Oh, Now I See What You Want: Learning Agent Models with Internal States from Observations

Panagiotis Lymperopoulos
Tufts University
Medford, United States
plympe01@tufts.edu

Matthias Scheutz
Tufts University
Medford, United States
matthias.scheutz@tufts.edu

ABSTRACT

Learning behavior models of other agents from observations is challenging because agents typically do not act based on observable states alone, but usually take their internal, for external agents unobservable, states such as desires, motivations, preferences, and others into account. Consequently, methods that only use observational states for modeling other agents' behaviors are insufficient for capturing and predicting agent behavior, especially for agents with rich internal processes. We propose a novel approach to online agent model learning that works incrementally with limited data, provides fine-grained and interpretable descriptions of the agent's behavior, and, most importantly, is able to hypothesize agent-internal states to better explain observed behavioral trajectories. We show in various proof-of-concept experiments that our method avoids the pitfalls of common agent-modeling strategies when agent-internal states govern behavior and is able to build accurate and interpretable behavior models. We also discuss how the method can work in conjunction with existing approaches (e.g., for goal recognition) to facilitate better modeling of open-world agents.

KEYWORDS

Agent Models, Internal states, Online learning

ACM Reference Format:
Panagiotis Lymperopoulos and Matthias Scheutz. 2024. Oh, Now I See What You Want: Learning Agent Models with Internal States from Observations. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024,* IFAAMAS, 9 pages.

1 INTRODUCTION

Dynamic multi-agent environments with unknown agents are notoriously difficult to negotiate because one's own performance might depend on the actions taken by other agents [9, 17, 22]. One way to reduce uncertainty and make one's own plans more likely to succeed in such contexts is to quickly learn behavior models of those agents online from observations and use them to predict their actions [4]. However, there is a core challenge with such online observational learning: Agents typically do not solely act based on observable states, but take their internal, for external agents unobservable, states such as desires, motivations, preferences into account when deciding on their actions.

Consider the example of a person walking into a bakery and buying either a cheese strudel or baklava every morning. Despite having full knowledge of the environment in the bakery, we cannot attribute the choice of pastry to any environment feature as, for instance, both pastries are available and fresh. We, therefore, have two choices when modeling this behavior: Either treat the choice as random, or hypothesize that there is an unobserved internal state of the person that determines which pastry they choose at any given visit. Hypothesizing internal states then allows us to separately consider situations distinguished by those states (e.g., if they buy a strudel, they also later sit down for coffee, whereas if they buy baklava they leave the store). Furthermore, as internal states are not dependent on environment features, they can enable us to better anticipate what the person may do in previously unseen situations (e.g., ordering a strudel when visiting a different restaurant). Note that the choice in the pastry shop could be completely independent of other goals the person has (e.g., to finish a contract at work, to meet the family out for movies, etc.): They may be going to work, returning home or going for a walk, none of which have any bearing on what that person may do in the pastry shop. In other words, this is not a question of more coarse-grained *goal recognition*, but rather one of more fine-grained *behavioral dispositions* that drive human actions in particular circumstances, based on their internal predispositions.

Past approaches to learning agent models from observed behavior are not well-suited in this setting for a few reasons: 1) they make no distinction between unobservable aspects of the environment and unobservable agent-internal states [7]; 2) they assume competent and near-optimal goal seeking behavior (rather than allowing for agents to not follow larger goals, or to not behave near-optimally) [15]; 3) they require a large number of observations [25]; 4) they are often too fine-grained to generalize to unseen situations, or 5) they are too abstract to allow for accurate step-by-step predictions.

We will address these challenges with an online learning method that incrementally forms environmental state abstractions and hypothesizes novel agent-internal states in order to capture agent dispositions, i.e., states that are equivalent with respect to the agent's chosen behavior and how the agent transitions between them. The model makes no assumptions about the nature of the observed dispositions and does not attempt to infer the agent's larger goals. Rather, it refines its state abstractions with every new observation and hypothesizes novel agent-internal states when other explanations of observed behavior lead to worse predictions. And critically, it is immediately usable for making behavioral predictions even with minimal observations and able to explain, based on its learned

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

state transition model, why it expects the other agent to select a particular action in a given state, factually and counterfactually.

The rest of the paper is organized as follows. We first review other approaches to agent model learning and then introduce our proposed method in detail, followed by demonstrations showing that common agent-modeling paradigms cannot adequately model agents that act according to agent-internal states even in very simple crafting tasks in a 2D Gridworld environment. We show how our model succeeds in modeling such agents and furthermore explain the learned model to demonstrate how agent-internal state modeling allows us to simulate agent behavior in previously unseen situations with no additional observations.

2 RELATED WORKS

Current agent modeling strategies aim to model behavior of competent, goal-seeking agents at varying levels of detail and abstraction: Behavioral cloning (BC) [5, 21] learns policies via supervised learning from observations of near optimal trajectories to try to replicate exactly the modeled agent's strategy. Inverse Reinforcement Learning (IRL) [3, 11, 25] instead takes a more flexible approach and approximates a reward function for the expert's task, then learns policies according to that reward function through environment interactions. Some adversarial training strategies also aim to improve the robustness of learned policies [2, 12]. These methods yield low-level representations of agent behavior in terms of policies: They can be used to simulate their behavior in similar environment states but typically generalize poorly to new situations and require extensive training data to adapt. In addition, they typically do not provide high-level, explainable representations of agent behavior, nor do they attempt to model agent-internal states.

Goal and plan recognition [6, 15, 20, 23] methods infer agents' goals by comparing observed trajectories with domain theories. The inferred goals can be both general and explainable, which can make these methods useful for integrating with multi-agent task planners and producing explanations of behavior. While the planning domains or value functions used in domain theories may be learned through observations or environment interactions [1, 8, 14], these methods typically require that agents are competent and goal-seeking, and they assume that a predetermined set of goals that agents may have is known beforehand. These requirements are in conflict with what we may encounter in the open world, where agents may be incompetent or their goals, if any, may be irrelevant to the situation we are interested in modeling.

Recent work in learning agent's skills [24] bears some similarities to our work as it attempts to learn interpretable descriptions of an agent's capabilities. This approach represents capabilities similarly to PDDL operators, expressed in a user-specified language. They initially collect agent trajectories and extract partial descriptions of the agent's capabilities, then they query the agent with specific tasks to revise and finalize capability descriptions. This approach is effective at learning abstract descriptions of a planning agent's capabilities, but assumes deterministic goal-seeking behavior, does not learn incrementally, and requires the learner's ability to actively query the observed agent with tasks. In contrast, our method learns an agent's dispositions incrementally by passively observing possibly stochastic behavior.

More broadly, our work follows an existing line of research on open-world learning [10, 13, 16, 19]. Our method aims to fill the gap left by other agent-modeling strategies in this setting: It is able to learn online from limited data, hypothesize agent-internal states if necessary and enable fine-grained modeling and simulation of agent behavior while not relying on competency or optimality of modeled agents.

3 ONLINE LEARNING OF AGENT DISPOSITIONS

We now formally present our proposed framework for modeling an agent's dispositions which depend on observable environmental and unobservable agent-internal states. The approach relies on forming clusters of labeled MDP states in which the agent is disposed to act similarly. The model continuously evolves as new observations are made: It expands to better explain observed behavior and it consolidates behaviorally similar clusters to remain compact and to be able to generalize them. When the agent behaves in distinct ways in the same cluster, the model hypothesizes internal states to explain the difference. We describe the "base version" of the model that does not hypothesize agent-internal states and indicate the necessary additions required to model internal states in the "full model".

3.1 Background

A *labeled Markov Decision Process (l-MDP)* is a tuple $M = \{S, A, P, R, \gamma, AP, L\}$ where S is the l-MDP state space, A is a discrete action space, $P : S \times A \to [0, 1]$, is the transition probability function, $R : S \times A \times S \to \mathbb{R}$ is the environment reward function, γ is the discount factor, AP is the set of atomic propositions and $L : S \to 2^{AP}$ is a labeling function that for a given state s indicates truth values for each element of AP.

A trajectory $X = \{\tau_t = (s_t, L(s_t), a_t), t : 1 \dots T\}$ in the labeled MDP is a sequence of T trajectory steps, which consists of state observations, state labels and actions taken by an agent in those states. We consider the problem of observing agent trajectories in M online and maintaining a model of its behavior. We now present the architecture of our agent model.

A state description d is a conjunction of $p_i \in AP$ or their negations. In this work we sometimes refer to state descriptions as sets of propositions for notational convenience. A state description describes a set of states $S_d = \{s \mid d \subseteq L(s)\}$ in which d "subsumes" their description. The labeling of an MDP state $L(s)$ is a full state description as it includes all (observable) propositions or their negations. A partial state description contains a subset of AP or their negations. The $\subseteq$ operator is used to indicate subsumption.

Least general generalization. Two state descriptions d and d' can be generalized into a single state description d^* such that $S_{d^*} = S_d \cap S_{d'}$. Then, d^* is the least general generalization (LGG) [18] of d and d'. For propositional state descriptions, $LGG(d, d') = d \cap d'$ when descriptions are treated as sets. The LGG of two state descriptions is the least general description that subsumes both d and d'.

Planning. A planning domain $\Sigma = (\mathcal{S}, \mathcal{O})$ is a tuple, where $\mathcal{S}$ are state descriptions expressed using atoms in AP and $\mathcal{O}$ are planning operators associated with low-level executors Ξ. Each executor

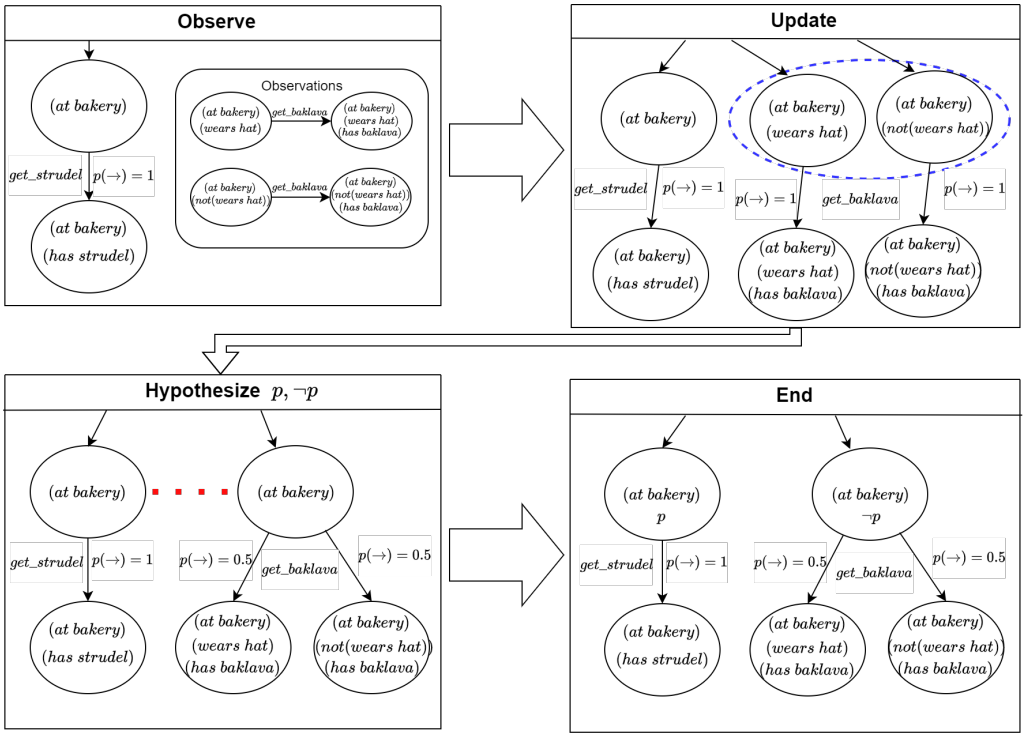

Figure 1: Illustration of a model update on the bakery example. In the upper left panel, an existing agent model receives two examples of an agent exhibiting previously unseen behavior. On the upper right panel, the two examples are used to update the model, creating new clusters to explain the transitions. Two clusters (circled in blue) have identical action distributions. As a result, they are merged in the lower left panel into a single more general cluster. However, the new cluster overlaps with an existing one (indicated by the red dotted line), while describing very different behavior: getting a strudel and getting a baklava. To explain the distinction, the model hypothesizes an agent-internal state described by p and $\neg p$. The lower right panel shows the final state of the model after the update.

$\xi \in \Xi$ is a program that uses low-level actions in A to advance the state of the MDP M.

A planning task $T = (\Sigma, s_0, d_g)$ is a tuple consisting of a planning domain Σ, an initial state description s_0 and a goal state description d_g. The task can be supplied to a planner, which provides a plan P: a series of operators that transition the planning state from s_0 to some state s_g s.t. $d_g \subseteq s_g$.

3.2 Dispositional Agent Model

An agent model (to be learned) is a tuple $\mathcal{A} = (C, E, AP_I, \mathcal{I}, \theta_m, \theta_p)$ consisting of a set of clusters $C = \{c_i, i : 1 \dots C\}$, a map $E : C \times C \to \mathbb{R}$, a set of atoms AP_I, a propositional knowledge base $\mathcal{I}$ and two hyperparameters $\theta_p, \theta_m \in \mathbb{R}$. Each cluster $c_i = (d_i, p_i(a), \beta_i, \gamma_i)$ is a tuple consisting of a partial state description d_i, a probability distribution $p_i(a), a \in A$ over the action set of the MDP, β_i is a conjunction of (negated) propositions in AP_I and γ_i is a disjunction of (negated) propositions in AP_I. We parameterize $p(a)$ by a vector of observed counts for each action and obtain the probability of each action in a cluster by normalizing by the sum of action counts observed in that cluster. To update the distribution online, we simply increment the observed action counter.

The model expands AP_I to hypothesize agent-internal states. The knowledge base $\mathcal{I}$ holds the model's belief of truth values of atoms in AP_I and is updated as observations are received. The formulas β_i, γ_i determine the interaction of each cluster with $\mathcal{I}$. β_i indicates (negated) propositions asserted to $\mathcal{I}$ when the agent is in a state belonging to c_i. γ_i holds states of $\mathcal{I}$ in which the agent has been observed to occupy states belonging to c_i. The values θ_m and θ_p are hyper-parameters controlling the level of abstraction of the agent model by determining when behavior is very similar and very different than previously observed.

Before we describe the algorithms involved in updating the model, we briefly return to the bakery example from the introduction to demonstrate the main ideas involved in learning the agent model. Figure 1 illustrates the update process for an agent model of the bakery example. On the upper left panel, we consider (part of) an existing agent model for an agent entering a bakery and getting a strudel. However, now we receive observations of the agent in the bakery getting baklava instead of a strudel. Since these observations are not explained by the current model, in the upper right panel the model is updated with the new observations. Circled in blue are two clusters created to explain the observations. However, since the agent's behavior is identical in both clusters,

Algorithm 1 Update Model

```
1: Input:
2: Agent Model 𝒜 = (C, E)
3: Observations (τ_t, τ_{t+1})
4: Anomaly threshold θ_p
5: procedure UPDATE(𝒜, τ_t, τ_{t+1})
6:     c_a ← ASSIGNCLUSTER(C, τ_t)
7:     p_a(a) ← OnlineUpdate(p_a(a), a_t)
8:     c_b ← ASSIGNCLUSTER(C, τ_{t+1})
9:     p_b(a) ← OnlineUpdate(p_b(a), a_{t+1})
10:    E(c_a, c_b) ← E(c_a, c_b) + 1
11:    MERGECLUSTERS(C,E,θ_m)
12: end procedure
13: procedure CREATECLUSTER(C, d)
14:    if ∃c_k ∈ C s.t. d_k = d & not c_k.isStable() then
15:        return c_k
16:    end if
17:    if ∃c_k ∈ C s.t. d_k = d & c_k.isStable() then
18:        p, ¬p ← InventAtom()                ▷ Hypothesize internal state
19:        β_k ← β_k ∪ {¬p}
20:        β ← β ∪ {p}
21:    end if
22:    Initialize(p(a))
23:    c' = (d, p(a))
24:    C ← C ∪ {c'}
25:    return c'
26: end procedure
27: procedure ASSIGNCLUSTER(C, τ_t)
28:    A = {c_i ∈ C | d_i ⊆ L(s_t)}
29:    if A = ∅ then
30:        c* ← CREATECLUSTER(C,L(s_t))
31:    else
32:        c_m ← arg max_{c_k ∈ A} |d_k|
33:        B = {c_i ∈ C | d_i = d_m, }
34:        c_m ← arg max_{c_k ∈ B} p_k(a_t)     ▷ Most likely cluster
35:        if p_m(a_t) < θ_p then               ▷ Anomaly
36:            c* ← CREATECLUSTER(C,L(s_t))
37:        else
38:            c* ← c_m
39:        end if
40:    end if
41:    UPDATEKB(ℐ, c*)
42:    return c*
43: end procedure
```

Algorithm 2 Merge Clusters

```
1: Input
2: Clusters C
3: Transitions E
4: Merge Threshold θ_m
5: procedure GENERALIZE(d,d',β,β',γ,γ')
6:    d* ← ∅
7:    d* ← LGG(d, d')
8:    β* ← β ∪ β'
9:    γ* ← γ ∪ γ'
10:   for p ∈ β* do
11:       if ¬p ∈ β* then
12:           β* ← β*\{p, ¬p}
13:       end if
14:   end for
15:   return d*, β*, γ*
16: end procedure
17: procedure MERGECLUSTERS(C, E, t_m)
18:   if |C| = 1 then
19:       return
20:   end if
21:   D ← COMPUTECLUSTERDISTANCES(C)
22:   Let (a, b) = arg min D
23:   if D[a, b] < θ_m then
24:       d', β', γ' ← GENERALIZE(d_a, d_b, β_a, β_b, γ_a, γ_b)
25:       p' ← p_a ⊕ p_b
26:       if ∃c_m ∈ C s.t. d_m = d' then
27:           if ModelInternalStates then
28:               q, ¬q ← InventAtom()
29:               β_m ← β_m ∪ {¬q}
30:               β' ← β' ∪ {q}
31:           else
32:               p' ← p_a ⊕ p_b ⊕ p_m
33:           end if
34:       end if
35:       c = (d', p', β', γ')
36:       C ← (C\{c_a, c_b}) ∪ {c}              ▷ Update C
37:       E ← MERGETRANSITIONS(E, c_a, c_b, c)
38:       return MERGECLUSTERS(C, E, t_m)
39:   end if
40: end procedure
```

they are merged in the lower left panel into a single cluster that describes a more general situation than either its constituents, since wearing a hat does not differentiate the agent's behavior. After the merge, two clusters are in conflict (red dotted line): They describe the same environmental situation—being at the bakery—but capture very different behavior. As a result, the model hypothesizes an agent-internal state, modeled by a proposition p, that explains the distinction in behavior. The lower right panel shows the final model after the update. Importantly, propositions modeling hypothesized internal states do not have interpretations according to observable external environment states. They can be interpreted functionally however: p means that the agent acts a certain way (gets a strudel). Finally, future observations of behavior after obtaining a baklava may reveal that the two states on the bottom right panel, which are differentiated only by wearing a hat, need to also be merged if the agent behaves identically in those states.

Algorithm 1 and Algorithm 2 formally describe how the model is updated online. The model may start out as empty or hold information from prior observations. Lines in blue are necessary for modeling agent-internal states and are omitted in the base agent model. Lines 5-12 of Algorithm 1 indicate the update procedure. Given two adjacent trajectory steps, the model first assigns them to a cluster, updates the cluster distributions with the observed actions and records the cluster transition. Finally, the model merges similar clusters.

The cluster assignment procedure is shown in lines 27-43. The agent checks if the observation fits in any known cluster, and creates a new one if it does not (line 30). Otherwise, it selects the least general cluster (line 32) that captures the current step. When modeling internal states, multiple clusters may describe the same set of states, but differ in action distribution. In that case, the appropriate cluster is assigned by maximum likelihood over those clusters' action distributions (lines 33-34). Finally, in lines 35-39, a new cluster is instead created and assigned to the step if the observed action is too unlikely ($< \theta_p$) under the previously selected cluster. As a result, the observed state-action pair is assigned to a cluster that adequately explains it, creating a new very specific cluster for it if necessary. Finally, the model calls the UPDATEKB procedure (see

appendix), which updates $\mathcal{I}$ with the assertions of the selected cluster and records the state of $\mathcal{I}$ in which the cluster was visited.

When creating a new cluster (lines 13-26) the algorithm first checks if one with the same description already exists and returns it if it does. When internal states are not modeled, there is no way to make a distinction in behavior in the same state cluster, and so the model will simply update the action distribution of that cluster in its update. When hypothesizing internal states, and the action distribution of the cluster in question is considered stable because sufficient samples from it have been observed, the model invents a new propositional atom to model the hypothesized internal state governing the difference in behavior (lines 17-20). Since those clusters distinguish two different agent-internal states, the assertion of the corresponding (negated) atom is added to each cluster.

Cluster merging happens after every update to maintain a compact model. Algorithm 2 shows the recursive merging procedure. The base cases of the recursion occur in line 17 and 22, when there are no more merges possible because there is only a single cluster or all clusters capture sufficiently different behavior. Line 20 uses the ComputeClusterDistances procedure (see appendix) to compute the distance between the action distributions of all pairs of clusters in which actions have been observed. The distance between two clusters $D[c_a, c_b] = JSD(p_a(a), p_b(a))$ is the Jensen-Shannon distance of their action distributions. If two clusters are sufficiently similar ($D[c_a, c_b] < \theta_m$), they are merged by generalizing their descriptions, assertions and consistencies and combining their action distributions (line 24-25). The generalization procedure is shown in lines 5-15. Cluster descriptions d_a, d_b are generalized by taking the LGG of the two clusters. Formulas β_a, β_b are combined by taking their union and removing contradictions. Formulas γ_a, γ_b are combined by taking their union.

If generalization yields a cluster description that already exists in the model, the model hypothesizes an internal state that distinguishes the two clusters (lines 27-30). Note that if the distinction is unnecessary because the action distributions are very similar, they will be merged in the next recursive call. If internal states are not model, the distributions of the two clusters are combined (line 32).

The set of transitions is also updated by consolidating incoming and outgoing transitions from the the old clusters to the new one and then removing the old transitions. The merge is completed by a recursive call that merges another pair of clusters if necessary.

Algorithm complexity. Assigning an observation to a cluster with the AssignCluster procedure in algorithm 1 requires $O(|C|)$ time, as every cluster needs to be examined. This includes the constant time required to create a cluster if necessary. Merging clusters using the MergeCluster procedure in algorithm 2 is the most expensive operation. In the worst case, in a single update step, merging 2 clusters may induce a sequence of merges that combine all existing clusters. Cluster distance computation takes $O(|C|^2)$ time for the first merge of the sequence and $O(|C|)$ for every subsequent merge, as only distances for the newly merged cluster need to be computed. The Generalize procedure takes at most $O(|AP| + |AP_I|)$ time if the merged cluster descriptions involve all observable and invented propositions. Overall a single update of the model requires at most $O(|C|^2(|AP| + |AP_I|))$ time. In computationally constrained contexts, the MergeClusters procedure can be modified to upper-bound $|C|$ by forcing a merge of the most similar clusters, sacrificing model fidelity for more efficient computation.

Figure 2: Illustration of 2D Gridworld environment. The initial environment (left) from which trajectories are observed has a number of trees the agent can collect wood from to craft other items. In the novel environment (right) all the trees are cut but planks can be obtained by interacting with a trader.

Simulating behavior using the agent model. A learned agent model can be used to simulate an agent's behavior, e.g., to be used with multi-agent planners or to simply predict or replicate the agent's behavior. Since the model captures the agent's dispositions and transitions between them, the agent's behavior can be simulated by observing the current environment state, assigning it to a cluster and then sampling the next cluster the agent is likely to transition to. Agent internal states are automatically inferred from the most recent observed trajectory, but can also be manually selected for simulation.

Then, the action sequence required to transition the environment state can be generated by a planner using appropriate planning operators and executors or by other means such as learned policies. In this work we are not concerned with learning these low-level behaviors and so in our experiments we use a handcrafted planning domain. The algorithms used to simulate behavior using the agent model and a planning domain are shown and discussed in detail in the appendix.

4 PROOF-OF-CONCEPT EXPERIMENTS

We performed an experimental evaluation to confirm that our proposed disposition learning approach can handle behaviors that depend on hypothesized agent-internal states while other state-of-the-art agent model learning approaches cannot. Note that for this confirmation a single small experiment that shows the failure of other approaches and the success of ours is sufficient, there is no need for a large scale evaluation (which would show the same failure of current approaches).

We first describe the test-bed that we used in experiments and the algorithms we compared. Then we present the experimental results and inspect the learned model to demonstrate its explainability. Finally, we apply the learned model to a new environment to demonstrate how internal-state modeling can assist predicting behavior in the open-world.

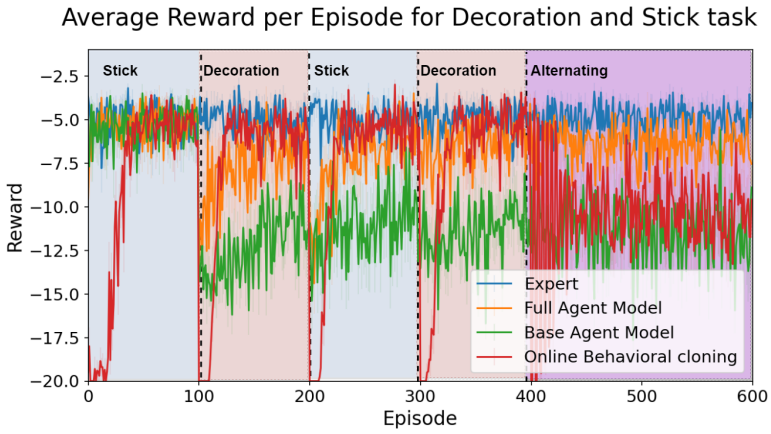

Figure 3: Comparison of algorithms when the expert agent changes preferences. Initially, all methods can replicate the agent's behavior perfectly. As preferences switch, BC always adapts but not before overwriting what it already knows. The base model cannot distinguish between the expert's preferences and thus cannot replicate the expert successfully. Only the full agent model is able to model the expert's preference and replicate its behavior even when preferences switch after every episode.

4.1 Environment Testbed

We evaluated our method in a 2D Gridworld illustrated in Figure 2 consisting of trees, the crafting agent whose behavior we want to learn from observations and additional trader agents. The (crafting) agent can collect wood from trees and use it to craft other items. We define two task settings: one without traders where the agent needs to gather wood from the environment to craft a target item and one with traders where all trees are cut and the agent needs to interact with traders to receive crafting materials. The agent can navigate in the four cardinal directions, craft five items in total and interact with traders yielding an action set of size 10.

- **Task 1: Resource gathering.** The agent is spawned randomly in a 10×10 arena with randomly placed trees and no traders. The agent aims to create a wooden item. Depending on its internal preference that may be a *stick*, or a *decoration*.
- **Task 2: Trading.** The agent is spawned randomly in a 10×10 arena where all trees are cut but traders can supply *planks*, an intermediate ingredient for crafting a *stick*, or *decoration*. The goal of the agent is to craft a wooden item, but as before the choice of item depends on an internal preference.

In both tasks, an observation episode ends once a target item has been crafted. The environment reward function is -1 for every step in which the target item is not crafted. Exact recipes and environment generation parameters are available in the appendix.

Evaluation. To evaluate our algorithm we model the behavior of an "expert": a competent, near-optimal goal-seeking agent that aims to craft a wooden item. We use the expert's task performance as a gold standard, and compare, using task performance as a metric, how agents guided by behavioral models are able to simulate its behavior. In the first experiment we observe trajectories from Task 1 where the choice of *decoration* and *stick* varies non-randomly. After each trajectory is observed, we run one simulation in a randomly initialized environment using the behavioral models and measure the task performance. In the second experiment we take the models trained on Task 1 and measure how well they can simulate the agent's behavior in Task 2. We repeat the same experiment for both wooden items by showing each model a single trajectory from Task 1 for each item to indicate the agent's crafting choice. The models observe no trajectories from Task 2.

Algorithms. We evaluate the agent model with and without internal-state modeling and compare it against a *behavioral cloning* baseline model adapted for online use. While we also train an , *IRL* baseline model and discuss its learned reward function, we do not include an IRL policy in our main experiments because it would require extensive additional training to learn a policy, making the comparison to BC and our agent model unfair. Furthermore, we do not include goal and plan recognition methods in our experiments, as given a planning domain and a set of candidate goals for the agent, it would be trivial to identify the correct item to craft. The point of our experiments is not to achieve the agent's goal but to simulate the moment-by-moment behavior of the agent, achieving goals in the process if they exist.

In the agent models, we set default values $\theta_p = 0.01$ and $\theta_m = 0.1$. The value of θ_m is set to avoid merging clusters when only a single action has been observed. We consider action-distributions stable after observing 10 samples. We adapt behavioral cloning to the online setting by updating the model regularly after each observed episode. We also keep a buffer of past experiences to stabilize learning. After each observed episode, we run two gradient descent epochs with minibatches drawn from the memory buffer as well as all the latest data. Additional epochs did not yield different results. Additional training details for the BC baseline are available in the appendix.

For the agent guided by our model, we use a planning domain with the same operators and executors as the expert agent. For the BC baseline, we use the learned policy. Importantly, we do not let any model learn from its interactions with the environment.

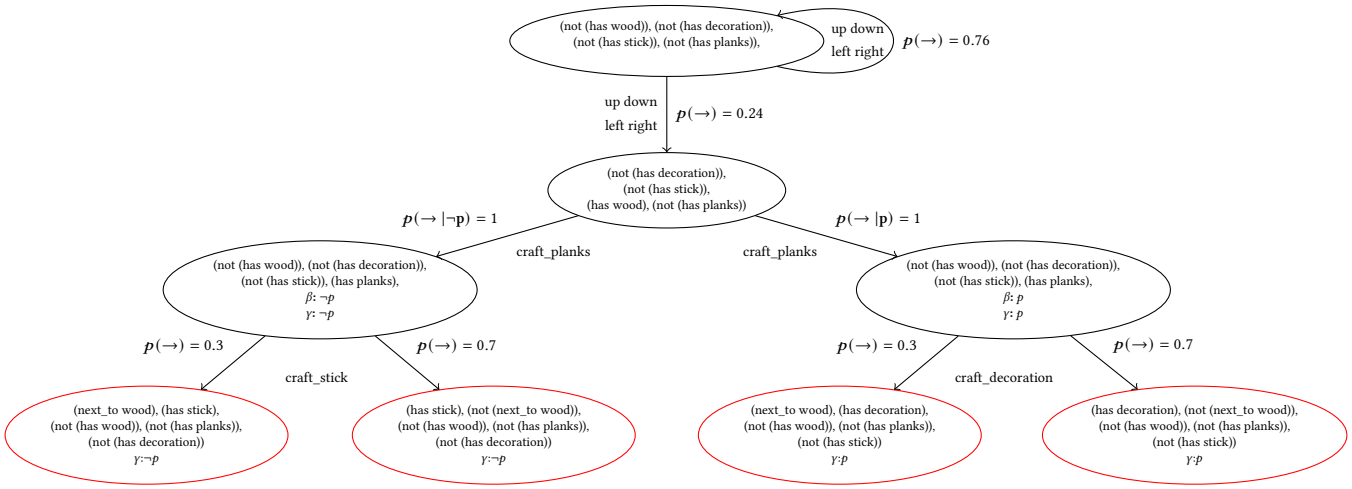

Figure 4: Graph representation of the learned agent model for Task 1. Nodes represent state clusters and edges denote observed transitions between clusters. Edge labels show the action taken and transition probability ($p(\rightarrow)$) conditioned on internal states when relevant. Each cluster is depicted with its learned state description and internal state information when necessary. The nodes in the third level are distinguished by invented predicates, modeling the agent's internal preference for crafting *stick* or *decoration* items. Given the preference, the transition to a state where one or the other is crafted is deterministic. The bottom layer includes clusters which happen to be terminal states (red). Since no actions are observed in those states, these clusters are never merged. In clusters where the formulas β and γ are non-empty, they are also displayed.

4.2 Experiment 1: Modeling Internal States

In this experiment we evaluate how modeling strategies respond to behavior due to agent-internal states.

Experiment Settings. We first run 400 episodes where the expert agent switches preference every 100 episodes. Then, we run an additional 200 episodes where the expert changes preference after each episode. As in the previous experiment, after each episode we update the models and run one episode with behavior derived from the agent models. Here we measure the reward over the entire trajectory and repeat the experiment 20 times. We also train GAIL [12], an *IRL* algorithm that learns robust reward functions with adversarial training, on the entire set of observations from the 600 episodes, allowing it to gather up to 2×10^5 observations from interacting with the environment online as the modeled agent. We record the estimated reward of taking the two crafting actions in question when all necessary materials have been gathered. We repeat the experiment 10 times and report the average and standard deviation. Additional training details are available in the appendix.

Results. Figure 3 shows the rewards gathered by each agent, averaged over 20 runs. In the first 100 steps, consistent with the previous experiment, all methods are able to learn a successful model of the agent for the *stick* task, with BC requiring more observations than our agent models.

Once the agent's preference shifts, all models suffer in performance. The BC agent is initially unable to solve the task in all 20 repeats, after which its performance recovers at a similar rate to learning from scratch. This is expected, as BC has no way to model agent-internal states, and so simply overwrites its previous knowledge. The base agent model may not drop as far in performance as BC, but it is also is unable to recover successfully. This is because it is unable to differentiate between the expert's prior and current preferences and thus learns that at all times, the expert crafts one or the other object with equal probability. The full agent model drops the least in performance and recovers to close-to-expert performance. Interestingly, after sufficient observations, BC outperforms the full agent model. This is because the agent model, by not overwriting past experiences, occasionally produces noisier runs that lead to slightly longer episode lengths.

When the agent's preference shifts again two more times, a similar pattern holds: BC learns again as-if from scratch, showing that it indeed overwrites past experiences. The base agent model converges to selecting randomly which item to craft leading it to fail about half the time, and the full agent model recovers and maintains its performance, showing no drop when the expert shifts to crafting *decoration* items for the last time.

In the last 200 steps where the expert alternates preference on every episode, BC performance initially varies rapidly as it is still crafting the correct item on every other episode. Then, it slowly converges to similar performance to the base agent model, as neither can model the agent-internal crafting preference as it shifts rapidly. In contrast, the full agent model tracks the changes in preference of the expert agent and is able to maintain high performance.

Finally, we examine the reward estimated by GAIL for crafting a *decoration* and *stick* when all prerequisite items are available. Crafting *decoration* gives an estimated reward of 2.14 ± 0.15 and crafting *stick* gives 2.02 ± 0.27, statistically equal rewards. This is to be expected, as IRL relies on environment features and the agent's action to estimate rewards and is unable to model agent-internal states.

Agent	Decoration	Stick
Expert	-12.10 ± 1.90	-11.60 ± 2.52
Base Agent Model	-17.80 ± 1.39	-16.80 ± 1.75
Full Agent Model	-12.80 ± 1.80	-12.80 ± 1.92
Online Behavioral Cloning	-20.00 ± 0.00	-20.00 ± 0.00

Table 1: Comparison of agent modeling methods in simulating behavior in previously unseen situation (task 2). Behavioral cloning cannot generalize to the new task as it has never seen examples of interacting with the trader. The full model performs almost as well as the expert because it can model the agent-internal state.

Inspecting the agent model. Figure 4 shows the learned agent model after all 600 trajectories are observed. The model converges to 8 clusters in total. In the the four clusters of the top three layers, the proposition *(next_to wood)* is generalized away as it is irrelevant to the agent's behavior. The cluster in the top represents the condition where the agent has no items gathered, and navigates to collect wood using navigation actions. The navigation actions either transition it to the cluster below or back to itself, as it navigates the world to collect wood. This leads into a cluster of states in which the agent has wood and crafts *planks*. After this point, the observed trajectories diverge according to the agent's internal preference for *stick* or *decoration* items. In response, the agent model builds two clusters which differ only by an invented atom. This atom is invented to explain the difference in behavior between the two preferences. The probability of transitioning between the middle and left (or right) state is 1, given the agent's preference. The last layer contains only terminal clusters (red). Since these clusters happen to capture final states of the MDP, where no actions are taken, there is no action distribution to determine which should be merged, and thus stay fully specified. Since wood is relatively scarce, final states next to wood are relatively rare.

4.3 Experiment 2: Transfer to Novel Situations

In this experiment we demonstrate how our agent model can generalize to unseen situations and how modeling internal states enables more accurate simulation of agent's behavior.

Experiment Settings. We use the learned agent models and BC policy learned on the first task and apply it to the second task. We prime each model with a single trajectory from Task 1 that matches the crafting target we simulate for. No learning takes place during this experiment. We run 20 episodes in Task 2 for each crafting target preference, repeat the experiment 10 times and report sample mean reward and standard deviation for each agent.

Results. Table 1 summarizes the results of this experiment. Not surprisingly, BC is unable to simulate what the expert will do for both crafting targets because it is unable to simulate behavior in previously unseen situations: nowhere in its training trajectories is there an example of the agent interacting with a trader, even though the *interact* action is available. While this result may seem trivial, we highlight it because it shows why modeling agents using low-level policies does not generalize well in unseen situations.

The agent guided by the base model performs better than BC, occasionally crafting the correct target item. This is because it is able to use the appropriate planning operators to acquire planks from the trader, even if it never observed the modeled agent doing so. However, it cannot distinguish between internal states and thus crafts randomly. The full agent model infers the expert's internal-state from task 1 trajectories and can therefore simulate behavior conditioned on that state. This allows it to simulate the expert's behavior very accurately in the previously unobserved second task.

5 DISCUSSION AND CONCLUSION

In this work we introduced a novel method for incremental online learning of observed agent dispositions that might depend on (unobservable) agent-internal states. We showed that the model can learn online from a small number of observations and is able to model agent-internal states that differentiate behavior in similar environment states without assuming that the modeled agents are competent or goal-seeking. Our model also facilitates simulating agents with sufficient detail, while remaining useful in unseen, novel situations.

The learning algorithm of the proposed model ensures that all observed behaviors are explained after each update, as clusters are either created or merged but never destroyed. In the limit where θ_m is set to 0 and no clusters are merged, every propositional state is assigned its own cluster and the action distributions are updated accordingly. In general, the update algorithm ensures that all clusters in the model maintain at least θ_m distance from each other, capturing different dispositional states of the agent. Changes in established behavior are also incorporated without overwriting past experiences by hypothesizing internal states that explain the discrepancy. If two clusters converge to similar action distributions, including ones with internal states, they are merged. As a result, for a given setting of θ_m, the model maintains only a small set of internal states needed to explain the observed behaviors.

The proposed agent model does, however, have some limitations. For one, the merging threshold θ_m depends on both the modeled agent and the level of abstraction desired, as agents with high-entropy policies may require smaller values than agents with low stochasticity. Also, the agent model naively models action distributions without taking the low-level MDP state into account. This may lead to falsely considering a particular observation (in)sufficiently explained by the model, leading to few overly general or many overly specific dispositions. Furthermore, when modeling competent goal-seeking agents, it does not natively offer goal recognition, even though it can be used to hypothesize goals (including agent-internal states); however, it could be used in conjunction with goal recognition when goal states are not given or observable. Finally, the proposed model can also work in conjunction with existing agent-modeling paradigms, (e.g., BC, IRL) which can be used to model clusters' action distributions with goal-conditioned policies. This enables more accurate modeling in each cluster and relieves the requirement of handcrafted operators to simulate behavior.

Funding Acknowledgement. This work was in part funded by AFOSR grant FA9550-18-1-0465 and by ONR grant N00014-22-1-2206.

REFERENCES

[1] Leonardo Amado, Reuth Mirsky, and Felipe Meneguzzi. 2022. Goal recognition as reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9644–9651.

[2] Samin Yeasar Arnob. 2020. Off-Policy Adversarial Inverse Reinforcement Learning. *arXiv preprint arXiv:2005.01138* (2020).

[3] Saurabh Arora and Prashant Doshi. 2021. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence* 297 (2021), 103500.

[4] Okan Asik, Fatma Başak Aydemir, and Hüseyin Levent Akın. 2023. Decoupled Monte Carlo Tree Search for Cooperative Multi-Agent Planning. *Applied Sciences* 13, 3 (2023), 1936.

[5] Michael Bain and Claude Sammut. 1995. A Framework for Behavioural Cloning.. In *Machine Intelligence 15*. 103–129.

[6] Nate Blaylock, James Allen, et al. 2003. Corpus-based, statistical goal recognition. In *International Joint Conference on Artificial Intelligence*, Vol. 18. LAWRENCE ERLBAUM ASSOCIATES LTD, 1303–1308.

[7] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. 2021. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences* 11, 11 (2021), 4948.

[8] Christopher Geib and Pavan Kantharaju. 2018. Learning combinatory categorial grammars for plan recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[9] Michael Georgeff. 1988. Communication and interaction in multi-agent planning. In *Readings in distributed artificial intelligence*. Elsevier, 200–204.

[10] Shivam Goel, Yash Shukla, Vasanth Sarathy, Matthias Scheutz, and Jivko Sinapov. 2022. RAPid-Learn: A Framework for Learning to Recover for Handling Novelties in Open-World Environments. In *IEEE International Conference on Development and Learning, ICDL 2022, London, United Kingdom, September 12-15, 2022*. IEEE, 15–22. https://doi.org/10.1109/ICDL53763.2022.9962230

[11] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. 2016. Cooperative inverse reinforcement learning. *Advances in neural information processing systems* 29 (2016).

[12] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. *Advances in neural information processing systems* 29 (2016).

[13] KJ Joseph, Salman Khan, Fahad Shahbaz Khan, and Vineeth N Balasubramanian. 2021. Towards open world object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5830–5840.

[14] Pavan Kantharaju, Santiago Ontanón, and Christopher W Geib. 2019. Scaling up CCG-based plan recognition via Monte-Carlo tree search. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–8.

[15] Sarah Keren, Avigdor Gal, and Erez Karpas. 2014. Goal recognition design. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 24. 154–162.

[16] Panagiotis Lymperopoulos, Yukun Li, and Liping Liu. 2022. Exploiting Variable Correlation with Masked Modeling for Anomaly Detection in Time Series. In *NeurIPS 2022 Workshop on Robustness in Sequence Modeling*.

[17] Sriraam Natarajan, Gautam Kunapuli, Kshitij Judah, Prasad Tadepalli, Kristian Kersting, and Jude Shavlik. 2010. Multi-agent inverse reinforcement learning. In *2010 ninth international conference on machine learning and applications*. IEEE, 395–400.

[18] Gordon D Plotkin. 1970. A note on inductive generalization. *Machine intelligence* 5, 1 (1970), 153–163.

[19] Vasanth Sarathy, Daniel Kasenberg, Shivam Goel, Jivko Sinapov, and Matthias Scheutz. 2020. Spotter: Extending symbolic planning operators through targeted reinforcement learning. *arXiv preprint arXiv:2012.13037* (2020).

[20] Maayan Shvo. 2023. *Theory of Mind Reasoning in Explanation, Plan Recognition, and Assistance: Theory and Practice*. Ph.D. Dissertation. University of Toronto (Canada).

[21] Faraz Torabi, Garrett Warnell, and Peter Stone. 2018. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954* (2018).

[22] Alejandro Torreno, Eva Onaindia, Antonín Komenda, and Michal Štolba. 2017. Cooperative multi-agent planning: A survey. *ACM Computing Surveys (CSUR)* 50, 6 (2017), 1–32.

[23] Franz A Van-Horenbeke and Angelika Peer. 2021. Activity, plan, and goal recognition: A review. *Frontiers in Robotics and AI* 8 (2021), 643010.

[24] Pulkit Verma, Shashank Rao Marpally, and Siddharth Srivastava. 2021. Discovering User-Interpretable Capabilities of Black-Box Planning Agents. *arXiv preprint arXiv:2107.13668* (2021).

[25] Shao Zhifei and Er Meng Joo. 2012. A survey of inverse reinforcement learning techniques. *International Journal of Intelligent Computing and Cybernetics* 5, 3 (2012), 293–311.