Learning from Exploration: Towards an Explainable Goal Reasoning Agent

Dustin Dannenhauer
NRC Postdoctoral Fellow
Naval Research Laboratory, Washington, D.C.
dustin.dannenhauer.ctr@nrl.navy.mil

Michael W. Floyd
Knexus Research Corporation
Springfield, VA
michael.floyd@knexusresearch.com

Matthew Molineaux
Wright State Research Institute
Dayton, OH
matthew.molineaux@wright.edu

David W. Aha
Navy Center for Applied Research in AI
Naval Research Laboratory, Washington, D.C.
david.aha@nrl.navy.mil

ABSTRACT

Complex, real-world environments may not be fully modeled for an agent, especially if the agent has never operated in the environment before. The agent's ability to effectively plan and act in the environment is influenced by its knowledge of when it can perform specific actions and the effects of those actions. We describe progress on an explainable exploratory planning agent that is capable of learning action preconditions using an exploratory planning process. The agent's architecture allows it to perform both exploratory actions as well as goal-directed actions, which opens up important considerations for how exploratory planning and goal planning should be controlled, as well as how the agent's behavior should be explained to any teammates it may have. We describe initial approaches for both exploratory planning and action model learning, and evaluate them in the video game Dungeon Crawl Stone Soup.

1 INTRODUCTION

Sufficiently complex environments, like those that may be encountered in the real world, are rarely fully defined and modeled in advance. Even minor changes in the environment, like weather conditions, can significantly impact how an agent is able to act. For example, the action preconditions for steering a vehicle in snow differ from the preconditions when the weather is clear. It is unrealistic to assume that a knowledge engineer would be able to provide the agent with information related to all possible environment variants. Instead, it would be advantageous for the agent to be able to dynamically learn how novel environments impact its ability to act and update its action model accordingly.

We present early work towards an explainable planning agent capable of operating in new environments, even if those environments differ noticeably from previously encountered domains. We do not assume the agent will have access to an expert-provided domain model or plan traces. Instead, we consider agents that perform exploration to obtain a collection of action interactions and uses those interactions for learning relational action models (i.e., preconditions and effects). The agents are not expected to be purely exploratory, but instead be members of teams where other teammates may request the agents to pursue provided goals. Given that, at any time, the agents may be performing a mixture of exploration (i.e., attempting actions), learning (i.e., learning action models), and goal pursuit (i.e., exploiting a learned model to achieve teammate-provided goals), explanation of the agents' behavior is important to ensure teammates understand and trust the agent. Our work has four primary contributions:

- A baseline exploratory planning approach that attempts to equally distribute the agent's actions in varying contexts.
- A method for learning action preconditions from the agent's own environment interactions.
- An implementation of both exploratory planning and action model learning as part of an explanatory goal reasoning agent.
- A preliminary evaluation in a video game setting.

The remainder of this paper describes our explainable exploratory planning agent and presents our current progress on implementing various capabilities in the agent. Section 2 describes our overall agent architecture and highlights the specific components we focus on in this paper, namely exploratory planning, action model learning, and explanation. Section 3 presents our exploration procedure, with Section 4 describing our technique for action model learning. We describe an initial evaluation of a subset of the agent's capabilities in the open-source video game Dungeon Crawl Stone Soup (DCSS). Section 5 discusses the explanation capabilities we anticipate will be necessary for the agent. Related work in the areas of action model learning, explainable planning, and explanations when reasoning over goals is discussed in Section 6. Section 7 describes our planned future work, with concluding remarks in Section 8.

2 EXPLORATORY PLANNING AGENT

We adopt the planning formalism from [2] with a state transition system $\Sigma = (S, A, E, \gamma)$ where S is the set of all states, A is the set of actions, E is the set of events, and γ is the state-transition function. In this paper, we build off of our architecture for an exploratory goal reasoning agent [8], shown in Figure 1. Although further details about the architecture are provided in our previous work, we focus our discussion on the *Exploration Planner* and *Transition Model Learner* components.

The role of the Exploration Planner is to obtain novel *interactions*. An interaction is defined as a triple $\langle s_i, a_i, s_{i+1} \rangle$ where $s_i \in S$ is the previous state before taking action $a_i \in A$ and $s_{i+1} \in S$ is the state after a_i is executed. As the agent acts in the environment, either in

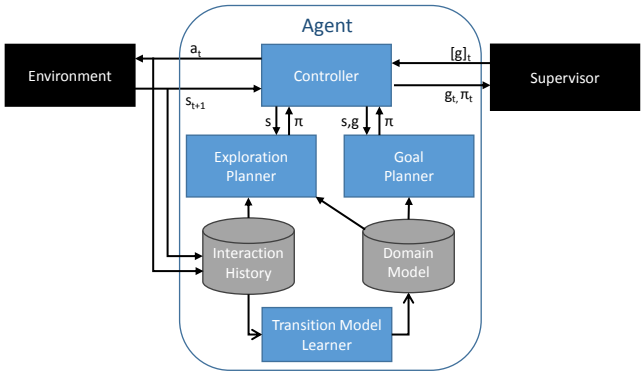


Figure 1: Architecture for our exploratory goal reasoning agent

an attempt to achieve a goal, as guided by the Goal Planner, or in an attempt to gain example interactions, as guided by the Exploration Planner, all interactions are recorded and added to the Interaction History I. We collect complete states from the environment, in this work the DCSS simulator, after each action is executed. The Interaction History provides training examples that can later be used by the Transition Model Learner to learn a Domain Model. Thus, by guiding the agent's actions, the Exploration Planner can impact which interactions are stored in the Interaction History and used for learning. The Exploration Planner is described in more detail in Section 3.

The Transition Model Learner uses an inductive learning procedure to learn the preconditions of each of the agent's actions (i.e., update the Domain Model with the actions' preconditions). An example of the learned preconditions for the *southeast* move action from the DCSS domain is shown in Figure 3. For more detail, see Section 4. While this work focuses on updating the Domain Model with the preconditions of actions, we also plan to learn action effects using a case-based approach that examines the set difference between the previous state s_i and the next state s_{i+1}. Since some actions are non-deterministic, there may be multiple possible effects for the same set of preconditions. In DCSS, the same keypress from a user is used for both movement (i.e., when the adjacent square is empty) and simple melee combat (i.e., when an enemy is in the destination square). When there is an enemy in an adjacent square, the attack may either *hit* or *miss*, and a hit causes a random amount of damage drawn from a state-dependent distribution.

The Goal Planner can use any automated planner, although in our work we currently use the Metric-FF planner [6] due to its ability to handle numeric values. Finally, the Controller is responsible for goal reasoning (i.e., reasoning over possible goals and selecting a subset to pursue) and explanation (e.g., explaining its behavior to teammates, such as the Supervisor).

3 EXPLORATION PLANNER

Our Exploration Planner uses *contexts* to guide the exploration process. A context c is a first-order relational conjunct $c_1 \wedge c_2 \wedge \ldots \wedge c_n$ that refers to one or more existentially quantified variables and is satisfied by some states $S' \subset S$. We describe a context c having n terms as being of size n (e.g. Figure 2 shows contexts of size $n = 3, 4, 5$). Context terms may include the following: finite relations over objects, finite equalities and inequalities between integers and functions over objects, and the infinite "successor" relation that describes consecutive integers. An upper bound[1] C on the number of possible contexts for a given size n is given by

$$C = \binom{M! \times 2 \times |P|}{n},$$

where M is the maximum number of arguments for a single predicate $p \in P$. In a context with multiple terms, each term must refer to at least one existentially quantified variable that is also present in another term; this requirement means that no context of size n is semantically equivalent to the conjunction of two or more smaller contexts of size $< n$. For example, in the context of Figure 2a, T1 is such an existentially quantified variable. We describe the set of all legal contexts for an environment Σ up to a size n as $contexts(\Sigma, n)$.

During execution, a context is considered *active* if it is satisfied by the current state s. The agent maintains a count of the number of times each context action-pair $\langle c, a \rangle$ has occurred in the Interaction History. This is given by the function:

$$examples(c, a) = \left| \{ \langle s_i, a_i, s_{i+1} \rangle \in I \mid c \models s_i \wedge a_i = a \} \right|$$

To determine the next action, the Exploration Planner chooses the action a that minimizes $examples(c, a)$ for all active legal contexts of a size less than or equal to n:

$$\arg\min_{a \in A} \min_{c \in contexts(\Sigma, n)} examples(c, a)$$

The intuition behind using contexts for exploration is that the same action needs to be executed in multiple contexts in order to learn the complete precondition rule, and also effects, for that action. Consider the preconditions for *southeast* shown in Figure 3. In order for the agent to learn the condition *not wall(V0)*, it needs to execute the action in a context where there is a wall southeast of the agent, thereby preventing the action from succeeding. This produces a negative interaction. To learn positive interactions, the agent must execute the action in contexts where there is not a wall in a cell to the southeast. These positive interactions enable learning the other preconditions besides *not wall(V0)*.

In future work, we will explore the use of goal achievement planning to explore regions of the state space that are underrepresented by prior interactions. We hypothesize that this will help in seeking rare contexts that may be necessary to witness interesting effects from certain actions (for example, a pickup action will not have interesting effects unless there is an object on the ground in the same tile as the agent). Seeking such contexts requires either planning for existentially quantified goals or attempting to plan for multiple grounded versions of a goal which may or may not be attainable. We expect to encounter issues related to planning with incomplete domain models [16] since the action model may first

[1]This upper bound assumes all predicates $p \in P$ have a number of arguments equal to the maximum number of arguments M, however usually some p have less than M arguments.

```
{agent-at(T1), at(T1,X1,Y1),
at(T2,X2,Y2)}
```

(a) Context of size 3

```
{agent-at(T1), at(T1,X1,Y1),
at(T2,X2,Y2), not wall(T2)}
```

(b) Context of size 4

```
{agent-at(T1), at(T1,X1,Y1),
at(T2,X2,Y2), not wall(T2),
X2 = X1-1}
```

(c) Context of size 5

Figure 2: Example contexts of varying sizes

```
southeast(V3) :-
   at(V0,V1,V2), at(V3,V4,V5),
   not wall(V0), V4 = V1-1,
   V5 = V2-1, agent-at(V3).
```

Figure 3: Learned rule from ILASP for the southeast move action in the Dungeon Crawl Stone Soup domain

require interactions in the desired context. However, this must entail learning; in deterministic environments, failure to attain a goal that is reachable given the current model necessarily involves interactions that contradict the current model. Such examples should serve to improve the model.

4 TRANSITION MODEL LEARNING

We are interested in agents that can learn their action model online while operating in a new environment. For our Transition Model Learner component, we assume that action predicates and argument types are known to the agent a priori (i.e., provided as expert domain knowledge) as well as a model of the environment that includes objects, types of these objects, and predicates for these objects. Currently we assume the world is fully observable and deterministic; we will relax these constraints in future work.

As we discussed previously, every action the agent takes is recorded into a triple $\langle s_i, a_i, s_{i+1} \rangle$ and stored in the Interaction History, which can be used later for learning. We have taken an inductive learning approach, inspired by prior work on learning action models (discussed later in Section 6). The primary difference in our work is that the agent is not provided with expert traces of action sequences and instead directs its own behavior to acquire examples of executing actions.

For the inductive learning algorithm, we use the Inductive Learning of Answer Set Programs (ILASP) system [7] to learn rules that act as preconditions of individual actions. We assume that if an action did not change the state of the world, then it failed[2], and use that assumption to label interactions as *positive interactions* that succeeded ($s_i \neq s_{i+1}$ after performing a_i) or *negative interactions* that failed ($s_i = s_{i+1}$ after performing a_i). Along with the set of positive and negative interactions stored in the Interaction History, the facts in the current state (except for the player's location, since this is often changing) are given as background knowledge to ILASP. Additionally, ILASP is given inductive biases describing the types of predicates that should be considered in the body (in the form of **#modeb** declarations), which are translated from the predicates and arguments types from our domain model of DCSS.

An example for a precondition rule generated by the ILASP system is shown in Figure 3. This rule encodes the learned preconditions for the *southeast* move action for the Dungeon Crawl Stone Soup domain. The various values used in the rule are:

- *V3*: The cell the player is currently located in - serving as the action's argument.
- *V0*: Another cell that is southeast of V3.
- *V1*: The x-coordinate of V0, an integer value.
- *V2*: The y-coordinate of V0, an integer value.
- *V4*: The x-coordinate of V3, an integer value.
- *V5*: The y-coordinate of V3, an integer value.

In the domain, x-coordinates increase from west to east, and y-coordinates increase from north to south. The rule that was learned is that to move southeast from *V3*, the agent must be at *V3*, and there must be another cell *V0* that has an x-coordinate that is one square east ($V4 = V1 - 1$) and one square south ($V5 = V2 - 1$). Additionally, there must not be a wall at *V0*.

In our current setup, we support learning the eight cardinal directions and the only change in the environment that occurs from executing actions is the agent's location. This allows us to use Version 2 of ILASP[3] and encode the current state as background knowledge except for the player's location. When we add actions that may change other parts of the state (such as picking up or dropping objects), we will use ILASP's contexts[4] feature which allows individual positive or negative examples to be associated with their own background knowledge. This allows for learning rules where some examples are true in some situations and others true in other situations, thus relaxing the requirement of having a single global set of facts (background knowledge) which all rules must be associated with. For full details of ILASP, see [7].

After every new interaction is recorded, the agent makes an external call to the ILASP system to perform learning. Each external call to ILASP takes less then 0.75 seconds, however we expect that learning will only need to be done occasionally (e.g., after a batch of new interactions has been collected), so even increased learning time (which may occur when using ILASP contexts or noisy examples) will be tolerable.

4.1 Evaluation

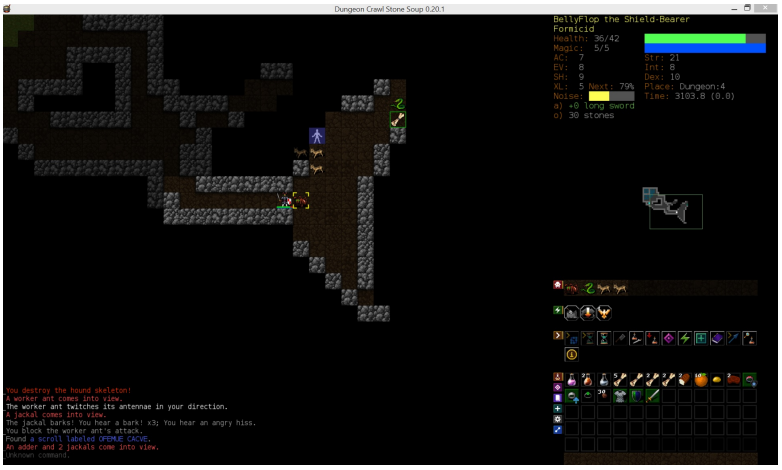
Our evaluation aims to provide some preliminary evidence that our Transition Model Learner can learn action preconditions when situated in a previously unseen domain, and that our initial implementation of the Exploration Planner can improve the agent's ability to learn action models. We hypothesize that:

H1 The agent will be able to learn action preconditions from collected interactions.

[2]We will relax this assumption for partially observable and dynamic environments for future work.

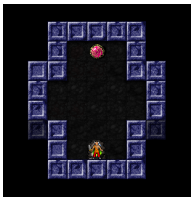
[3]ILASP supports inductive learning under various conditions, including a small number of examples (Version 2), an iterative version that scales well with the number of examples (Version 2i), and learning with large numbers of noisy examples (Version 3).

[4]This is not related to our use of the term *contexts* in this paper.



(a) Screenshot of the full DCSS Game in Progress

(b) Open-room Custom Scenario



(c) Small Custom Scenario

Figure 4: Screenshots of Dungeon Crawl Stone Soup

H2 The agent will demonstrate improved learning when using the Exploration Planner compared to random exploration.

H3 Exploratory planning with larger context sizes will allow for improved learning using fewer actions.

H1 relates to our claim that our action learning approach is beneficial. The rational behind **H2** is that because the exploratory planning agent maintains information about which actions have been taken in which contexts and prioritizes taking actions that have never been taken in particular contexts, it will quickly obtain the needed interactions for learning an action. Regarding **H3**, contexts of larger sizes capture more complex relationships within the state space, as shown in the example contexts in Figure 2, and should be more informative.

4.2 Domain

We conduct experiments in the open-source rogue-like[5] video game Dungeon Crawl Stone Soup[6] (DCSS). DCSS is a turn-based video game in a 2-dimensional grid world that has a number of properties conducive to evaluating exploration-focused cognitive systems with automated planning components:

- Large state-action space: 100s of monsters, objects, and actions, and a typical game visits 80,000 to 100,000+ tiles (i.e., positions in the grid)
- Partially observable: player does not see a tile until it is within line of sight
- Dynamic: monsters take their own actions independent of the player, and there are time-based events
- Multiple subtasks required to win the game: item management, skill point allocation, and combat
- Natural language text accompanies every item in the game: player can ask for a description of any tile, object, monster, etc., within view or in the player's inventory
- Many actions are probabilistic: spell casting, melee attacks, and other actions may fail
- Simple to make custom experiments for levels and allows for integration with custom AI agents
- Other aspects include: permanent death (i.e., repercussions for poor performance) and procedurally generated levels

We have developed an API[7] for the DCSS game that provides a game state object containing all of the information that a human player would have access to, as well as the ability to execute actions received from an external agent. The agent begins by receiving the initial state and after each action is sent to the game, a new game state object is obtained from the API.

Figure 4 shows screenshots of the graphical version of the game with tiled images (the game also comes with an alternative interface using colored ASCII text displayed in a command line interface). Figure 4a shows an image of what a human sees when playing the normal full game. In the lower right are the items in the agent's inventory, with monster information just above that, and in the top right there is information about the player's health, magical reserves, and a summary of the base attributes of the player including the player's experience level, strength, defense, hunger, currently equipped weapon, etc. The bottom left corner shows the text messages describing events that have happened in ascending order of most recent messages. For example, the second most recent message reads: *"An adder and 2 jackals come into view"*. The center of the image shows the map which is centered on the player. Tiles within the line of sight of the player are lighter, with tiles the player has seen but are no longer in its line of sight being darker, and tiles the player has yet to see are completely black.

Figures 4b and 4c show custom scenarios we designed for our evaluation in learning action preconditions. These scenarios are simply empty rooms with no monsters or objects except for a single orb. We designed these scenarios to evaluate our method in simple initial scenarios, however we plan to use more complex scenarios

[5] A subgenre of role-playing video games known for procedurally generated content and permanent death.

[6] https://github.com/crawl/crawl

[7] We hope to release this API for other AI researchers in the near future.

with a richer action space, monsters, and other objects as part of our future work.

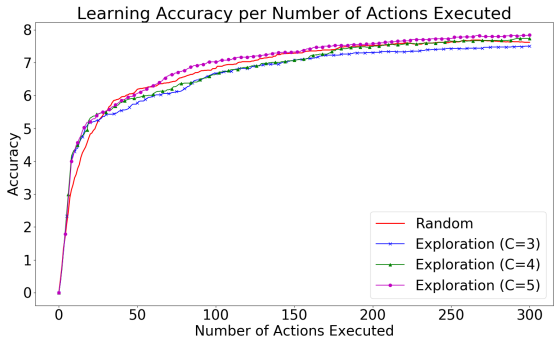
4.2.1 Experimental Conditions. We conduct experiments on two custom scenarios in the DCSS domain. The first scenario (shown graphically in Figure 4b) is a rectangular room surrounded by walls. The player begins in the centermost tile adjacent to the southern wall. The room has a width of nine tiles and height of six tiles. The second scenario (shown graphically in Figure 4c) is a smaller room with 22 reachable tiles surrounded by walls. The room structure was chosen to provide more opportunities to test the diagonal cardinal actions than the first scenario. Each agent is identical but may differ in how they select actions. We compare three exploration agents with context sizes of three, four, and five and one random action selection planner that acts as a baseline.

In these experiments we only considered eight movement actions for the agent to learn (the cardinal directions N,S,E,W,NE,NW,SE,SW). Learning accuracy was calculated by taking all floor tiles in the scenario and applying perfect precondition rules to determine if that action could be taken in that tile. We then reduced the number of tiles used for scoring by removing any tiles where every perfect precondition rule returned true (these were tiles where any action could be taken, so any rule that said an action could be taken in that tile would have been correct). We stored the outcomes of the perfect rules applied to each of these tiles and computed the outcomes of the learned rules. The learning score is the percentage of correct predictions that the learned rule had compared to the perfect rule for each tile in the test set. Each learned rule was given an accuracy score between 0 and 1 (inclusive), and the total accuracy is the sum of the accuracies of each rule. Since there are eight actions, the minimum score is 0 and the maximum score is 8. This scoring function was chosen to be agnostic to the specific rules learned and only identified if the rules contained the correct information (i.e., alternate methods of representing the same rule are both equally correct).

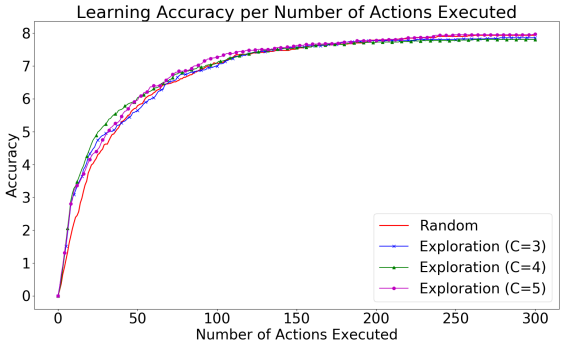
4.2.2 Results. Our results show the performance of an agent performing random exploration as well as agents using our Exploration Planning technique with various context sizes ($C = 3, 4, 5$). We ran each agent for 20 experimental runs and averaged the results over all runs. Eventually, all agents would reach a perfect learning score of 8 after collecting sufficient interactions. However, the results shown in Figures 5a and 5b only show the result up to the first 300 actions as we are interested in examining which approaches learn quickly.

4.2.3 Discussion. Looking at the results in Figure 5, we see that for both scenarios and all agents, hypothesis **H1** is met. After about 100 actions, agents have learned rules that total 75% accuracy or higher, and after about 200 actions most agents have learned six or seven rules perfectly. While it is not shown, after enough actions all approaches learn all eight rules perfectly. It is important to note that even if only random exploration is used, the agent is still able to learn.

Hypothesis **H2** is true for about the first 25 actions in both scenarios. In Figure 5a we see that the exploration planner with a size of five learns faster than all other agents including the random baseline, for most of the scenario. This behavior is more noticeable



(a) Results on the Open-Room Scenario from Figure 4b



(b) Results on the Small Scenario from Figure 4c

Figure 5: Learning Accuracy Score

in Figure 5a than in Figure 5b, we believe this is partly due to the size of the rooms and the number of opportunities to learn certain actions. The exploration agents are more likely to try an action that has not been tried as often in a new context. Since the agent must be adjacent to a wall in the large room to learn the *not wall(T)* condition, the exploration agents have a greater chance of taking the first opportunity to do so.

After the first 25 actions in Figure 5a we see that the exploration agents with contexts of size three and four perform worse than the random baseline agent for rest of the execution (except for the end when the exploration agent (C=3) meets the random agent around action 175). This contradicts hypothesis **H2** for exploration agents of size three and four, and demonstrates that context size is important for exploration agents.

In both graphs, hypothesis **H3** is true most of the time for all exploration agents, where increasing context size either matches or performs better than lower context sizes, except for a brief period in Figure 5b where the exploration agent with context size four performs best (between the 25 and 50 action marks).

While these results are preliminary, they are encouraging and provide motivation for developing improved exploration techniques, and performing an extended evaluation in larger scenarios with more diverse actions. The primary conclusion from these results is that even with random exploration, action learning for movement actions is feasible.

5 EXPLANATION

Our commitment to explanation is intended to help an interested supervisor, or potentially a collaborator, to understand what the agent is doing. At a high level, the agent's reasoning is encapsulated by the following decisions: a) whether to perform exploration or goal achievement; b) (if exploring) where to explore; c) (if achieving) which goal to achieve; and d) how to explore or achieve. Each decision should be transparent, in order to explain to the supervisor the agent's intention. Decisions "b" and "c" take the form of a logical sentence that describe the context or goal to be achieved. Decision "d" is a plan (sequence of actions). Decisions "a" and "c" should give a rationale for selection of exploration, or a particular goal. Therefore, we initially construct an explanation as a tuple $\langle rationale, goal, plan \rangle$.

Applicable rationales for this agent include the following symbols:

- $urgency$: meaning that the need to achieve a supervisor's request promptly takes precedence over the agent's lack of knowledge
- $failure$: meaning that the agent is unable to construct a plan to achieve a supervisor's request, and thus will explore its actions
- $idle - voyaging$: meaning that the agent is not tasked and is visiting a new area where more novel interactions are expected
- $idle - experimenting$: meaning that an agent is not tasked and has an insufficient model to predict the effects of its actions in its current state, and is therefore trying novel actions

While the above explanation is sufficient to make the choices of the agent transparent, it does not shed much light on what the agent has learned or failed to learn, which is a key problem facing humans interacting with learning systems. In future work, to further illuminate this for an exploratory planning agent, we will depict its option space and expected plan traces. An option space will allow a supervisor to see the options the agent thinks it has available to it in the next step or next several steps; missing options or incorrect options will indicate what the agent hasn't learned about the transition model. Expected plan traces will go further in showing, when an agent chooses a plan, why it might choose a poor plan because of an incorrect assumption somewhere down the line.

6 RELATED WORK

Since we are concerned with both exploration and learning relational action models for planning, we draw on prior work from both reinforcement learning (RL) and inductive learning. A difference of this work from RL is that our agent is goal-driven. Given that our agent does not know when actions can be applied (preconditions) and the changes made to the state (effects), we draw on ideas from RL to determine what actions to take in order to obtain enough examples for learning action models.

Hester and Stone ([2017]) discuss multiple exploration reward schemes and reward metrics in intrinsically motivated RL systems. They present two exploration-based intrinsic reward schemes that seek to take the most novel action. Briefly, action novelty takes into account the difference of the current state compared to the most recent state in which an action was executed as well as how uncertain the agents predictions for the state reached after executing the action.

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. For example, novelty is proportional to the number of times an action has been used in a given situation before. The use of contexts in our agent is an attempt to reach a similar notion of novelty: taking actions that have not been tried in similar situations. Currently, our agent takes the action that has been taken the least among all the currently active contexts.

Inductive learning approaches that use heuristics have been effective [4, 13, 15] but may not apply to all learning problems. FOIL [11] 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. Our learning problem is similar to those of OBSERVER and FOIL, hence our use of an inductive learning approach.

To deal with incompleteness of action models, Weber and Bryce ([2011]) give a planning algorithm that reasons about the action model's incompleteness in order to avoid failure during planning. 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 not ever happen).

Human-agent teaming, and specifically the need for explanation in such teams, has been identified as a potential challenge problem for Goal Reasoning [9]. Their work presents a model for how explanations can be exchanged among teammates and used in decision making, and presents several common instantiations of the model. Our work falls under the **Single Supervisor** instantiation, where an agent has a supervior that may make requests of it, and it may need to explain its reasoning or behavior.

Existing applications of explanation to Goal Reasoning have focused primarily on *internal explanations* [1] (i.e., for the benefit of the agent itself) rather than *external explanations* (i.e., for the benefit of an external user). DiscoverHistory [10] generates explanations for why observed environment transitions occurred. More specifically, it generates explanations that contain the external actions (i.e., of other agents) and exogenous event that are most likely to have resulted in the changes in the environment. In a human-robot teaming domain, such explanations have been used to allow a Goal Reasoning agent to determine the plans and goals of other agents, and reason over its own goals [3]. However, even though these explanations are used by an agent that is a member of a team, they are only used internally and never provided to any teammates.

7 FUTURE WORK

This paper presents our preliminary work on action model learning using exploratory planning. While our initial work on this topic

looks promising, a number of areas of future work remain. We have demonstrated the feasibility of our action model learning approach even when the agent performs random exploration of the environment, and improved performance when the agent balances the actions it performs. However, the exploratory approach leaves significant room for improvement. Our current approach attempts to maintain an even distribution of performed actions for each context, but does not attempt to transition between contexts. This can result in the agent spending significant time exploring more common contexts while rarely, or never, exploring less common contexts. Our future work will examine several aspects of this. First, we will develop methods to estimate how well each context has been explored (i.e., the total number of actions taken in each context) and whether any contexts have knowledge gaps (e.g., actions that have not yet been performed in those contexts). This will involve both high-level metrics based on the number of actions performed in each context as well as self-evaluation to determine the agent's ability to correctly act in each context. Second, we will provide mechanisms by which the agent can create goals to move to underrepresented contexts and explore those regions. Using this approach, the agent will be able to continuously evaluate its current exploration and generate plans to guide further exploration. Since there may be multiple underrepresented contexts, the agent can maintain multiple exploratory goals and generate plans that effectively achieve those goal (or a subset of the goals if they can only be partially satisfied).

For example, initially the agent will have no information and will generate goals to explore all contexts. However, since the agent will not yet have an action model, generating a plan to achieve those exploratory goals will not yet be possible until it has performed exploration in the initial context. After initial exploration has been performed, the agent can learn an initial action model and use that model to generate plans to achieve its exploratory goals (i.e., place itself in other contexts). For reasonably complex action models, the initially learned model will likely be incorrect. However, useful information can be obtained from both plan generation failure (i.e., more exploration in the current context is needed) and action failure (i.e., a useful observation that can be used to update the action's model).

Although generating exploratory goals and plans is useful for the agent to learn, outside of its initial training exploration will not be the only motivation of the agent. Thus, it will be necessary for the agent to manage both exploratory goals and other goals. For example, if a teammate or operator provided the agent with a goal to *collect samples from the forest*, the agent would need to manage its exploratory goals along with its provided collection goal. Assuming the agent had already learned an action model that could complete the collection goal, no exploration would be necessary if the agent gave priority to non-exploratory goals. However, if the agent had no existing action model or an incomplete action model, exploration may be required in order to perform the provided goal. If, upon entering the forest, the agent determines its action model for *moving* is not working as expected because of the debris on the forest floor, it would need to evaluate whether it should: (a) continue to work towards the provided goal even if it may consume more resources (e.g., fuel, time) or fail, or (b) perform exploratory actions and relearn an improved action model. This type of reasoning by the agent will involve several important mechanisms of Goal Reasoning, including discrepancy detection (e.g., if an action failed or did not have the expected effect) and goal management (e.g., whether to add exploratory goals).

Pursuing both exploratory goals and provided goals concurrently opens up several important areas of future work related to explanation. When the agent learns an improved action model, it can use the differences between its old and new models to provide explanations to teammates. For example, consider an agent deployed in a new environment containing a slippery surface that significantly impacts movement. After exploring the new environment and learning an updated action model, the agent can provide explanations to its teammates about how its action model has changed. Even if the agent's physical embodiment (e.g., a robot) differs from its teammates' (e.g., humans), the explanations may allow the teammates to more efficiently update their own action models (e.g., learn to walk slowly to avoid slipping).

In a supervisor-supervisee setting, explanations can be used to justify failures to the agent's supervisor. Using the previous example of a slippery environment, if the agent's supervisor was inquiring about why the agent was exceeding time expectations or failing to achieve goals, the agent could justify its poor performance by providing an explanation for the precise reasons it was not performing as expected (i.e., the flaws in its previous action model). Especially if the supervisor is not physically located in the same area as the agent, such explanations would be beneficial because they update the supervisor about unexpected environment conditions and show the agent is capable of self-diagnosis. Similarly, the explanations could be used proactively to inform the supervisor about potential delays in goal completion and any necessary exploration goals that must be performed concurrently.

8 CONCLUSIONS

This paper describes our explainable exploratory planning agent and presented methods to allow the agent to perform exploratory planning, learn the preconditions of its actions, and generate explanations for its behavior. The ability of an agent to modify its action model dynamically is important if it is placed in new environments and is not explicitly provided full domain knowledge. Our approach overcomes this by not only allowing the agent to learn its action model, but also perform exploratory planning such that it can evaluate the success or failure of actions in a guided manner. We feel that explanation is important in such an agent because it allows for explaining changing environments to teammates, explaining how the environment differs from expectations, and explaining why it is achieving exploratory goals in addition to mission-specific goals.

Although our initial implementations for exploratory planning and action model learning leave ample room for improvement, our results in simple scenarios from the Dungeon Crawl Stone Soup game are promising. Even when exploration is random, the agent is still able to learn its action preconditions. When simple exploratory planning is used, the learning performance of the agent is improved. These results provide motivation for numerous paths of future work, as described in the previous section, and more thorough evaluations in increasingly complex scenarios in the Dungeon Crawl Stone Soup domain.

ACKNOWLEDGMENTS

Thank you to DARPA for supporting this research. 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 Department of Defense or the U.S. Government.

REFERENCES

[1] A. Aamodt. 1993. Explanation-driven case-based reasoning. In *Proceedings of the First European Workshop on Case-Based Reasoning*. Springer, Kaiserslautern, Germany, 274–288.

[2] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: Theory and Practice*. Elsevier.

[3] Kellen Gillespie, Matt Molineaux, Michael W. Floyd, Swaroop S. Vattam, and David W. Aha. 2015. Goal Reasoning for an Autonomous Squad Member. In *Proceedings of the Workshop on Goal Reasoning (held at the 3rd Conference on Advances in Cognitive Systems)*. 52–67.

[4] Frederick Hayes-Roth and John McDermott. 1978. An interference matching technique for inducing abstractions. *Commun. ACM* 21, 5 (1978), 401–411.

[5] Todd Hester and Peter Stone. 2017. Intrinsically motivated model learning for developing curious robots. *Artificial Intelligence* 247 (2017), 170–186. https://doi.org/10.1016/j.artint.2015.05.002

[6] Jörg Hoffmann. 2003. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. *Journal of Artificial Intelligence Research* 20 (2003), 291–341.

[7] Mark Law, Alessandra Russo, and Krysia Broda. 2015. The ILASP system for learning Answer Set Programs. https://www.doc.ic.ac.uk/~ml1909/ILASP. (2015).

[8] Matthew Molineaux, Dustin Dannenhauer, and David W. Aha. 2018. Towards Explainable NPCs: A Relational Exploration Learning Agent. In *Proceedings of the Knowledge Extraction from Games Workshop (held at the 32nd AAAI Conference on Artificial Intelligence)*. AAAI.

[9] Matthew Molineaux, Michael W. Floyd, Dustin Dannenhauer, and David W. Aha. 2018. Human-Agent Teaming as a Common Problem for Goal Reasoning. In *Proceedings of the AAAI Spring Symposium on Integrating Representation, Reasoning, Learning, and Execution for Goal Directed Autonomy*. AAAI Press.

[10] M. Molineaux, U. Kuter, and M. Klenk. 2012. DiscoverHistory: Understanding the Past in Planning and Execution. In *Proceedings of the 11th International Conference on Autonomous Agents and Multi-Agent Systems*. IFAAMAS, Valencia, Spain, 989–996.

[11] J Ross Quinlan. 1990. Learning logical definitions from relations. *Machine learning* 5, 3 (1990), 239–266.

[12] Pedro Sequeira, Francisco S Melo, and Ana Paiva. 2011. Emotion-based intrinsic motivation for reinforcement learning agents. In *Proceedings of the International Conference on Affective Computing and Intelligent Interaction*. Springer, 326–336. https://pdfs.semanticscholar.org/28e6/3b995213efe0eac8169eae3ae8ffe6eb04e3.pdf

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

[14] Xuemei Wang. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of the 12th International Conference on Machine Learning*. 549–557. https://doi.org/10.1016/B978-1-55860-377-6.50074-8

[15] Larry Watanabe and Larry A Rendell. 1990. Effective Generalization of Relational Descriptions.. In *Proceedings of the 8th National Conference on Artificial Intelligence*. 875–881.

[16] Christopher Weber and Daniel Bryce. 2011. Planning and Acting in Incomplete Domains. In *Proceedings of the International Conference on Automated Planning and Scheduling*. 274–281.