Synthesizing Policies That Account For Human Execution Errors Caused By State Aliasing In Markov Decision Processes

Sriram Gopalakrishnan, Mudit Verma, Subbarao Kambhampati

Arizona State University
690 S Mill Ave
Tempe, AZ 85281
sgopal28@asu.edu

Abstract

When humans are given a policy to execute, there can be policy execution errors and deviations in execution if there is uncertainty in identifying a state. So an algorithm that computes a policy for a human to execute ought to consider these effects in its computations. An optimal MDP policy that is poorly executed (because of a human agent) maybe much worse than another policy that is executed with fewer errors. In this paper, we consider the problems of erroneous execution and execution delay when computing policies for a human agent that would act in a setting modeled by a Markov Decision Process (MDP). We present a framework to model the likelihood of policy execution errors and likelihood of non-policy actions like inaction (delays) due to state uncertainty. This is followed by a hill climbing algorithm to search for good policies that account for these errors. We then use the best policy found by hill climbing with a branch and bound algorithm to find the optimal policy. We show experimental results in a Gridworld domain and analyze the performance of the two algorithms. We also present human studies that verify if our assumptions on policy execution by humans under state-aliasing are reasonable.

1 Introduction

Markov Decision Processes (MDPs) have been used extensively to model settings in many applications((Boucherie and Van Dijk 2017),(Hu and Yue 2007),(White 1993)) but when the agent that has to act in such a scenario is a human, the optimal policy can be prone to errors and delays in execution. Our cognitive and perceptual limitations can result in mistakes. These mistakes can result in the human confusing similar states (state aliasing) and executing the wrong action for the current state. We may also take longer to execute an optimal policy since it could require more cognitive effort to discern between similar states correctly; this would be necessary when the policy for those similar states are different. A different, seemingly less-optimal policy that is easier for the human to execute faithfully and quickly can result in more accrued rewards. This would mean that it is actually superior to the policy computed without accounting for the effects on the human.

There is precedent for preferring simpler to execute policies from the medical literature; one example of this is the Apgar score(AmericanAcademyOfPediatrics 2006). It is a policy that relies on a simple scoring method to determine what action to take with newborn babies.

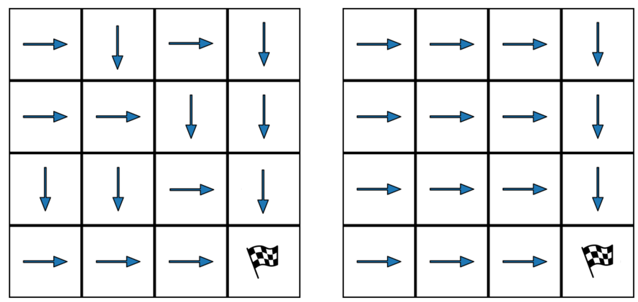

Figure 1: Policies with identical value in the original MDP, but can have different values after accounting for state aliasing; the policy on the right with similar actions across nearby (similar states) would have a higher value

The Apgar scoring procedure is a perceptual procedure to determine the state; think of it as classifying the state. In this work, we assume the perceptual or state-classification procedure is set. Due to human mistakes, and the limitations of the procedure, there can be state misidentification (misclassification) resulting in State Aliasing (mistaking one state for another). Our objective is to compute a policy that accounts for the likelihood of these errors and its effects; "account" here means how it affects the policy value (rewards accrued). There are two effects of state-aliasing we consider, that can affect the human's execution of the policy, viz. erroneous policy execution, and delay in execution. The erroneous policy execution occurs because the human misclassified the state. The delay in execution arises because of state uncertainty; if the human thinks it could be one of two possible states, *and* the policy is different in these states, then the human, in-effect, doesn't know which action to take. So they might try to resolve this uncertainty, since their objective is to properly execute the policy or just pause (delay action) out of uncertainty before acting. This response to uncertainty entails taking a "non-policy" action since it is not directly prescribed in the policy. If the prescribed policy-actions were the same across the set of possible states in the human mind, then we assume the human would act right away.

One can model the setting of human misclassifying states, as the state emitting observations which are also states. Then learning a policy (controller) as a function of the observations is just learning a deterministic reactive (memoryless) controller for a POMDP, and there is much prior work on this ((Littman 1994),(Meuleau et al. 2013)).However, none of the prior methods for learning such controllers handle the case where an action is a direct function of the policy; this the case with the non-policy action in our problem setting which happens due to the combination of state uncertainty and policy across those states. A more detailed comparison with existing literature and methods is done in the related work section, which is kept near the end of this paper in order to better compare against our method.

In this paper, we formally define the problem of computing a policy for a Human-Agent State-Aliased MDP (HASA-MDP). We describe how the human agent is approximated, and how our model of the agent helps compute the two effects of state-aliasing on human policy execution; erroneous execution and likelihood of non-policy action. This is followed by a description of a Hill Climbing algorithm to locally search the space of policies and find good policies that account for state aliasing. We also present a branch and bound algorithm to find optimal policies at the cost of increased computation time. We then show experimental results for our approach on a Gridworld domain, and in the supplemental we share results for a warehouse-worker domain. We also present the results of a human study which compares the execution of two policies, one that considers state-aliasing and one that does not; this is to show how our assumptions translate to real-world behavior.

1.1 Illustrative Examples Of State Aliasing Effects

To help understand and emphasize the value of this problem, as well as the ideas in this paper, we use two examples of two qualitatively different domains; a warehouse-worker example, and a gridworld example. In the warehouse-worker setting, the human agent's job is to put a product into the best fit box for shipping to the customer. The states are small, medium, and large, and correspond to the box size needed. The state classification procedure that the human is trained on involves comparing the product to a set of prototypes for small products, medium products, and large products. Needless to say, no set of prototypes will completely cover the space of all possible products, and some products will lie near the classification boundary between states. So the worker might misclassify and try to put a medium sized product into a small box. They may also delay acting because they are uncertain as to what box to put a product into. This is what we capture as a non-policy action. A non-policy action is an action unrelated to the policy to be executed. During this non-policy action, one can think of the human as dealing with their uncertainty. The non-policy action may just be a delay/inaction for 1 step ,they may check the features of a state, step through the classification procedure again, or even ask for help. The specific dynamics of the non-policy action is a domain model decision.

The other example is a gridworld example. Let's consider the example of a simple 4x4 grid illustrated in Figure 1. The reward is a single reward obtained at the bottom right. There are two policies with the same value in every state for the underlying MDP. However, if we consider that the human agent who is acting, doesn't know their exact position in each grid with certainty–state aliasing across nearby grids–then the policy on the right would be a lot better for the human to execute and accrue the reward faster. This is as opposed to the human possibly stopping execution (inaction, non-policy step) in each state because they are uncertain what state they are in, *and* the possible states require different actions.

Similarly, in the warehouse-worker example, if the loss in reward is negligible for always using the largest box (in comparison to the reward gained in completing an order), then that policy can be executed perfectly, and quickly. This would lead to more rewards accrued over time.

2 Problem Definition

The problem of generating policies for a Human-Agent State-Aliased MDP (HASA-MDP) is defined by the tuple $\langle S, A, T, r, \gamma, p_i, p_c, p_u, \psi_\varnothing \rangle$. Each of the terms are defined as follows:

- S is the set of states in the domain;
- A is the set of actions including $a_\varnothing$ which is a state reidentification action
- $T : S \times A \times S \to [0, 1]$ is the transition function that outputs the likelihood of transition from one state to a successor state after an action. This includes the non-policy action ($a_\varnothing$) dynamics.
- $r : S \times A \to \mathbf{R}$ is the reward function. This includes the reward associated to reidentification actions.
- γ is the discount factor
- $p_i : S \to [0, 1]$ is the probability of a state being the initial state
- $p_c : S \times S \to [0, 1]$ gives the likelihood of classifying(identifying) one state as another; $p(\hat{s}|s^*)$ where s^* is the true state, and $\hat{s}$ is the best guess of the state that the human agent thinks it is
- $p_u : S \times S \to [0, 1]$ gives the likelihood of uncertainty, i.e. probability of the human agent considering two states for the current state; $p(\{\hat{s_i}, s_j\}|s^*)$, where $\{\hat{s_i}, s_j\}$ is the pair of possible states for the true state with $\hat{s_i}$ being the best guess. We discuss this more in the human model section.
- $\psi_\varnothing$ is a scaling-factor that affects the probability of the non-policy action being taken when the agent is uncertain about the right policy action (which inturn is because of state uncertainty). It can be thought of as a reflection of pressure to act, or patience of the agent.

The objective in the HASA-MDP problem is to output a deterministic policy ($\pi_d : S \to A$) that optimizes for policy value (equation 3) after accounting for state aliasing effects determined by $p_c(.)$ and $p_u(.)$ that we will shortly formalize.

In this problem, we consider that if the agent has uncertainty over the current state, and if the policy conflicts between the states that the agent considers, then the human takes the non-policy action to try to resolve uncertainty. The likelihood of the non-policy action being taken is a function of the policy over states since the goal of the agent is to choose the right action, not to detect the state. The non-policy action is not an explicit part of the action space that is directly selected when optimizing the policy. This is expressed in Equation 1. The non-policy action in the domain could translate to many possibilities: (for example) calling a supervisor (oracle) for the right action; checking state features; repeating the classification procedure; or just inaction/delay step. The specific dynamics of the non-policy action is domain dependent. The point is that it is a function of both the human's state uncertainty and the policy over those states.

Because of state-aliasing, any deterministic policy – when actually executed – will become a stochastic policy. The first effect, is the likelihood of taking a non-policy action in a state:

$$\pi_{sa}(s, a_{\varnothing}, \pi_d) = \psi_{\varnothing}(s)\times \sum_{i\in\{1\ldots|S|\}} \sum_{j\in\{1\ldots|S|\}} p_u(\{\hat{s}_i, s_j\}|s) \times 1[\pi_d(s_j) \neq \pi_d(\hat{s}_i)] \tag{1}$$

where π_d is the deterministic policy given to the human agent, and $\pi_{sa} : S \times A \rightarrow [0, 1]$ is the stochastic policy after accounting for state aliasing. The second effect is on the likelihood of taking a policy action, and is defined by:

$$\pi_{sa}(s, a, \pi_d) = (1 - \pi_{sa}(s, a_{\varnothing}, \pi_d))\times \sum_{s_i\in S} p_c(\hat{s}_i|s) * 1[\pi_d(s_i) = a] \tag{2}$$

The overall value of the *determinisitic* policy π_d given to the human is defined as:

$$V_{sa}(\pi_d) = \sum_{s\in S} p_i(s) * V_{\pi_{sa}}(s) \tag{3}$$

This means we compute the policy value as per π_{sa} and weight the state values by the initial state likelihood.

3 Human Agent Model For Action Likelihoods

The human-agent model for our problem is defined by $\langle p_c, p_u, \psi_{\varnothing}\rangle$. What we really want is the probabilities of actions given a state and deterministic policy. We cannot get these probabilities directly since we do not know the policy before hand, and getting these likelihoods for all possible policies can be infeasible. So instead we use the probabilities from state classification and translate them to action likelihoods as per equations 1 and 2.

In this work, we assume that the likelihoods in p_c and p_u are given. In our supplemental material, we present an empirical process to approximate the probabilities in the human model by testing the human agent's inference on state instances. One would collect tuples of $(\hat{s}_i, s_j)$ for each state, where $\hat{s}_i$ is the human's best guess and s_j is the other most probable state in the human's mind. These represent the mental states after classification. We then normalize the counts to get p_u and p_c; for computing p_c one would normalize using the best guess counts. These data points, and consequently p_c and p_u, help capture the kind of state confusion or errors a person makes. p_u can be thought of as a distribution over the mental states of a human-agent in the state. In this presentation, we limit the number of alternate possible states in the human's mind to 1 for succinctness (can be the same state if confident). Our method naturally extends to cover the case that the human was uncertain over sets of 3 or more, and of varying sizes; this is detailed in the supplemental. In our model of the human behavior, we consider that if the policy matches across the set of uncertain states, then the human agent would take that policy action since the real objective is to choose the policy action, not get the state correct.

The last piece of the human model is the scaling-factor $\psi_{\varnothing}(s^*)$. This represents how likely the human is to act even if uncertain, and can be state specific. This scaling-factor is needed as we do not expect a person to just get stuck in a state and never act unless they are completely certain. They may act regardless of uncertainty because it maybe impossible to have absolute certainty for all state instances with a given classification procedure. One can also think of this scaling-factor as a reflection of the pressure to act on the human agent, or the patience of the agent to resolve uncertainty; when this scaling-factor is lower, then there is less probability of a non-policy action. One can either set a heuristic value to this scaling-factor, or empirically compute it; we relegate an empirical approach to estimating the scaling-factor to supplemental material.

The aforementioned probabilities in the human model could also be estimated via a plausible approximate model when available. For example, it could be computed using the number of shared features between states and normalizing some function of the counts. An approximate set of probabilities can helpful/sufficient to start with if empirical data is not available, especially during initial analysis of a domain. Let's consider the grid world setting; we could use conservative probabilities obtained from normalizing a function of L1 distances; the intuition is that states physically very close together might be confusable with each other, but this confusion likelihood drops very quickly with distance. Such a model can be a conservative approximation of state aliasing that can help compute better policies.

4 Policy Computation Algorithm for HASA-MDP

Finding an optimal solution to the HASA-MDP problem is at least as challenging as computing a reactive/memoryless controller for a POMDP, which is what our problem reduces to if one ignores the non-policy action (set $\psi_{\varnothing} = 0$). Computing a reactive controller has been shown to be NP-hard ((Littman 1994)); we discuss this more in the related literature section. To handle this computational complexity, we present two algorithms; a hill-climbing algorithm for computing good albeit suboptimal policies quickly, and a branch and bound algorithm for computing the optimal policy at higher computational cost.

4.1 State-Aliased Policy Improvement (SAPI)

We call our hill climbing approach State-Aliased Policy Improvement (SAPI) because it takes the greedy best step to change the policy while accounting for state-aliasing effects. In this we start with a random deterministic policy (π_d), and compute the corresponding stochastic policy after state aliasing (π_{sa} as per equations 1 and 2), followed by it's value as per equation 3. Then for every policy change in $S \times A$, we compute the value as just mentioned, and select the best-valued action to change the policy. This is repeated until no better changes can be made. Each step's computational complexity is $O(|S|^4|A|)$ since each step tests a number of changes equal to $|S||A|$, and the value of a fixed policy can be computed in $O(|S|^3)$ by computing the state transition likelihoods for that policy and using the following closed form computation (this is the standard equation for value computation in a Markov Reward Process (Ibe 2013)).

$$\vec{v_s} = (I - \gamma * P_{ss'})^{-1} * \vec{r_s} \quad (4)$$

The total time complexity for SAPI will naturally be problem specific; the number of improvement steps will depend on the initial point and the possible improvements in the domain. An alternate approach is to randomly iterate through states, and greedily take the best action for that state alone. Each steps's computation reduces to $O(|S|^3|A|)$, but as one might expect, we observed this resulted in worse results than taking the globally-optimal greedy step (best action across all states).

4.2 HASA Branch-And-Bound Policy Search

SAPI is effective in quickly finding a good policy, but if one wanted the optimal policy, the following branch and bound approach can be used. This branch-and-bound searches in policy space by choosing an action for each state (each level) in the policy search tree. We assume the reader is familiar with the basics of branch and bound (Brusco, Stahl et al. 2005). At any given point in the policy search, only a partial policy is defined. We need a lowerbound, and an upperbound to determine if the node in the search tree should be expanded. We use the lowerbound output by SAPI. This idea of using hill climbing for the lowerbound was used in (Littman 1994) as well.

We still need an upperbound that accounts for the non-policy action. To compute this, we use an MDP relaxation of the problem for a given partial policy by assuming full state observability for the remainder of the undefined policy space. We call this a "Partially-Controlled MDP" (PC-MDP). This is similar to the bound employed in (Meuleau et al. 2013) except theirs cannot account for the non-policy action. Additionally, their method doubles the state space for reactive controller computation which ours does not. We compare this further in the related work section. The proof for the upperbound is in the supplemental under the branch and bound section.

For each step in the branch and bound, we need to run value iteration on the PC-MDP. In our approach, we stop after a certain number of iterations; we set this to 1000 which was adequate for our experiments. We then take an upperbound for each state's value computed as $v_k(s) + \frac{\epsilon * \gamma}{1-\gamma}$(Chapter 17 (Russell and Norvig 2021)) where ϵ here is $||v_k - v_{k-1}||$. Alternatively, one could set a target error ϵ and determine the number of iterations needed as $\lceil \frac{log(2R_{max}/(\epsilon(1-\gamma))}{log(1/\gamma} \rceil$ (Russell and Norvig 2021).

The size of the policy search tree is unfortunately $|A|^{|S|}$, but a good bound and ordering the states intelligently to make pivotal decisions early can greatly prune the tree, as the experiments will show.

The score(heuristic) we use to order the states is:

$$score(s) = p_i(s) \times \sum_{s' in S} p_c(s|s') \times \max_{a \in A(s')} r(s', a) \quad (5)$$

This score looks at how likely a state is to be the initial state (it's value matters more). It also considers how much state-aliasing this state causes (how confusing is it).We look at how probable it is that others states are confused with it, and scale that by the max reward possible in the other state. The scaling is because we want to prioritize states that can have an influence on the policy in high reward states.

5 Experiments and Results

We tested our algorithm on two domains; gridworld and warehouse-worker. We relegate the experimental results of warehouse-worker to the supplemental in favor of more experiments and analysis of gridworld experiments where we can clearly vary the degree of confusion between states, the distribution of rewards, and study their effects on the effectiveness of our algorithm.

For SAPI, we repeated the search ten times for each setting and took the best result. All results can be consistently reproduced from our codebase since any variability (like randomness in initialization) in the program is controlled by a seed parameter. This is set to 0.

5.1 Experimental Setup

For the following experiments in Gridworld, we used 4×4 grids with varying properties to analyze and compare the performance between Hill Climbing and Branch and Bound. The actions for each state include moving up, down, left, and right. The goal is the bottom right square like in Figure 1, which is an absorbing state. The reward is 100 to reach it. All action transitions are stochastic with a 5% chance of the transition from a random action, and all action costs are 0 (only reaching the goal matters). An invalid action results in the agent staying in the same state with zero cost. The non-policy action also just stays in the same state but has a small negative cost of 0.1; we take this to reflect the annoyance of the human agent at not knowing which direction to go. Most importantly, the agent would get a reward of 100 for entering the goal state in the bottom-right corner.

As for the likelihoods of confusion and uncertain, we model the likelihood of confusing a grid state (position) with another grid state as determined by the L1 distance as defined in Equation 6.

$$p_c^(s|s') = \frac{1/(L1(s,s') + 1[s = s'])^m}{\sum_{s'' \in S} 1/(L1(s,s'') + 1[s = s''])^m} \quad (6)$$

where m is a scalar in $[1, \infty)$; this allows us to vary the amount of state aliasing between nearby states; when m is 1, the likelihood of a This results in neighboring grid states being much more likely to get confused with the current state than those further away. For our experiments we used m=5 to quickly decay the likelihood of confusing with states more than 1 step away. We add the $+1$ to avoid divide by zero.

The $p_u(.)$ probabilities were set by taking the average of the classification probabilities from each pair of states, computed as :

$$p_u(s_1, s_2|s*) = \frac{p(s1|s*) + p(s_2|s*)}{2} \quad (7)$$

This would mean that if two states had a high probability of being classified in the state $s*$, then we model the likelihood that the agent would be uncertain between those two states as higher too. Lastly, we kept the scaling-ratio $\psi_\emptyset$ to 1.0 for our experiments.

We present results of a 5x5 grid, because while SAPI (hill-climbing) can easily handle larger sizes of grids (we have tested up to 20x20 = 400 states), branch and bound speed drops very quickly as the policy space grows as $|A|^{|S|}$; 4^{25} is over 1.23×10^{27} possible policies. The bound helps with pruning this space, and the effectiveness of the bound is affected by the distribution of rewards, discount factors, and other parameters (as we will see). Additionally, we must keep in mind that each node requires solving an MDP by value iteration. We were able to study this in the 5×5 setting in a reasonable time (for branch and bound) and so used that size to compare the two algorithms. We will also show the number of nodes opened by branch and bound to show that the upper bound used was a helpful bound.

To study the behavior of the algorithms in this gridworld setting, we focused on two parameters: (1) The discount factor γ, whose baseline value is 0.7; (2) A "reward noise range" parameter (RNR) to add random rewards to each of the actions in the grid and whose default value is 1. $RNR = 1$ would assign rewards to each action in the range $[-0.5, 0.5]$, i.e., uniformly distributed about 0. We varied these parameters to see how SAPI compares with branch and bound.

Our code was implemented in python, using pybnb library for branch and bound and PyTorch and NumPy for matrix operations. The experiments were run on a PC with Intel® Core™ i7-6700 CPU, running at 3.40GHz on Ubuntu 20.04 with 32 GB of memory. For each experiment, SAPI was run 30 times, and branch and bound were obviously only run once.

5.2 Results

The general trend with increasing discount factor is that the effectiveness of SAPI goes down, and tends to find more suboptimal policies. With increasing random noisy rewards

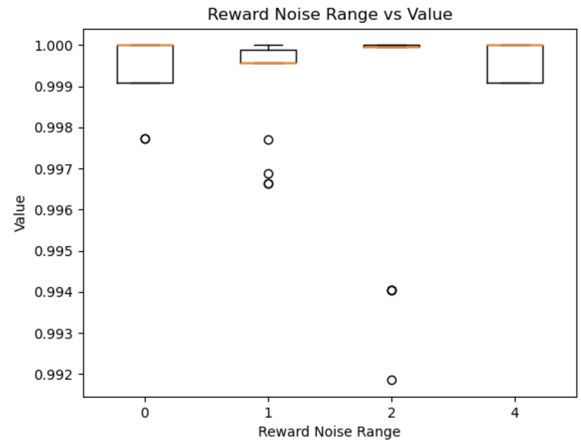

Figure 2: Box plot of SAPI policy values for 5x5 Grid with varying Reward Noise Range; values normalized by the policy value from Branch and Bound search

(RNR) a clear trend was not seen; we think this is because it may smooth the policy search space as much as complicate it with more local maxima if the rewards are uniformly random, which they are (in our experiments). Some of this trend can be seen in the number of nodes opened by branch and bound (Table 1). The nodes opened is higher for discount factor of 0.7 and 0.9. No clear trend is observable with the increase in RNR. The spike when RNR = 4 ($[-2, +2]$ uniform random reward added) could be because of the specific actions to which the change in reward was made, as well as it's relative values to other actions (as one might surmise). This effect is not as clear as increasing the discount factor.

We did notice that within 30 iterations, SAPI found an optimal policy in all settings, which is nice because SAPI scales much better than the branch-and-bound policy search. SAPI got stuck in local maxima more as the discount factor increase (Figure 3)

We interpret this as the discount factor increasing the coupling between states as rewards from distant states travel longer and with a larger magnitude. This makes the policy search harder.

Lastly, we share in Table 1 the number of nodes opened by our branch and bound search before finding the optimal solution. The number of nodes is significantly less than the policy space of 4^{25}, which is helpful because the cost of each node is high since we do value iteration on each node for the associated PC-MDP. This also shows that our upper bound was helpful in pruning the policy search space.

5.3 Human Studies

In addition to our computational experiments, we wanted to see if people executing a policy that accounts for state aliasing really translates to faster execution and fewer errors.

The user study asked participants to press the arrow key associated with the color of a large square displayed. The policy for which arrow key to press associated to color was permanently displayed on the right of where the large color

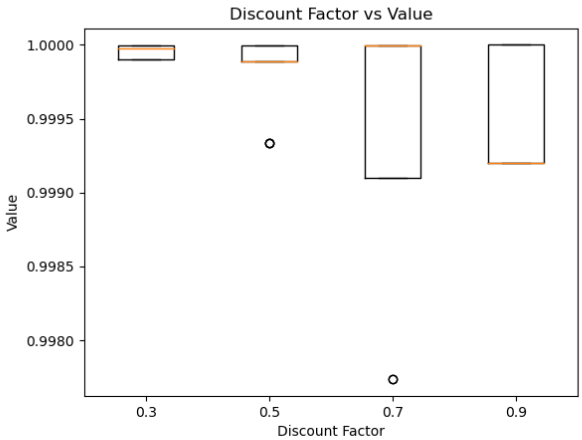

Figure 3: Box plot of SAPI policy values for 4x4 Grid with varying discount factor (γ); values normalized by the policy value from Branch and Bound search

square is displayed; this is so they do not have to memorize the policy and have quick access to it. As soon as they press the arrow, a new color would be displayed; the color was chosen from a set with uniform probability. The color here corresponds to the state, and the arrow keys are the actions.

In our human study, we tested the same user on two policies. One policy did account for state aliasing (policy A) , and one that did not (policy B) had different actions across similar states. This is shown in Figure 4. We hypothesized that for policy A, the number of actions correctly executed would be higher, and the error rate would be lower than policy B.

Note that some color states are intentionally visually similar to other color states to cause state-aliasing. The participants were filtered by their ability to distinguish between different colors so that they could execute both difficult and simple policies. The table 2 shows results for the 41 participants in this study.

We wanted to see if the number of actions executed was greater with the simpler policy; since less confusion likelihood should imply fewer delays. We are especially interested in the number of correct actions (throughput). We also wanted to check if the rate of errors was lower. Since data from the two settings of simple and difficult policies may have unequal variances, we used a Welch's t-test (one-tail) to evaluate the results. We used the implementation in the Scipy python library (Virtanen et al. 2020)

For the total number of actions executed by a participant, we can reject the null hypothesis that the number of actions executed is the same or lower with the simpler policy than the difficult policy; the one-tailed T-test gave a p-value of $< .0001$. For the second hypothesis, that a simpler policy yields a higher number of correctly executed actions, a one-tailed T-test gave a p-value of $< .0001$. So with a very low likelihood of error, we can say our hypothesis held good in this human study. The results are clearly significant at $p < 0.05$.

Discount Factor (S=0.05,RNR=0)		**Reward Noise Range (S=0.05, γ=0.7)**	
γ	#nodes	RNR	#nodes
0.3	125	0	141
0.5	125	±0.5	125
0.7	141	±1	113
0.9	141	±2	741

Table 1

We are also interested in the *rate* of errors when executing the simple versus difficult policies. We wanted to show that the rate of errors in a difficult policy is more than that of a simple policy. A one-tail T-test for the rate of errors (number of errors/total attempts) was significant only at a p-value of 0.065. So while we have good reason to believe this to be the case, we cannot say with confidence (p-value < 0.05) that the rate of errors is definitely lower. There might have been other factors affecting the rate of errors that we had not considered, such as the speed of execution of the simpler policy. One possible effect is that when the participants were acting very fast with the simpler policy, the likelihood of errors went up.

Overall, our human studies give support to the idea that policies which consider state aliasing and executed faster and more reliably, atleast in this setting with visually similar states.

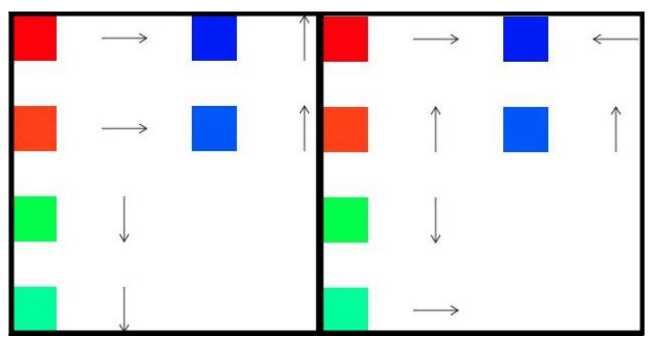

Figure 4: Policy A that accounts for state-aliasing (left), and Policy B that doesn't account for it(right) that were given to users to execute

	Policy A (left)(μ, σ)	Policy B(right) (μ, σ)
Correct Attempts	26.34, 5.62	17.95, 3.77
Total Attempts	28.12, 6.13	19.68, 3.82

Table 2: Mean and standard deviation for the number of correct actions and total actions executed for Policy A(which accounts for state aliasing) and Policy B (which doesn't account)

6 Related Work

The Apgar score (AmericanAcademyOfPediatrics 2006), as mentioned earlier, is an example that shows the importance of a simpler policy that can be reliably executed for consistent results. This idea of consider computational limits of humans was taken further in the work "Super Sparse Linear Integer Models (SLIM)" (Ustun and Rudin 2016) in which the authors build sparse linear models with emphasis on smaller integer weights because they make computation by humans easier and more reliable.

We are similarly motivated to account for our cognitive limitations and uncertainty in inference. We compute simpler policies for MDPs that seek to account for our cognitive limitations, and the errors we may make. Specifically, we consider how humans can misidentify states, and how it can be better (by policy value) to work with a policy that maps confusable states to the same action. (Lage et al. 2019) works with a similar assumption, albeit for policy summarization. They proposes an Imitation Learning (IL) based summary extraction that uses a Gaussian Random Field model to model how human's extrapolate a partially defined policy. They considered that people use the similarity between states for generalizing policy summaries to states that were not part of the summary. For one of the human studies in their work, they reported that 78% of their participants used state similarity based policy-summary reconstruction. In the context of our work, this translates to the human incorrectly choosing actions due to state similarity.

If we can afford to ignore the cost or effect of non-policy action arising from policy uncertainty, then we are left with just state misclassification errors. The problem then reduces to the problem of computing a reactive policy or a memoryless controller (Littman 1994) for a Partially Observable MDP (POMDP); this mapping is evident if we interpret the human's state inference as an observation emitted by the state. Computing such a controller has been shown to be NP-hard (Littman 1994) (Meuleau et al. 2013). While there have been improvements made on computing controllers based on observation history (history can be more than 1) (Meuleau et al. 2013)(Kumar and Zilberstein 2015), none handle the case where an action's likelihood is a direct, computed function of the policy, as opposed to just a mapping from observations to action. This is needed for our problem since uncertainty with respect to the action to take depends on the policy across states.

Additionally, our branch and bound algorithm reduces to computing a memoryless/reactive controller if the scaling-factor $\psi_{\varnothing}$ is set to 0. It is different to the branch and bound method presented in (Meuleau et al. 2013) as they take the cross product of a policy graph and the underlying MDP, which results in doubling the state space during policy search; we keep the original state space. It must be noted that their method can learn a controller for variable lengths of observation history, and ours is limited to the current state observation and also supports the non-policy action.

With respect to the state-aliasing phenomenon itself, this has been studied in the lens of POMDPs for agents with "active-perception" capabilities (Whitehead and Ballard 1991),(Tan 1991), (Whitehead and Lin 1995). In (Whitehead and Lin 1995) the authors call the problem as perceptual aliasing when an internal state maps to more than one external state due to limitations of the sensing process. Their approach of "Consistent Representation"(CR) pulls together the prior work on the perceptual aliasing problem. The common assumption across those works is that a consistent representation of the state that is Markovian can be built from the immediate environment with sensing actions. (Whitehead and Lin 1995) also considered "stored-state" architectures where history was incorporated for state inference. In all of these works, there are additional computation steps required in the policy to improve state detection. These approaches assume the agent has sufficient computational capacity, and consistently infers the correct subsequent state representation. We do not think one can expect this from humans. We instead rely on reducing the execution errors by allowing for incorrect sensing (incorrect state classification) by humans.

7 Conclusion and Future Work

In this paper, we describe the problems that can arise from state aliasing when humans execute a policy; these are execution errors and policy deviation/delays due to uncertainty. We formally define the problem of computing policies in HASA-MDPs and define how non-policy actions and policy execution errors can be incorporated into the policy computation. We discuss how the state classification and state uncertainty likelihood can be empirically inferred or approximated using a heuristic model for human agents. We also incorporate a scaling-factor to model how patient/impatient or under pressure an agent is to act. Given our formalization of the HASA-MDP problem, we present a modified policy iteration algorithm (SAPI) which is a hill-climbing search for policies that account for the effects of state aliasing. We also present an optimal branch-and-bound algorithm for the problem and compare it's performance with SAPI. Lastly, we conducted human studies to show how our assumptions translate to real-world behavior.

References

AmericanAcademyOfPediatrics. 2006. The Apgar score. *Advances in neonatal care: official journal of the National Association of Neonatal Nurses*, 6(4): 220.

Boucherie, R. J.; and Van Dijk, N. M. 2017. *Markov decision processes in practice*, volume 248. Springer.

Brusco, M. J.; Stahl, S.; et al. 2005. *Branch-and-bound applications in combinatorial data analysis*, volume 2. Springer.

Hu, Q.; and Yue, W. 2007. *Markov decision processes with their applications*, volume 14. Springer Science & Business Media.

Ibe, O. 2013. *Markov processes for stochastic modeling*. Newnes.

Kumar, A.; and Zilberstein, S. 2015. History-based controller design and optimization for partially observable MDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25.

Lage, I.; Lifschitz, D.; Doshi-Velez, F.; and Amir, O. 2019. Exploring computational user models for agent policy summarization. In *IJCAI: proceedings of the conference*, volume 28, 1401. NIH Public Access.

Littman, M. L. 1994. Memoryless policies: Theoretical limitations and practical results. *From animals to animats*, 3: 238–245.

Meuleau, N.; Kim, K.-E.; Kaelbling, L. P.; and Cassandra, A. R. 2013. Solving POMDPs by searching the space of finite policies. *arXiv preprint arXiv:1301.6720*.

Russell, S.; and Norvig, P. 2021. Artificial Intelligence: A Modern Approach, Global Edition 4th. *Foundations*, 19: 23.

Tan, M. 1991. *Cost-sensitive robot learning*. Ph.D. thesis, Carnegie Mellon University.

Ustun, B.; and Rudin, C. 2016. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 102(3): 349–391.

Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; van Mulbregt, P.; and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17: 261–272.

White, D. J. 1993. A survey of applications of Markov decision processes. *Journal of the operational research society*, 44(11): 1073–1096.

Whitehead, S. D.; and Ballard, D. H. 1991. Learning to perceive and act by trial and error. *Machine Learning*, 7(1): 45–83.

Whitehead, S. D.; and Lin, L.-J. 1995. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1-2): 271–306.

Supplemental Material

The following content is in the supplemental material

- Code Appendix
- HASA Branch And Bound Details
- Warehouse Worker Domain Experiments
- Computing Classification Likelihoods In The Warehouse Worker Example

Code Appendix

All source code for running the experiments is provided in the following github repository. Please see the “Readme.txt” file for instructions on how to run the code.

HASA Branch And Bound Details

7.1 Proof for Upperbound

Before we go into the details of the branch and bound proof, let us review the equations that define how the stochastic policy that is actually executed, is a function of the deterministic policy given , the classification probabilities, and uncertainty probabilities.

The likelihood of non-policy action is a function of the state as well as the policy defined over other states, as follows:

$$\pi_{sa}(s, a_\varnothing, \pi_d) = \psi_\varnothing(s)\times \sum_{i\in\{1...|S|\}}\sum_{j\in\{1...|S|\}} p_u(\{\hat{s}_i, s_j\}|, s) \times 1[\pi_d(s_j) \neq \pi_d(\hat{s}_i)] \tag{8}$$

The likelihood of policy actions are:

$$\pi_{sa}(s, a, \pi_d) = (1 - \pi_{sa}(s, a_\varnothing, \pi_d))\times \sum_{s_i\in S} p_c(\hat{s}_i|s) * 1[\pi_d(s_i) = a] \tag{9}$$

The value of a policy is defined as :

$$V_{sa}(\pi_d) = \sum_{s\in S} p_i(s) * V_{\pi_{sa}}(s) \tag{10}$$

The key idea in the proof is that we can get an upper-bound by considering the domain has perfect observability, the same assumption made in (Meuleau et al. 2013). During policy search, at any node after the first, the policy has been partially decided in the previous steps. This partial policy needs to be kept for it to be a useful bound. The key difference with the (Meuleau et al. 2013) is that we do not use a cross product between the policy graph and MDP (which doubles the state space for our probelm), *and* we need to account for the non-policy action. In order to do so we leverage the following lemma

Lemma: A partial stochastic policy whose action probabilities are the same or lower than those of another partial policy, can be completed and optimized to the same or better value as the optimization of the other policy

This is best understood starting with an illustration in Figure 5

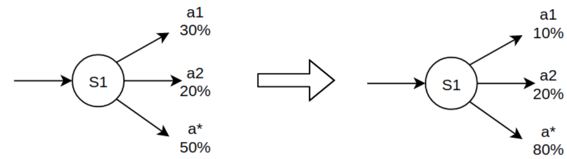

Figure 5: Example of a partial stochastic policy that is reduced to another policy such that the probability of all actions are the same or lower

Note that it is not enough that the probability of choosing the optimized action is higher. Even if the overall sum of probabilities for the fixed partial policy goes down, if any one action has increased probability, then this lemma is not guaranteed to hold. The simplest counter example is a single state MDP with two actions that loop back (this is like a multi-arm bandit with discounted future rewards). The first action has a reward of 10, and the second has a reward of 1. If the total percentage of the partially defined policy decreased, but the probability of the least reward action increases as in Figure 6, then we cannot assign more percentage of the optimal action than the original partial policy. So the state value in this case would be lower.

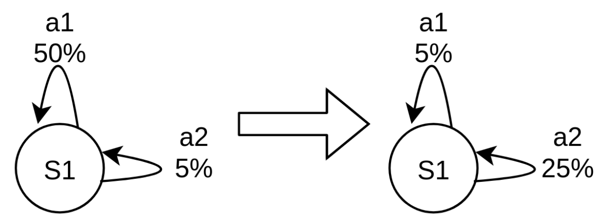

Figure 6: Example of reduced total probabily that will not result in a higher state value after optimization; the lower reward probability increases

The proof for the lemma is rather simple and is as follows. We first define a Partially Controllable MDP (PC-MDP) as an MDP in which the states can have some of their actions fixed with a certain probability (hence partially controllable). This PC-MDP can be converted into an equivalent MDP by simply updating the action transitions and rewards to be the expected sum of each action and the other actions in the partial policy. The probabilities are the from the partially defined stochastic policy. For this MDP any deterministic policy (fixed action choice on the derived actions) will have a value greater than or equal to any stochastic policy in the derived action space. This is a general statement for MDPs.

Now let’s consider the case that the probability of actions decrease in the partially defined policy. An example of this is illustrated in Figure 7.

If we optimize the MDP associated to the original set of probabilities (before reducing), that results in a stochastic policy in the MDP, lets call this policy A. This same policy distribution as policy A for any state can be chosen in case

Figure 7: Example of reduced total probability in a PC-MDP; a* is the optimizable action

when all the probabilities are reduced. So the optimal policy value in the reduced probability case will be atleast as good as the value of policy A. So the value of a state with the optimal policy after the probabilities of the partial policy are reduced will always be greater than or equal to the optimal policy value for the same state with the original probabilities of the partial policy.

Proof For Upperbound In Branch And Bound

As we have seen in equation 2, a determinisitic policy translates to a stochastic policy after state aliasing. Using the same functions, a partial deterministic policy can be mapped to a partial stochastic policy. If we could do so, then we could relax the setting to a fully observable MDP and compute the of the PC-MDP as an upperbound.

The problem with what we have said so far is that we *cannot* know the partial probabilities of the actions (including $a_{\varnothing}$) as it these probabilities are affected by the rest of the policy. So instead, we compute a lower bound on the probabilities of the actions in a state. The lowerbound for delay probability ($\pi_{sa}(s, a_{\varnothing}, \pi_d)$) is computed by considering that all undefined state policies will *not* conflict with the state-policies defined thus far, i.e. $\pi_d(s_i) == \pi_d(s_j)$ where s_i represents the state's whose policy is defined, and s_j is an undefined state. Then the probability is computed as per equation 1. As for the policy action probabilities, we use misclassification probabilities from p_c and multiply it with one minus the maximum delay probability possible. The maximum possible delay is computed by considering the worst possible case of policy conflicts for the probability sum to be maximum (for each state). So equation 2 becomes

$$\pi_{sa}(s, a, \pi_d) = (1 - max\_delay(s, \pi_d)) \times \sum_{s_i \in S} p_c(\hat{s_i}|s) * 1[\pi_d(s_i) = a] \tag{11}$$

Now we have probabilities for actions that are less than or equal to the probabilities for the same actions for any possible policy completion. With these probabilities we can get the value of the associated "PC-MDP" by value iteration. As we showed in the lemma, the value associated with this policy–where all the predefined action probabilities are lower– will be greater than or equal to any policy completion on the original partial policy.

We also empirically verified in our experiments that the bound converges from above, and converges to the true policy value of any completion of the partial policy. To ensure that it is an upperbound, we use the value error bounds and add to the value iteration estimate to get a true upperbound. The value used after k-iterations is $v_k(s) + \frac{\epsilon * \gamma}{1-\gamma}$(Chapter 17 (Russell and Norvig 2021)) where ϵ here is $||v_k - v_{k-1}||$. Alternatively, one could set a target error ϵ and determine the number of iterations needed as $\lceil \frac{log(2R_{max}/(\epsilon(1-\gamma))}{log(1/\gamma} \rceil$ (Russell and Norvig 2021).

Warehouse Worker Domain Experiments

7.2 Warehouse-Worker Domain Description

In our Warehouse-Worker domain, a worker stands at the end of a conveyor belt on which customer orders are sent. The customer orders comprises of a group of products. Each order (group of products) needs to be put into a small, medium, or large box. Additionally, the worker has to decide if bubble wrap is necessary for those products or not.

The states in this domain is what kind of an order a group of products is, and there is an associated correct action for each order type. The state and action sets are defined by the cartesian product of the set of box sizes $\{small, medium, large\}$, and if bubble wrap is needed $\{wrap, no\_wrap\}$. For example a group of items with glass items could be a small order that requires a small box with bubblewrap, $small \times wrap$. When a worker sees an order, it is not always apparent what box size is needed. For some orders, they may mistake a small order for a medium sized one or vice versa. Additionally, due to the diversity of products, the worker has no idea which products actually need bubble wrap or not. For example there maybe tempered (hardened) glass products that do not need bubble wrap, but the worker might not know this.

In our conceptualization of this domain, after a worker goes through basic training in the warehouse, the worker is evaluated by the supervisor to evaluate how they classify orders, and the kind of confusion/errors they make; this helps compute p_c and p_u. Based on this, a policy is developed for the worker by considering the company's average estimates for the money made per order when using different types of packaging (reward specification), and the likelihood of order types.

The generic transitions are defined as follows (specific values will follow later). If state is a large-order, and the agent takes the action to package as a medium-order, then the agent gets a reward for packaging some of the items, and the state transitions 100% of the time to a medium-order state, since some of the items are still not packaged. The same is done for medium orders when the action taken is to package as a small-order. If the action taken is for a size larger than the order, then the order is completed but with a lower reward than if the correct sized package was used. Once an order is complete, the state transitions to the next order based on the probability of an order-size as per the company's statistics.

Thus far we have only talked of package sizes. The other dimension is whether an order needs bubblewrap (soft packaging material) or not. If an order is large and needs bubblewrap, then packaging it as medium without bubble wrap gives a lower reward and transitions to "medium-size with-bubble wrap" state 50% of the time, or medium-size (no bubble wrap) state for the other 50% of the time. This is to reflect that the user might have packaged the items that required bubble wrap already. The analogous transition happens with medium-sized orders that require bubble wrap.

As for the reward. The right action–right size and addition of bubble wrap if needed– gives a reward of 1.0. All other suboptimal actions gives a lower reward. How much lower is randomly determined by the Reward Noise Range (RNR) parameter. We vary this parameter between $0.1, 0.2, 0.3, 0.5$. Based on this parameter the reward is randomly reduced by upto this amount for each action.

The other part of the experimental setup is the classification likelihoods which are shown in Table 3. The likelihood of a worker confusing any small order with any medium sized order or vice versa is about 16%, and is the same for misclassifying any medium with any large order. The likelihood of confusing a small order with a large order is less than 1%. The likelihood of the worker correctly determining if bubblewrap is needed is 50% across all order sizes; there are so many products that their accuracy for determining if bubble wrap is needed is random.

	l	l × w	m	m × w	s	s × w
l	32.68%	32.68%	12.50%	12.50%	0.98%	0.98%
l × w	32.68%	32.68%	12.50%	12.50%	0.98%	0.98%
m	16.34%	16.34%	25.00%	25.00%	16.34%	16.34%
m × w	16.34%	16.34%	25.00%	25.00%	16.34%	16.34%
s	0.98%	0.98%	12.50%	12.50%	32.68%	32.68%
s × w	0.98%	0.98%	12.50%	12.50%	32.68%	32.68%

Table 3: Classification Likelihood Matrix (p_c) for Warehouse-Worker Domain, where (s,m,l) stands for (small,medium,large) and "w" means bubblewrap needed.

The probability of uncertainty $p_u(.)$ is computed using the same method as for the gridworld experiments. The $p_u(.)$ probabilities were set by taking the average of the classification probabilities from each pair of states, computed as :

$$p_u(s_1, s_2|s*) = \frac{p(s1|s*) + p(s_2|s*)}{2} \tag{12}$$

This would mean that if two states had a high probability of being classified in the state $s*$, then we model the likelihood that the agent would be uncertain between those two states as higher too. Lastly, we kept the scaling-ratio $\psi_\varnothing$ to 1.0 for our experiments. The non-policy action is inaction as it was in gridworld, and the cost is -0.1.

In our experiments we vary the reward noise range (RNR) parameter as $0.1, 0.2, 0.3, 0.5$ for one set of experiments, these are presented in Figure 8. We also vary the discount factor as $0.3, 0.5, 0.7, 0.9$. These results are presented in 9. The default RNR is 0.0 when it is not being varied, and the default discount factor is 0.7. We also set the probability of taking a random action is 5% to add more stochasticity into the domain. For each setting we run SAPI 30 times and present the distribution of the policy value (Equation 3) normalized by the value of the policy returned by branch and bound. Since the branch and bound policy value is optimal , we expect the range to be [0,1].

7.3 Warehouse Worker Experimental Results

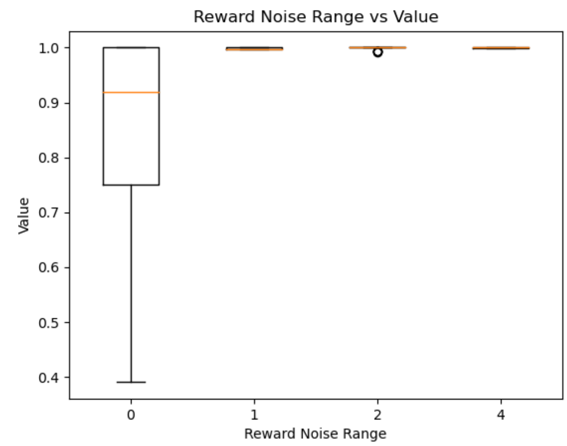

Figure 8: Box plot of SAPI policy values for warehouse-worker domain with varying Reward Noise Range; values normalized by the policy value from Branch and Bound search

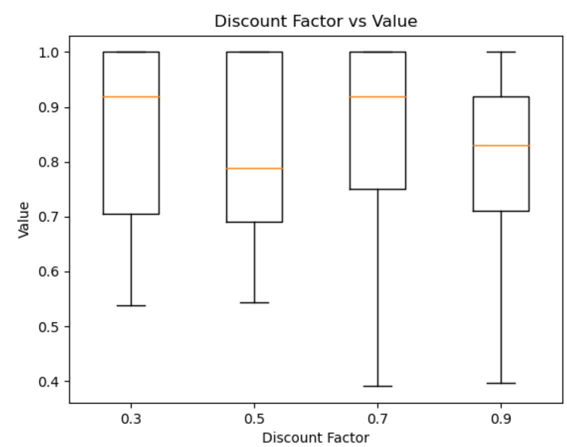

Figure 9: Box plot of SAPI policy values for warehouse-worker domain with varying discount factor (γ); values normalized by the policy value from Branch and Bound search

In all the experiments, as we wanted, the branch and bound policy has the highest value. The SAPI policy search found the optimal policy value atleast once out of 30 iterations, which is what we hoped. The variance in policy value found by SAPI is a lot more than in the grid world experiments. The trend of incresing variance in policy value for SAPI as discount factor increases is not as evident in this domain as it was in gridworld. This maybe because of fewer states.

The surprising trend is that as the RNR parameter was increased (less reward for actions) SAPI seemed to find the optimal policy a lot more consistently. It is easy to see why this holds in the limit when all the rewards are zero and the only cost is from inaction (non-policy action). Then the optimal policy is any policy that has the same action across all states, of which there are many and easy to find. We were surprised that the effect of the non-policy action was pronounced even with smaller values of RNR. Indeed the optimal policies found were just to put all state orders in the large-box with bubblewrap, which fits out intuition; if the difference in rewards is negligible, then use the action that applies to all states and avoids non-policy actions.

Computing Classification Likelihoods In The Warehouse Worker Example

In this section we present a process to compute the probabilities in p_c and p_u for the Warehouse-worker setting as an example for how such probabilities could be obtained empirically. After the company trains the worker on the classification procedure (which was using prototypes), the employees (human agents) are tested on different instances of small, medium and large products. For each instance, we record what their best guess is, as well as any other guesses that they considered with non-trivial probability, i.e. caused them uncertainty. The human agent's response (for example) might be "I'm sure (or think) this is a medium sized product, but I did consider that it could be a small-sized product too". They may still be uncertain when they make their decision. Their other guesses can be null, for cases that were obvious to the human. These other guesses helps capture the uncertainty of the person, and is used to build the distribution of possible mental states. This is what causes the non-policy action, which could be actions like repeating the classification procedure, checking features, calling the supervisor, or just inaction in this warehouse worker setting.

Each data point collected is a tuple $(\hat{s}_i, s_j, s_{j+1}, s_{j+2}, s_{j+3}...)$ where $\hat{s}_i$ was their best guess, and the other s_j states are the alternate states that were guessed. From this we can compute p_c by normalizing the counts for the best guess per state, and p_u by normalizing the counts of each tuple as a whole; each tuple is an event (of uncertainty in the human's mind). Since the number of uncertain states is more than two, we have to generalize the Equation 1 to Equation 13. This equation is simply the extension of Equation 1 by considering that the agent will be uncertain about the right action if *any* of the set of uncertain state's policies conflict with the most likely state's policy; previously there was only one alternate state, now there is a set of them.

$$\pi_{sa}(s, a_\varnothing, \pi_d) = \psi_\varnothing(s) \times \sum_{i \in \{1...|S|\}} \sum_{J \in U^s_{\hat{s}_i}} p_u(\{\hat{s}_i\} \cup {}_J U^s_{\hat{s}_i} | s) \times 1[\bigvee_{s_j \in U_j} \pi_d(s_j) \neq \pi_d(\hat{s}_i)] \tag{13}$$

where the terms shared in 1 are identically defined, and $U^s_{\hat{s}_i}$ represent the set of uncertain state sets in the human agents mind associated with true state s and with the best guess as $\hat{s}_i$. The term ${}_J U^s_{\hat{s}_i}$ returns one of those possible sets indexed by J.

The reason we use the counts of the best guess to compute the posterior likelihood for state classification (p_c) is to get the correct policy action likelihood for our model. This is can be understood with a simple example. Let's consider that the collected data of human inference for a state has 90/100 instances with "State 1" as the best guess. Then we expect that if the human infers one of these cases during acting, and the policy matches with the other uncertain state, then the action they take should correspond to the policy of "State 1". We expect this should happen 90% of the time when they do not take a non-policy action; this is what Equation 2 describes. Note that in this work, we do not consider how the likelihood will evolve with successive non-policy actions; If one wanted to model the likelihood as changing when the non-policy action stays in the same state, then additional states are needed to represent the change in mental state of the agent. Our framework will still apply, but the computational cost grows quickly. Additionally, one would have to collect data for the probabilities after 1 non-policy action, or after 2, and so forth.

7.4 Empirically estimating the scaling factor

The last piece of the model is the scaling-factor $\psi_\varnothing(s^*)$. This represents how likely the human is to act even if uncertain, and can be state specific (as we do in this work). This scaling-factor is needed as we do not expect a person to just get stuck in a state and never act unless they are completely certain. They may act regardless of uncertainty because it maybe impossible to have absolute certainty for all state instances with a given classification procedure. One can also think of this scaling-factor as a reflection of the pressure to act on the human agent since when this scaling-factor is lower, then there is less probability of non-policy action; zero probability of non-policy action when the scaling-factor is 0.

To determine the scaling-factor from data, we would compute the expected number of repeat guesses the human took for a given state, and stopped while still not completely certain (as per their feedback). We would use intentionally difficult state instances helps to collect this data. With this data, first we compute a probability ($p_\varnothing$) such that the expected number of non-policy actions from the following equation (Equation 14) matches the number obtained empirically.

$$E[|non-policy|] = 0 \times (1-p_\varnothing) + 1 \times p_\varnothing \times (1-p_\varnothing) + ... + n \times p_\varnothing^n \times (1-p_\varnothing) + ... \tag{14}$$

If we take out a $p_\varnothing$ term as common, and substitute $p = (1-p_\varnothing)$, we see that this is the form of a standard geometric random variable, and we know the expected value for such a series is $1/p$.

$$E[|non-policy|] = p_\varnothing \times (1 \times p + 2 \times p \times (1-p) + 3 \times p \times (1-p)^2 + ...) = \frac{p_\varnothing}{p} = \frac{p_\varnothing}{1-p_\varnothing} \tag{15}$$

Once we get the target $p_{\varnothing}$, this is also our desired scaling-factor $\psi_{\varnothing}$ since we want 100% probability (of non-policy action) to be scaled down to $p_{\varnothing}$. Recall that this is in order to model that the human may choose to act, even when they cannot remove uncertainty.