–881.

Weber, C., and Bryce, D. 2011. Planning and Acting in Incomplete Domains. 274–281.

Wilson, M. A.; Molineaux, M.; and Aha, D. W. 2013. Domain-independent heuristics for goal formulation. In *FLAIRS Conference*.