Zero-Shot Assistance in Novel Decision Problems

Sebastiaan De Peuter
Department of Computer Science
Aalto University
Helsinki, Finland
sebastiaan.depeuter@aalto.fi

Samuel Kaski
Department of Computer Science
Aalto University
Helsinki, Finland
Department of Computer Science
University of Manchester
Manchester, UK
samuel.kaski@aalto.fi

Abstract

We consider the problem of creating assistants that can help agents solve novel sequential decision problems, assuming the agent is not able to specify the reward function explicitly to the assistant. Instead of acting in place of the agent as in current approaches, we give the assistant an advisory role and keep the agent in the loop as the main decision maker. The difficulty is that we must account for potential biases of the agent which may cause it to seemingly irrationally reject advice. To do this we introduce a novel formalization of assistance that models these biases, allowing the assistant to infer and adapt to them. We then introduce a new method for planning the assistant's actions which can scale to large decision making problems. We show experimentally that our approach adapts to these agent biases, and results in higher cumulative reward for the agent than automation-based alternatives. Lastly, we show that an approach combining advice and automation outperforms advice alone at the cost of losing some safety guarantees.

1 Introduction

In this paper we consider the problem of assisting agents in tackling sequential decision problems which they have never encountered before. Human decision makers are routinely faced with this problem. Take for example engineering design [29], where one looks to find or construct the best possible design within a space of designs that are feasible. Every design problem is new: each time an architect builds a house it is for different clients. Although the problem is novel to the agent, we can assume that it already knows how to solve it in principle, though not optimally. An architect does not need to re-learn architecture when designing a new house, but can apply their general knowledge and experience to this new design problem.

These design problems can be thought of as single-episode decision problems: they consist of a sequence of decisions, each changing or elaborating a design in some way. Once a satisfactory design has been found, the episode terminates. The decisions are driven by a goal, which can be encoded as a reward function, known to the agent. This goal is usually tacit and complex, meaning that the agent is unable to describe it explicitly.

We seek to create *assistants*[1] which can assist agents in solving these types of decision problems. The goal for the assistant is to increase the quality of the agent's decisions, measured by cumulative reward, relative to the agent's effort. But there are two things the assistant does not know a-priori about the agent: the agent's reward function, and any biases that may cause the agent to deviate from

[1]Though the assistant is an agent, to avoid confusion we will always refer to it as *assistant* and will only use *agent* to refer to the agent being assisted

Preprint. Under review.

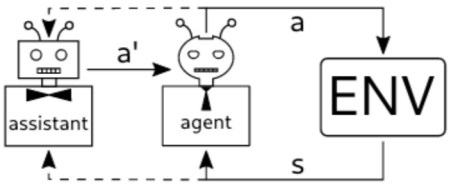


Figure 1: In *zero-shot assistance* an assistant helps an agent solve a problem without initially knowing the agent's reward function. We propose an assistant which helps the agent primarily by advising it, leaving the agent in direct control of its environment. In every time step the assistant gives new advice a', appropriate for the current state, based on its inference of the agent's reward function and potential biases. When acting the agent incorporates the advice into its own decision making. The assistant observes both the action a taken by the agent and the new state of the environment s, and uses this to infer the agent's reward function and biases.

optimal behaviour. Although the agent knows its reward function, it has never solved its problem before – ruling out inferring the reward function from prior observations – and is not able to provide an explicit description of the reward function. Similarly, biases are not something an agent is aware of, and thus are not something it can communicate to the assistant. The assistant is forced to infer both during the episode, therefore we call this *zero-shot assistance*. Zero-shot assistance is closely related to zero-shot coordination [20], though in the latter the reward function is known to all participants.

We introduce *AI-Advised Decision-making* (AIAD). In AIAD an assistant helps an agent primarily by giving advice, while the agent remains responsible for taking actions in the environment. Figure 1 shows the interaction between the agent and the assistant. The advice is based on a reward function that is inferred from the agent's behaviour. For simplicity we will focus on advice of the type "have you considered doing a", where a is an action, though AIAD readily generalizes to other forms of advice. We see two fundamental advantages in having an assistant that advises. First, taking context-appropriate advice into account takes comparatively little effort while it can be really helpful. Bad advice can be swiftly rejected. Second, it keeps the agent firmly in control and able to reject advice that would have a negative impact. This is a minimum requirement in applications where safety is a concern. More generally though, it allows the assistant to adopt a high-risk high-reward strategy for its advice, by relying on the fact that the agent will reject advice that is bad.

Choosing advice to give to a biased agent requires the assistant to account for those biases. Biases can cause apparently irrational behaviour, including the rejection of useful advice. We consider these biases to include innate limitations or constraints within the agent's decision making process, incorrect problem understanding, or limited knowledge. Incorrect problem understanding, for example, has been shown to cause humans to reject advice which rationally is in their interest [11]. Prior work on various types of assistance has been able to incorporate agent biases that were known *a priori* [15, 19, 32]. In zero-shot assistance, however, biases must be inferred online. To address this we model the uncertainty over biases explicitly, allowing the agent to maintain beliefs over which biases an agent has, and to incorporate present and future beliefs into its planning.

Contributions In this paper we formalize the assistant's problem of advising agents with unknown reward function and biases as a decision problem. We propose a planning algorithm, a variant of Monte Carlo Tree search (MCTS), for finding the assistant's policy. To evaluate the practicality of AI-advised decision-making we introduce an idealized but realistic design problem: planning a day trip. A popular baseline approach for reducing agent effort and improving decision making is to automate, by leaving the decision making entirely to an assistant. When no reward function is available, prior work has proposed to first elicit the reward function [25, 36], and then automate. In simulation experiments we show that **(1)** AIAD significantly outperforms these automation-based baselines for the same amount of agent effort. We also show that **(2)** an assistant which infers and accounts for agent biases outperforms one that does not. Lastly **(3)** we show that a hybrid approach – in which AIAD is extended to both automate and advise – outperforms the baselines and standard AIAD when assisting agents with biases, but loses some of the safety advantages of standard AIAD.

2 Related work

Learning reward functions from others The main alternative to our proposed approach of advising agents is to take decisions in their place, i.e. automation. For this, the reward function must be known. Thus, before automating one needs to elicit or learn the reward function from the agent. Inverse Reinforcement Learning (IRL) proposes to learn a reward function directly from observing the agent act [25, 1, 28, 3]. In preference-based elicitation [36, 9, 5], the agent is asked which of two trajectories or individual decisions it prefers. The agent is assumed to prefer the trajectory with the highest reward. An alternative is to ask the agent for direct feedback on a single trajectory of decisions [22, 35]. A subset of this literature has looked specifically into the feasibility and utility of learning both agents' reward functions and biases [12, 13]. Chan et al. [8] found that biases can make agents' behaviour more informative of their reward function, and that incorrectly modeling biases can result in poor reward inference. Armstrong and Mindermann [2], however, show that jointly identifying biases and reward from observations is not always possible. Shah et al. [31] investigate under what assumptions biases can be learnt purely from data.

Where applicable, inference and automation happen in two distinct phases in these works; the elicitation process is not informed by the immediate needs of automation. In an assistance method like ours, both happen at the same time, allowing the assistant to reduce its uncertainty with regards to the agent's biases and reward where it matters for the decisions it needs to make [32]. We note also that under reward uncertainty, an automating policy must necessarily be more risk-averse than an assistant that gives advice, as the automating policy cannot rely on the agent to prevent it from making bad decisions.

Human-AI collaboration Our work fits within a larger body of approaches which consider collaboration between an assistant and an agent to solve a common problem. Dimitrakakis et al. [10] consider a setting in which an assistant acts autonomously but can be overridden by an agent at a cost. Çelikok et al. [7] consider a similar problem in partially observable environments. Both, however, assume the assistant already knows the reward function. Others have considered collaboration when the assistant does not know the agent's reward function. Shared autonomy [21, 30] considers a setting in which the agent gives commands to the assistant, which then acts in the environment. As the assistant does not necessarily follow the commands directly, but uses them to infer the agent's reward function which it then maximizes, we consider this an automating approach. In Cooperative Inverse Reinforcement Learning [19] an assistant and agent jointly solve a problem. The assistant uses IRL to learn the agent's reward function. Fern et al. [15] has proposed assistants which assist agents (not necessarily through advice) in decision problems; Shah et al. [32] proposed similar assistants for partially observable settings. These last three works are similar to ours but assume that, except for the reward function, everything that determines the agent's policy – including any potential biases – is known *a priori*. Unlike our method, these methods not support the online inference of biases needed for zero-shot assistance.

3 Problem setup

We consider an agent solving a decision problem which can be modeled as an infinite-horizon MDP $E = \langle \mathcal{S}, \mathcal{A}, T, \mathcal{R}_\omega, \gamma, p_{0,s} \rangle$. Here $\mathcal{S}$ is a set of states and $\mathcal{A}$ is the set of actions available to the agent. At time step i the transition function $T(s_{i+1} \mid s_i, a_i)$ defines a distribution of potential next states s_{i+1} given that the agent has taken action a_i in current state s_i. $\mathcal{R}_\omega(s_i, a_i, s_{i+1})$ is the reward function. It defines the instantaneous reward for taking action a_i in state s_i and ending up in s_{i+1}. $\gamma \in (0, 1]$ is the discounting rate. The agent's objective is to maximize its expected discounted cumulative reward $\mathbb{E}\left[\sum_{i=0}^{\infty} r_i \gamma^i \mid T, p_{0,s}\right]$ where r_i is the reward it achieved at time step i. Finally, $p_{0,s}$ is the start state distribution: $s_0 \sim p_{0,s}$.

When assisting an agent we will assume that we know certain things about that agent's problem E. In line with prior work [1] we assume that we know $\mathcal{S}$, $\mathcal{A}$, T, γ and $p_{0,s}$. Though we do not know $\mathcal{R}_\omega$, we do have access to its parametric function class $\mathcal{R} = \{\mathcal{R}_{\omega'}\}_{\omega' \in \Omega}$. Note that this assumption is not particularly restrictive; $\mathcal{R}$ could be the space of all reward functions.

4 AI-advised decision-making (AIAD)

We now formalize AI-Advised Decision-making. As shown in Figure 1, the agent acts in environment E based on advice from the assistant. The goal of the assistant is to maximize the cumulative discounted reward obtained by the agent through this advice. We define this from the assistant's point of view as a decision problem with reward function $\mathcal{R}_\omega(s_i, a_i, s_{i+1})$ and as actions the advice it can give.

For the assistant to be able to plan, we will assume we have an *agent model* $\hat{\pi}(a \mid s, a'; \theta, \omega)$ available, a model of the agent's fixed policy upon receiving advice a'. This could be an expert-created model based on appropriate assumptions, or could have been learned based on observed agent behaviour on similar problems. It depends on two sets of unobserved parameters: $\omega \in \Omega$ and $\theta \in \Theta$. We defined Ω earlier as the parameter space of the reward function. Θ is the parameter space for the possible biases the agent has. We call θ the bias parameters.

4.1 Advice as a decision problem

We define the assistant's decision problem as a generalized hidden parameter MDP (GHP-MDP) $\mathcal{M}$ [27]. A GHP-MDP is an MDP in which both the transition and reward function are parameterized but where the true values of those parameters are not observed. In our definition these parameters are ω and θ, the two unobserved parameters which determine an agent's reward function and biases. We define $\mathcal{M}$ such that the instance $\mathcal{M}_{\omega,\theta}$ defines the problem of assisting an agent with reward parameters ω and bias parameters θ.

We define $\mathcal{M} = \langle \mathcal{S}, \Omega, \Theta, \mathcal{A}', \mathcal{A}, \mathcal{T}, \hat{\pi}, \mathcal{R}, \gamma, p_{0,s}, p_{0,\omega}, p_{0,\theta} \rangle$. Here $\mathcal{S}, \mathcal{A}, \gamma$, and $p_{0,s}$ are the state space, agent action space, discounting rate and initial state distribution from the agent's problem E. $\mathcal{R}$ and $\hat{\pi}(a \mid s, a'; \theta, \omega)$ are as defined earlier. The assistant's actions $\mathcal{A}'$ constitute advice that can be given to the agent. Ω and Θ are the parameter spaces for the reward function and biases, and $p_{0,\omega}$ and $p_{0,\theta}$ are prior distributions over them. Lastly, $\mathcal{T}$ is a collection of transition functions $\mathcal{T}_{\omega,\theta}$ for all possible values of $\omega \in \Omega$ and $\theta \in \Theta$. It encodes the interaction between the assistant and the agent and between the agent and the environment from Figure 1.

When the assistant gives advice $a'_i \in \mathcal{A}'$ to the agent, the agent is free to choose which action $a_i \in \mathcal{A}$ to take in E. It is this action taken by the agent that leads to a new state, according to the transition function of E. Thus, the assistant only indirectly influences the change of state, by using advice to induce a different policy from the agent. The agent model $\hat{\pi}$ predicts which policy will be induced by advice. Thus, for given reward and bias parameters ω and θ, the transition function is

$$\mathcal{T}_{\omega,\theta}(s_{i+1} \mid s_i, a'_i) = \sum_{a_i \in \mathcal{A}} \hat{\pi}(a_i \mid s_i, a'_i; \theta, \omega) T(s_{i+1} \mid s_i, a_i)$$

Because the true values of θ and ω are not known, we can think of $\mathcal{M}$ as defining a space of MDPs. The challenge in planning over $\mathcal{M}$ is that planning must happen without knowing which MDP in this space the assistant is truly operating in, i.e. what kind of agent the assistant is advising. We can, however, maintain beliefs over θ and ω based on the transitions we observe. For every transition (s_i, a'_i, s_{i+1}) we observe we can calculate the likelihood of that transition under the various possible parameter values in $\Omega \times \Theta$ to update our posterior belief distributions over the parameters. In other words, by observing the agent's decisions in response to advice we can maintain beliefs about the agent's biases and reward function.

4.2 Root sampling for GHP-MDPs

Finding an optimal policy over $\mathcal{M}$ involves not only planning on a belief distribution over MDPs, but also accounting for how that distribution will change as we act. With every action we take, we observe a new transition which will change our beliefs over Ω and Θ. However, not every action is equally informative: advice which gets wildly different reactions from different types of agents will be more useful for determining what type of agent we are assisting than advice to which all agents react in the same way. Planning must consider both the expected long-term reward of actions, and their informativeness towards the unknown parameters.

Perez et al. [27] oringinally proposed to use model predictive control to plan over GHP-MDPs, but that does not account for action informativeness. Thompson sampling has been used to plan in

parameterized MDPs, where only the transition function is parameterized [17]. Though this could be extended to GHP-MDPs, Thompson sampling results in policies with high variance. Fern et al. [15] proposed sparse sampling for a similar problem, but that would not scale with large action spaces $\mathcal{A}'$.

Instead, we propose a modification of *Bayes-adaptive Monte Carlo Planning* (BAMCP) [18]. BAMCP is based on MCTS [6] and enables planning over MDPs where the transition function is not known, and needs to be inferred from transition observations. The advantage of BAMCP is that it efficiently maintains beliefs over transition functions. By using the tree as a particle filter it can implicitly track posterior belief distributions at all future states in the tree [18]. This allows it to incorporate the future information value of actions into its value estimates. We extend this algorithm from operating on beliefs over transition functions to joint beliefs over transition and reward functions, i.e. beliefs over Θ and Ω. We call this new variant *Generalized Hidden Parameter Monte Carlo Planning* (GHPMCP).

We give a short overview of the algorithm here, and refer the reader to appendix A for a detailed explanation. Like any MCTS algorithm, in every planning iteration GHPMCP simulates an MDP down the tree following a UCT policy. The main difference is that the simulated MDP is resampled for every iteration. Before an iteration starts, parameters θ, ω are sampled from p_θ, p_ω. $\mathcal{M}_{\omega,\theta}$ is then simulated down the tree. The Q-function estimates along the path are updated using $\mathcal{M}_{\omega,\theta}$'s specific reward function R_ω. Here lies the difference to BAMCP, which – because it only considers uncertainty over the transition function – uses the same fixed reward function R in every iteration.

5 An agent model for assistance

We now introduce a general-purpose agent model, applicable to any decision problem E as defined in section 3. We have developed this agent model to be a good starting point for most use cases. It is based on established and grounded theories of human decision making. It is also consistent with how RL agent policies are often implemented. We use instances of it in our experiments here, but stress that our proposed method does not require this agent model specifically.

The proposed agent model is built on the following choice rule:

$$p(a|u) = \frac{p(a)\exp\left(\beta u(a)\right)}{\sum_{\hat{a}\in\mathcal{A}} p(\hat{a})\exp\left(\beta u(\hat{a})\right)} \tag{1}$$

where $a \in \mathcal{A}$ is an action, u is a function that assigns a utility to every action, $p(a)$ is a prior distribution over actions, and β is a temperature parameter. β allows us to interpolate between fully rational choices ($\beta = \infty$) and fully random choices ($\beta = 0$). This choice rule has repeatedly proven to be a compelling model of human cognition. It is known under various names including "Boltzmann rational model" and "Luce-Sheppard model". Theoretical work has proposed it as a result of a bounded rational view of human cognition with information processing costs [26, 16]. It is also frequently used in practice to model human behaviour [23, 4, 34, 9, 5]. Surprisingly, this rule is also a popular choice of model in RL for representing agent policies [33]. There it is known as a softmax policy.

The agent model $\hat{\pi}(a \mid s, a'; \theta, \omega)$ consists of a sequence of choices made according to this choice rule. It is based on the idea that an agent will only settle for the assistant's action if it cannot find a better one itself. As utility function we use the Q-function of the current state $u(a) = \hat{Q}(s, a; \theta, \omega)$. This Q-function could be derived directly from the agent's problem E, or from some derivative of it $\hat{E}$ in case we want to model an agent with an incorrect or limited view of the world. Some biases can therefore be modeled as part of $\hat{E}$. $\hat{Q}$ then depends on ω to allow us to model different reward functions, and on θ so that we can use the bias parameters as parameters of the problem model $\hat{E}$.

Under our model the agent starts by choosing the best action it can think of. This is a stochastic choice which we represent as a random variable A_1 defined over the actions. The distribution of A_1 results from a straightforward application of equation (1):

$$p\left(A_1 = a\right) = \frac{p(a)\exp\left(\beta_1 \hat{Q}(a)\right)}{\sum_{\hat{a}\in\mathcal{A}} p(\hat{a})\exp\left(\beta_1 \hat{Q}(\hat{a})\right)} \tag{2}$$

As the state and parameters are constant within this context, we have used the shorthand $\hat{Q}(a) := \hat{Q}(s, a; \theta, \omega)$ and $p(A_1 = a) := p(A_1 = a|s; \theta, \omega)$. We will continue to drop the conditioning variables s, θ and ω for the rest of this section.

Next the agent chooses whether to switch to the assistant's recommended action a' or to stick to the action a it has chosen. The probability of switching from a to a' (denoted $a \to a'$) is

$$p(a \to a') = \frac{\exp\left(\beta_2 \left(\hat{Q}(a') - \hat{Q}(a)\right)\right)}{1 + \exp\left(\beta_2 \left(\hat{Q}(a') - \hat{Q}(a)\right)\right)} \tag{3}$$

This probability is the result of applying equation 1 to the binary choice of switching or not with a uniform prior. The utility for both choices is the gain in Q-value realised: $\hat{Q}(a') - \hat{Q}(a)$ in the case of switching and 0 otherwise. As this choice is easier than the choice in equation 2 we use a different temperature parameter β_2 here. Because the originally chosen action A_1 and the switch to a' are stochastic, we represent the agent's choice of action after considering a' by A_2, a random variable with distribution

$$p(A_2 = a|a') = \begin{cases} [1 - p(a \to a')]p(A_1 = a) & \text{if } a \neq a' \\ [1 - p(a' \to a')]p(A_1 = a') + \sum_{a'' \in \mathcal{A}} p(a'' \to a')p(A_1 = a'') & \text{if } a = a' \end{cases}$$

Some problems, such as design problems, have a special NOOP action ϕ which allows the agent to choose to do nothing. We model the agent as considering a switch to this action as its last step. The idea is that if the agent has arrived at an action that would have negative effect, it should choose to do nothing instead. This ca be though of as the agent recommending ϕ to itself, analogous to how the assistant had recommended a'. The probability of switching $p(a \to \phi)$ is therefore the same as equation (3) with a' replaced by ϕ. The policy produced by our agent model is then the result of combining A_2 with this switch, analogous to how we combined A_1 with the switch to a':

$$\hat{\pi}(a|a') = \begin{cases} [1 - p(a \to \phi)]p(A_2 = a) & \text{if } a \neq \phi \\ [1 - p(\phi \to \phi)]p(A_2 = \phi) + \sum_{a' \in \mathcal{A}} p(a' \to \phi)p(A_2 = a') & \text{if } a = \phi \end{cases}$$

with the shorthand $\hat{\pi}(a|a') := \hat{\pi}(a \mid s, a'; \theta, \omega)$. If a NOOP action is not applicable for E, this last step can be skipped by defining $\hat{\pi}(a|a') = p(A_2 = a|a')$.

6 Experiments

We will present the results from three simulation experiments on an idealized design problem: day trip design.[2] We compare our proposed method to an unassisted agent and two automation-based baselines from the reward learning literature. We test the following hypotheses: **H1** Agents assisted through advice, as in AIAD, achieve higher cumulative reward than those assisted by automation-based approaches, like the baselines, even when unknown biases are present. **H2** Assistants that infer and account for agent biases help the agent achieve higher cumulative reward than ones that do not. **H3** Extending AIAD to allow it to automate as well as advise can yield higher cumulative reward than standard AIAD.

AI-advised trip design We run our experiments on a day trip design problem; an idealized but otherwise realistic instance of a design problem. The agent is given 100 points of interest (POI) and must plan a day trip by choosing some subset of the POIs to visit in a day. Its goal is to choose a subset that it would maximally enjoy visiting. Every POI has a location, visit duration, admission cost, and belongs to a number of topics. There are 20 topics in total. Given a set of POIs the agent wants to visit, a trip itinerary is automatically produced which starts at a home location and visits all chosen POIs in an order that minimizes travel distance. The agent is not allowed to spend more than about 12 hours on the trip, including travel time.

The formalization of this problem as an MDP is as follows. The state space spans all subsets of the 100 POIs. In addition to a NOOP action there are 100 actions, each associated with a specific POI, allowing the agent to add that POI to the current trip or to remove it depending on whether it is already part of it or not. When the agent is in a state corresponding to a trip that takes more than 12 hours, it is only allowed to remove POIs. Figure 2 visualizes this process.

[2] Our experiment code is available in the supplementary.

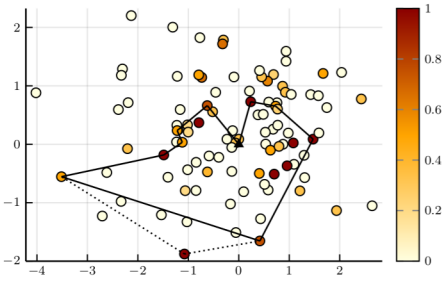


Figure 2: In day trip design one looks to choose a set of interesting POIs to visit in a day. This figure shows the locations (axes are coordinates) of a set of available POIs. Their shading correspond to an example agent's interest (lighter color signifies lower interest). An example itinerary is shown, starting at the home location (triangle at the center). The dashed line shows a potential change to the itinerary from additionally visiting the bottom-most POI.

The agent tries to maximize an objective function $f_\omega(s)$ which captures how much it likes a trip s. Therefore, we define its reward function as the improvement in objective value from one time step to the next: $R_\omega(s_i, a_i, s_{i+1}) = f_\omega(s_{i+1}) - f_\omega(s_i)$. The objective function f_ω is a product of two scores. The first measures the agent's enjoyment of visiting the selected POIs, and is based on how many of the POIs' topics it is interested in. The second score is determined by the total admission cost for visiting all POIs and decreases as cost increases. The parameters ω therefore consist of 20 binary parameters representing the agent's interest in the 20 topics, and a parameter determining what total admission cost the agent is willing to accept. We describe f_ω in greater detail in appendix B.1.

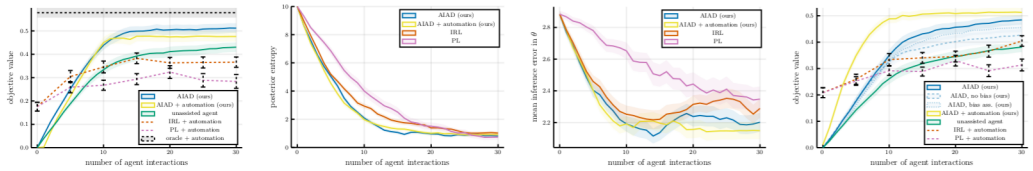
We use the agent model we introduced in section 5. The problem $\hat{E}$ on which the agent plans differs in one key aspect from the true problem. To predict its reward following a change to its trip, the agent must predict what the associated itinerary will be. But finding this itinerary is a hard traveling salesman problem. Therefore we model the agent as using a visual heuristic used by humans to solve this problem instead [24]. Whenever a POI is added the agent inserts it into the itinerary according to a maximum-angle heuristic. The dotted line in Figure 2 shows the bottom POI being added to the itinerary according to this heuristic. The Q-values used in the agent model are calculated using depth-limited planning on $\hat{E}$. We leave β_1 as a free parameter of this model, but fix $\beta_2 = 10\beta_1$.

Experiment set-up In these experiments we will pair various assistance methods with a simulated agent solving the day trip design problem. We consider three baselines: **(1) unassisted agent** To create an unassisted agent we modify our agent model by removing the switch to a recommended action encoded in eq. (3) ($p(a \to a') = 0 \; \forall a \in \mathcal{A}$). **(2) IRL + automation**. This is an IRL-based approach following prior work in learning rewards from biased agents. It observes N timesteps of a the unassisted agent acting without assistance. It then infers both θ and ω from the observations, using the same agent model but without knowledge of the parameters, and automates based on the inferred reward function. **(3) PL + automation**. This approach is based on preference learning (PL). The agent is presented with N queries that are comparisons between two day trips (i.e. states). These queries are selected based on their expected information gain. We create an agent model that chooses between the two day trips according to the choice rule from eq. (1) with $u(s) = f_\omega(s)$. This model is both used to simulate the agent and to infer the reward function from the agent's responses. We then automate based on the inferred reward function.

To automate we must find an automation policy that is optimal in E (as automation operates directly in the environment) but over the posterior beliefs inferred through IRL or PL instead of the agent's R_ω (which is not observed). To do this we turn E into a GHP-MDP with θ as unobserved parameter and parameterized reward function $\mathcal{R}$. The prior over ω is the posterior inferred by IRL or PL. We plan over this GHP-MDP using GHPMCP. After the N observations or queries this automation policy runs for the rest of the episode without further interaction with the agent.

Every experiment run lasts a single episode. Posterior beliefs in all implementations are maintained using a weighted particle filter. For every run we sample new agent model parameters and a new set of POIs. The baselines and AIAD are each applied once per run. Our quantity of interest is the objective value achieved[3] as a function of the number of agent interactions N. As interactions we count either actions by the agent (in AIAD, unassisted agent, and IRL) and queries (PL). The PL and IRL baselines are evaluated at 5, 10 15, 20, 25 and 30 interactions, while the other methods are evaluated on a continuous range of N from 1 to 30. We use a two-sided paired Wilcoxon signed rank

[3]Due to our definition of the reward function this is equal to the undiscounted total reward for the episode.



(a) (b) (c) (d)

Figure 3: Results for the day trip design experiments. The shading and error bars in all plots show the standard error of the mean. **(a)** Mean objective value achieved by unbiased agents as a function of different numbers of interactions. **oracle + automation** here shows the value of day trips produced by an optimizer with access to the agent's reward function. **(b)** Posterior entropy over number of interactions for an unbiased agent. **(c)** Mean posterior error between inferred reward parameters and the true reward parameters for an unbiased agent. **(d)** Mean objective value achieved by agents with potential anchoring biases at different numbers of interactions.

test to compare the objective value achieved with each assistance method. Additional details and results from the experiments are available in the appendices.

6.1 AIAD for day trip design without biases

In this experiment we apply AIAD and the baselines to the agent model introduced above, providing evidence towards **H1**. We ran this experiment 50 times. Figure 3a shows that AIAD significantly ($p < 0.01$) outperforms all baselines after 9 interactions. Note that because we start from an empty trip, AIAD needs a minimum of interactions to produce a complete design. This minimum could be reduced by allowing AIAD to recommend a sequence of actions instead of a single one, or by starting with a reasonable partial design. The IRL baseline at 0 interactions shows that relatively good designs can be found under the prior over reward parameters. These designs could be used as a starting point.

AIAD achieves lower uncertainty with regards to the agent parameters (figure 3b), and lower mean loss in its inference of the reward (figure 3c), compared to the automation-based baselines. However, based on the difference in inference error between AIAD and IRL, we do not believe that this is the main reason why AIAD performs better. We see a large part of the explanation two specific advantages of assistance through advice. The first is that through the agent model, the assistant can intentionally rely on the agent to remove the negative effects of risky advice. This allows it to adopt advice that is good for most agents, as opposed to advice that is okay for all. The second advantage is the mitigation of misspecification problems. In virtually all runs of this experiment there was a difference on at least one topic between the simulated agent's topic interests and the closest matching particle in the particle filter. This causes reward misspecification, evidence of which we see for all methods in the low uncertainty and relatively high inference error in the latter part of the episode. The fact that bad advice resulting from misspecification is quickly rejected by the agent mitigates this issue for AIAD. Automating methods, however, do not have this safety net.

6.2 AIAD for day trip design with an anchoring bias

We now repeat the experiment above but change the set-up so that some agents have an anchoring bias. This bias, which is typical in human designers, causes the agent to resist large changes to its design. In our implementation, agents with this bias will refuse to consider adding any POI that is more than 500 meters away from their current itinerary. Half of the simulated agents in the 75 runs of this experiment had this bias. Whether an agent was biased was not known to the assistance methods.

Regarding **H1** (advice works better than automation or no assistance) we observe that agents assisted by AIAD achieved significantly ($p < 0.01$) higher objective value than those assisted by the other baselines after 12 interactions (figure 3d). Comparing these results to the previous experiment we see that the anchoring bias has reduced the improvement of AIAD over the automation-based baselines. This is expected as the presence of this bias will limit the POIs AIAD can recommend, but does not similarly impact automation. We will show in the next experiment how this can be alleviated.

To test **H2** (an assistant that infers and accounts for these biases is more helpful) we additionally test two other versions of AIAD that assume, rather than infer, this bias. One (**AIAD no bias**) assumes that the anchoring bias is not present, and thus never accounts for it, while the other (**AIAD bias ass.**) assumes it is present, and thus always accounts for it. Both were significantly outperformed ($p < 0.05$) by standard AIAD after 11 interactions. Note that there is initially no benefit in inferring the anchoring bias. Interesting POIs that are close-by are preferred over ones that are far away regardless, so properly accounting for the bias's presence only becomes important once these interesting close-by POIs have been added.

6.3 AIAD that can automate

In the previous experiment we saw that the presence of biases adversely affected the performance of AIAD. Meanwhile, automation – which does not rely on the agent to change the design – was not affected. Extending AIAD by allowing it to automate and advise should allow it to sidestep these biases and perform better (**H3**). The agent model is instrumental here, as it allows the assistant to estimate whether automation or the safer option of advice is the best choice given its current posterior beliefs. To test this we modify AIAD by equipping $\mathcal{M}$ with additional actions that directly edit the design. These actions have transition function $\mathcal{T}_{\omega,\theta}(s_{i+1} \mid s_i, a'_i) = T(s_{i+1} \mid s_i, a'_i)$. We then applied this new variant, which we call **AIAD + automation**, to the previous two experiments. The results are shown in respectively figure 3a and figure 3d. Note that because we are interested in agent effort, these graphs do not record direct edit actions taken by AIAD. The utility plotted at N interactions shows the utility achieved after the agent has been advised N times, but may additionally include an arbitrary number of direct edit actions taken by AIAD in between and after the advice.

For agents with no bias we observe no significant advantage in allowing AIAD to automate. This is not surprising as there are no biases to work around. However, on the second experiment, where biases are present, we see a significant ($p < 0.05$) improvement over standard AIAD across the board. AIAD + automation further outperforms the automation baselines after 7 interactions ($p < 0.01$). Although combining AIAD with automation shows promise, it is clearly not without risk. Automation means that the agent cannot interrupt AIAD if any model misspecification causes it to act poorly. This means that AIAD + automation cannot give the same agent control and safety guarantees as standard AIAD. But for applications where this is not a hard requirement allowing an assistant to automate can yield a significantly better assistance.

7 Conclusion

In this paper we have considered zero-shot assistance: the problem of assisting an agent in a decision problem when no prior knowledge of the agent's reward function or biases is available. To this end we have introduced *AI-Advised Decision-making* (AIAD), in which an assistant helps an agent by giving advice. We have introduced a decision-theoretic formalization of the assistant's problem of advising such an agent, and have proposed a planning algorithm for determining the assistant's policy. An important novelty in this formalization is that it accounts for individual agent biases, something which we showed experimentally improves the quality of the assistant's advice. Through our experiments we have also shown that both when biases are present and when not, assistance through advice yields better results than assistance through automation. Lastly we showed that a hybrid approach combining advice and automation outperforms advice alone at the cost of losing some of the safety guarantees.

Limitations Although our work does not require the explicit definition of a reward function, we do require the definition of a space of reward functions, which may still be difficult to provide. This difficulty is, however, inherent to any reward learning approach. Though our framework supports advice of any type, our experiments only covered action recommendations. Other types of advice could be designed to push an agent's reasoning in a general direction rather than toward a single action, or could include additional information (such as visualizations) designed to convince the agent of the quality of an action recommendation. We leave this to future work. Our experiments did not test the effects of misspecification in the agent model itself, only in the reward function. However, for complex agent like humans, it is likely that we would not be able to create a perfect agent model. As we have explained earlier our advise mechanism will mitigate the negative effects of this. However, it remains unclear how accurate an agent model has to be to allow the assistant to be useful.

Broader impact We believe that assistance through advice is a highly promising solution for the value alignment problem [14]. Advice reduces the negative effects of value misalignment in the assistant and ensures agent (human) control. Bad advice from a misaligned assistant will simply be rejected. Further, advice forces the assistant to be understandable, as it must convince the agent to follow its advice. Both of these elements present a significant improvement in AI safety compared to automation-based approaches. A risk that remains, however, is that the assistant may leverage a highly accurate agent model to find ways to deceive the agent or exploit its biases and limitations. This could pose a real safety risk, even if the assistant means well.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004. doi: 10.1145/1015330.1015430.

[2] Stuart Armstrong and Sören Mindermann. Occam's razor is insufficient to infer the preferences of irrational agents. *Advances in Neural Information Processing Systems*, 31, 2018.

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

[4] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009.

[5] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations. In *Proceedings of the 36th International Conference on Machine Learning*, pages 783–792. PMLR, 2019.

[6] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[7] Mustafa Mert Çelikok, Frans A Oliehoek, and Samuel Kaski. Best-Response Bayesian Reinforcement Learning with Bayes-adaptive POMDPs for centaurs. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 235–243, 2022.

[8] Lawrence Chan, Andrew Critch, and Anca Dragan. Human irrationality: both bad and good for reward inference. *arXiv preprint arXiv:2111.06956*, 2021.

[9] Paul F Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep Reinforcement Learning from Human Preferences. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4302–4310, 2017.

[10] Christos Dimitrakakis, David C Parkes, Goran Radanovic, and Paul Tylkin. Multi-View Decision Processes: The Helper-AI Problem. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[11] Avshalom Elmalech, David Sarne, Avi Rosenfeld, and Eden Shalom Erez. When Suboptimal Rules. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1313–1319, 2015.

[12] Owain Evans and Noah Goodman. Learning the preferences of bounded agents. *NIPS Workshop on Bounded Optimality*, 2015.

[13] Owain Evans, Andreas Stuhlmueller, and Noah Goodman. Learning the Preferences of Ignorant, Inconsistent Agents. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, page 323–329, 2016.

[14] Tom Everitt and Marcus Hutter. The Alignment Problem for Bayesian History-Based Reinforcement Learners. Technical report, 2018. URL `https://www.tomeveritt.se/papers/alignment.pdf`.

[15] Alan Fern, Sriraam Natarajan, Kshitij Judah, and Prasad Tadepalli. A Decision-Theoretic Model of Assistance. *Journal of Artificial Intelligence Research*, 50:71–104, 2014.

[16] Tim Genewein, Felix Leibfried, Jordi Grau-Moya, and Daniel Alexander Braun. Bounded Rationality, Abstraction, and Hierarchical Decision-Making: An Information-Theoretic Optimality Principle. *Frontiers in Robotics and AI*, 2, 2015. doi: 10.3389/frobt.2015.00027.

[17] Aditya Gopalan and Shie Mannor. Thompson Sampling for Learning Parameterized Markov Decision Processes. In *Proceedings of The 28th Conference on Learning Theory*, pages 861–898. PMLR, 2015.

[18] Arthur Guez, David Silver, and Peter Dayan. Efficient Bayes-Adaptive Reinforcement Learning using Sample-Based Search. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[19] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative Inverse Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[20] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "Other-Play" for Zero-Shot Coordination. In *International Conference on Machine Learning*, pages 4399–4410. PMLR, 2020.

[21] Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared Autonomy via Hindsight Optimization. In *Proceedings of Robotics: Science and Systems*, 2015. doi: 10.15607/RSS.2015.XI.032.

[22] W. Bradley Knox and Peter Stone. Interactively Shaping Agents via Human rRinforcement: The TAMER Framework. In *Proceedings of the Fifth International Conference on Knowledge Capture*, pages 9–16, 2009.

[23] Christopher Lucas, Thomas Griffiths, Fei Xu, and Christine Fawcett. A rational model of preference learning and choice prediction by children. In *22nd Annual Conference on Neural Information Processing Systems (NIPS)*, 2008.

[24] James N MacGregor, Thomas C Ormerod, and EP Chronicle. A model of human performance on the traveling salesperson problem. *Memory & Cognition*, 28(7):1183–1190, 2000.

[25] Andrew Y Ng and Stuart J Russell. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670, 2000.

[26] Pedro A Ortega and Daniel A Braun. Thermodynamics as a theory of decision-making with information-processing costs. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469(2153):20120683, 2013.

[27] Christian Perez, Felipe Petroski Such, and Theofanis Karaletsos. Generalized Hidden Parameter MDPs: Transferable Model-Based RL in a Handful of Trials. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5403–5411, 2020.

[28] Deepak Ramachandran and Eyal Amir. Bayesian Inverse Reinforcement Learning. In *IJCAI*, volume 7, pages 2586–2591, 2007.

[29] Singiresu S Rao. *Engineering Optimization: Theory and Practice*. John Wiley & Sons, 2019.

[30] Siddharth Reddy, Anca Dragan, and Sergey Levine. Shared Autonomy via Deep Reinforcement Learning. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.005.

[31] Rohin Shah, Noah Gundotra, Pieter Abbeel, and Anca Dragan. On the feasibility of learning, rather than assuming, human biases for reward inference. In *International Conference on Machine Learning*, pages 5670–5679. PMLR, 2019.

[32] Rohin Shah, Pedro Freire, Neel Alex, Rachel Freedman, Dmitrii Krasheninnikov, Lawrence Chan, Michael Dennis, Pieter Abbeel, Anca Dragan, and Stuart Russell. Benefits of Assistance over Reward Learning. *Workshop on Cooperative AI (Cooperative AI @ NeurIPS 2020)*, 2020.

[33] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2nd edition, 2018.

[34] Paolo Viappiani and Craig Boutilier. Optimal Bayesian Recommendation Sets and Myopically Optimal Choice Query Sets. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems*, page 2352–2360, 2010.

[35] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive Agent Shaping in High-Dimensional State Spaces. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, volume 32, pages 1545–1553, 2018.

[36] Christian Wirth, Riad Akrour, Gerhard Neumann, Johannes Fürnkranz, et al. A survey of Preference-Based Reinforcement Learning Methods. *Journal of Machine Learning Research*, 18(136):1–46, 2017.

A The GHPMCP algorithm

This section provides a detailed description of the GHPMCP algorithm. As explained in section 4.2, GHPMCP is based on *Monte-Carlo Tree Search* (MCTS) [6]. In every planning iteration GHPMCP samples ω and θ from respectively a posterior distribution over reward parameters p_ω and bias parameters p_θ, and simulates $\mathcal{M}_{\omega,\theta}$ down the tree following an Upper Confidence bound for Trees (UCT) policy [6].

Algorithm 1 GHPMCP

function $\mathrm{plan}(s, p_\omega, p_\theta)$
  for $i = 1 \dots n\_iterations$ **do**
    $\omega \sim p_\omega$
    $\theta \sim p_\theta$
    $h \leftarrow s$
    $\mathrm{simulate}(h, s, \mathcal{M}_{\omega,\theta}, 1)$
  end for
  $h \leftarrow s$
  return $\arg\max_a Q(h, s, a)$
function $\mathrm{simulate}(h, s, M_{\omega,\theta}, d)$
  if $N(h, s) = 0$ **then**
    for all $a \in \mathcal{A}'$ **do**
      $N(h, s, a) \leftarrow 0$
      $Q(h, s, a) \leftarrow 0.0$
    end for
  end if
  $a' \leftarrow \arg\max_{a \in \mathcal{A}'} Q(h, s, a) + c\sqrt{\frac{\log N(h,s)}{N(h,s,a)}}$
  $s' \sim \mathcal{T}_{\omega,\theta}(\cdot \mid s, a')$
  $r \leftarrow \mathcal{R}_\omega(s, a', s')$
  $h' \leftarrow ha's'$
  $d' \leftarrow d + 1$
  if $N(h, s, a) = 0$ **or** $d = max\_depth$ **then**
    $q \leftarrow r + \gamma\ \mathrm{Est\_Value}(s', M_{\omega,\theta}, d')$
  else
    $q \leftarrow r + \gamma\ \mathrm{simulate}(h', s', M_{\omega,\theta}, d')$
  end if
  $N(h, s) \leftarrow N(h, s) + 1$
  $N(h, s, a) \leftarrow N(h, s, a) + 1$
  $Q(h, s, a) = Q(h, s, a) + \frac{q - Q(h,s,a)}{N(h,s,a)}$
  return q

Algorithm 1 shows this procedure in detail. The function $\mathrm{plan}(\dots)$ runs a predetermined number of planning iterations with sampled $\mathcal{M}_{\omega,\theta}$ and returns the action with the highest Q-value. It takes as input the posteriors p_ω and p_θ which we assume are maintained and updated externally. In our implementation we used a particle filter to maintain these beliefs. $\mathrm{simulate}(\dots)$ implements the UCT policy used to recursively simulate $\mathcal{M}_{\omega,\theta}$ down the tree, up to a maximum depth max_depth. In Algorithm 1 we have used the variable h, which represents the path from a node to the root of the tree, to identify tree nodes. Variable d represent the current depth. As simulation progresses down the tree, the state and action node visit counts $N(h, s)$ and $N(h, s, a)$ and Q-values $Q(h, s, a)$ of nodes encountered are updated (last lines of the procedure). Simulation proceeds recursively until a leaf node is reached, either because the maximum depth is reached or because a new node is encountered (second if statement). The values of leaf state nodes are estimated using $\mathrm{Est\_Value}(\dots)$, which may implement a roll-out algorithm or a function approximator.

B Additional details on the experiments

B.1 Objective function

We describe here in detail the objective function $f_\omega(s)$ we use in our experiments. We will first introduce some notation. Let p_i be the i-th POI. Any trip s is a subset of the $N = 100$ available POIs $\{p_i\}_{i=1}^N$. Let $c(p_i)$ be the admission cost for visiting p_i and let $d(p_i)$ be the time it takes to visit it in minutes. Similarly let $c(s) = \sum_{p_i \in s} c(p_i)$ be the total admission cost of a trip. Finally, let $t_{p_i}^j = 1$ if p_i belong to topic j, and 0 otherwise. Let there be $M = 20$ topics. Analogously let $t_a^j = 1$ if the agent is interested in topic j and 0 otherwise. The set of reward parameters ω then consists of the parameters $\{t_a^j\}_{j=1}^M$, and two additional parameters $\mu_c \in [0, \infty)$ and $\sigma_c \in [0, \infty)$ which we will define below.

The objective function f_ω is a product of two scores: $f_\omega(s) = f_\omega^{(1)}(s) f_\omega^{(2)}(s)$. The first score, $f_\omega^{(1)}(s)$, measures the agent's enjoyment of visiting the POIs in s. To define this we must first define an agent's interest, according to ω, in a POI p_i. This is defined as the fraction of the POI's topics the agent is interested in:

$$\text{interest}(p_i; \omega) = \frac{\sum_{j=1}^M t_{p_i}^j t_a^j}{\max\left(\sum_{j=1}^M t_{p_i}^j, 1\right)}$$

The shading of the POIs in Figure 2 correspond to this function for a random ω. We can now define

$$f_\omega^{(1)}(s) = \frac{\sum_{p_i \in s} d(p_i) \text{interest}(p_i; \omega)}{\text{MAX\_TIME}}$$

where MAX_TIME is the maximum allowed duration of a day trip in minutes, 12×60 in our case.

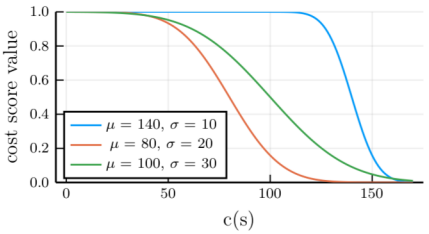


Figure 4: Different $f_\omega^{(2)}(s)$ as a function of $c(s)$ for different values for the mean μ_c and standard deviation σ_c.

The second score, $f_\omega^{(2)}(s)$, determines the agent's willingness to pay the total admission cost of the day trip $c(s)$. It is defined as[4]

$$f_\omega^{(2)}(s) = 1 - cdf\left(\mathcal{N}_0^\infty(\mu_c, \sigma_c), c(s)\right)$$

where $cdf(d, x)$ is the cumulative density function of distribution d at x. Note that for a specific choice of mean μ_c and standard deviation σ_c, $f_\omega^{(2)}(s)$ defines the opposite of a sigmoid function. This function has value 1 at $c(s) = 0$ and slopes down to value 0 at $c(s) = \infty$. It crosses the 0.5 threshold at μ_c while σ_c determines how quickly it slopes down. Figure 4 visualizes how μ_c and σ_c change $f_\omega^{(2)}(s)$. Intuitively μ_c roughly determines the agent's cost limit while σ_c determines how stringent the agent is with that cost limit.

B.2 Determining day trip durations

In our implementation we put a soft limit of 12 hours on the length of a day trip. This means that once the duration of an agent's day trip exceeds this, it is not allowed to add further POIs to its trip. The duration of a day trip depends on the time needed to visit the POIs in the day trip ($\sum_{p_i \in s} d(p_i)$

[4]We use the notation $\mathcal{N}_a^b(\mu, \sigma)$ to denote a normal distribution with mean μ and standard deviation σ, truncated on the interval $[a, b]$.

for a day trip s) and the time it takes to move between the POIs. To estimate the time it takes to move between two subsequent POIs in an itinerary, we calculate the straight-line distance between them (in kilometers), and divide it by the agent's walking speed. The agent's walking speed is fixed to 5 km/h.

B.3 Prior distributions and POI generation

Generating POIs The POIs used in the experiments are randomly generated. POI attributes are generated from the following priors. The priors were chosen to be representative of a real day trip problem. The x and y coordinates, in kilometers from an artificial center at $(0, 0)$, are sampled from a $\mathcal{N}^5_{-5}(0, 1.15)$ distribution. The admission cost of a POI is sampled from $\mathcal{N}^\infty_0(10, 3)$ and the visit time in minutes from $\mathcal{N}^{100}_0(30, 20)$. Whether a POI p_i belongs to a topic j is sampled from Bernouilli distribution for every one of the 20 topics: $t^j_{p_i} \sim Bern(0.1)$.

Priors over Θ and Ω The agents we simulate in our experiments are instantiated with parameter values sampled from the priors $p_{0,\omega}$ and $p_{0,\theta}$. All parameters have independent prior distributions, which are listed in the table below. The prior distributions were chosen to be realistic for a day trip. The prior over the temperature parameter β_1 was tuned to yield agents that were optimal enough to benefit from assistance (of any kind, not specifically our method), but not so optimal that they would not need assistance. These priors are also used in the beliefs maintained by the agent, with one minor difference. Within the agent's beliefs, β_1, the temperature parameter in equation 2, is always assumed to be 2. This simplifies belief maintenance slightly and had no effect on the quality of assistance.

Table 1: Prior distributions for the parameters of the agent model. The left-most column indicates whether the parameter is part of the reward parameters Ω or the bias parameters Θ

	name	prior distribution	explanation
Ω	t^j_a	$Bern(0.3)$	indicates whether the agent is interested in topic j
	μ_c	$\mathcal{N}(140, 25)$	parameter of cost the score
	σ_c	10	parameter of the cost score
Θ	β_1	$Unif(1, 4)$	temperature parameters of the choice in eq. (2)
	β_2	$10\beta_1$	temperature parameters of the choice in eq. (3)
	anchoring bias	$Bern(0.5)$	if applicable, indicates if the agent has an anchoring bias

B.4 Algorithms and hyperparameters

Q-Value estimation in the agent model The Q-values used in the agent model are obtained from a decision tree. This tree is constructed with depth-limited best first search (BFS) over the agent's view of the problem $\hat{E}$. We ran 500 iterations of BFS with a depth limit of 3.

Planning for assistance We used GHPMCP for all planning within our experiments. It was used to plan over the AIAD formulation to find a policy for the assistant, but also to find an optimal policy to follow when automating. Below we describe in detail how this automating policy was obtained. Table 2 describes the parameter values we used for GHPMCP. Apart from the discounting rate, these parameters were tuned through experimentation.

Because automation involves operating directly in the agent's decision problem, an automating policy must be optimal within E. But, as the agent's true reward function R_ω is not known, we can only optimize this policy w.r.t. our current beliefs regarding the reward parameters. We therefore formulated this planning problems as a GHP-MDP $\mathcal{E} = \langle \mathcal{S}, \Omega, \mathcal{A}, T, \mathcal{R}, \gamma, p_{0,s}, p_\omega \rangle$. Here $\mathcal{S}, \mathcal{A}, T, \mathcal{R}, \gamma$ and $p_{0,s}$ are as they were defined in E (see section 3). The hidden parameters of this GHP-MDP are the reward parameters Ω. The prior distribution over Ω is the posterior distribution p_ω that has been inferred from the agent using IRL or PL. Note that because T does not depend on any of the parameters of the GHP-MDP, the observed transitions are not informative toward the unobserved parameters. It is therefore impossible to update our beliefs about ω beyond the prior distribution p_ω. The automation policy, the policy used when automating, was then the policy obtained from planning using GHPMCP on $\mathcal{E}$.

Table 2: Parameters used in GHPMCP for both AIAD and automation. The names of the parameters correspond to those used in Algorithm 1.

	parameter value		
parameter	AIAD	automation	description
γ	0.95	0.99	discounting rate used while planning
max_depth	2	2	maximum depth of the tree
$n\_iterations$	750,000	10,000,000	number of planning iterations
c	0.1	0.1	exploration constant used by the UCT policy
Est_Value(...)	$= 0$	$= 0$	function estimating the state value of leaf nodes

Because our agent model $\hat{\pi}(a \mid s, a'; \theta, \omega)$ is expensive to evaluate we used caching to cache some of its computations. To increase the hit rate of the cache while planning with GHPMCP, we sub-sampled the particle filter used to maintain the assistant's posterior beliefs. The sub-sampled particle filter contained 100 particles, sampled according to their weights, and was re-sampled after every update of the beliefs.

B.5 Implementation and computational resources

Our experiments were implemented in Julia. The code is available under MIT licence from the supplemental. The implementation relies heavily on the excellent POMDPs.jl ecosystem [5] (available under MIT licence). The plots in this paper were generated using Plots.jl. [6] A complete list of the packages used is available in the "Project.toml" file that is part of the codebase.

Our experiments were run on a private cluster consisting of a mixture of Intel ® Xeon ® Gold 6248, Xeon ® Gold 6148, Xeon ® E5-2690 v3 and Xeon ® E5-2680 v3 processors. We report in table 3 and table 4 the computational resources used in our experiments. As automation was run multiple times per run in the PL and IRL baselines – after 0, 5, 10, 15, 20, 25, and 30 interactions with the agent – we have split reward inference and automation in these tables. The per run figures reported for automation for these baselines is averaged over all times when automation was ran.

Tables 3 and 4 additionally contain carbon intensity estimates. For these estimates we have assumed that all experiments were run on an Intel ® Xeon ® Gold 6248 CPU running at maximum power. Most of our experiments were run on this specific model of CPU. Our cluster is located in Finland. According to the latest available data (year 2020), the average carbon intensity of one KWh of electricity consumed in Finland is 72 grams. [7] This yields estimated carbon emissions of 0.00015 grams CO_2 per second of CPU time.

B.6 Additional results for day trip design without biases

Here we present additional results from the first and third experiment, where which we considered agents without biases. Table 5 presents the objective value achieved by agents assisted by the different methods tested.

[5] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for Sequential Decision Making under Uncertainty. Journal of Machine Learning Research, 18(26):1–5, 2017. URL http://jmlr.org/papers/v18/16-300.html

[6] Tom Breloff. Plots.jl. URL https://doi.org/10.5281/zenodo.6523361

[7] Finngrid Oyj. Real-time CO2 emissions estimate. URL https://www.fingrid.fi/en/484 electricity-market-information/real-time-co2-emissions-estimate/

Table 3: Computational resources used and associated carbon intensity (CI) for various assistance methods considered for the experiments on unbiased agents (see sections 6.1 and 6.3).

	average per run			total	
	memory (GiB)	CPU time	CI (g CO_2)	CPU time	CI (g CO_2)
AIAD	3.64	11h 35m 48s	6.26	579h 50m 0s	313
AIAD + automation	3.66	11h 52m 37s	6.41	593h 60m 50s	321
IRL – inference	1.18	4h 29m 13s	2.42	224h 20m 50s	121
IRL – automation	1.54	1h 24m 43s	0.76	494h 10m 50s	229
PL – inference	0.81	2h 31m 1s	1.36	125 50m 50s	68.0
PL – automation	1.49	1h 13m 58s	0.67	369h 50m 0s	200
oracle + automation	1.5	1h 26m 3s	0.77	71m 42m 30s	39
			total:	2459h 35m 50s	1328

Table 4: Computational resources used and associated carbon intensity (CI) for various assistance methods considered for the experiments on agents with potential anchoring biases (see sections 6.2 and 6.3).

	average per run			total	
	memory (GiB)	CPU time	CI (g CO_2)	CPU time	CI (g CO_2)
AIAD	3.56	10h 9m 33s	5.49	761h 56m 15s	411
AIAD, no bias	3.73	10h 25m 47s	5.63	825h 13m 45s	422
AIAD, bias ass.	2.82	6h 46m 26s	3.66	508h 2m 30s	274
AIAD + automation	4.32	13h 46m 11s	7.44	1032h 43m 45s	558
IRL – inference	1.16	6h 47m 4s	3.66	508h 50m 0s	274
IRL – automation	1.54	1h 25m 31s	0.77	748h 16m 15s	404
PL – inference	0.81	2h 31m 5s	1.36	188h 51m 15s	102
PL – automation	1.51	1h 16m 10s	0.69	571h 15m 0s	308
oracle + automation	1.54	1h 16m 55s	0.69	96h 8m 45s	52
			total:	5198h 17m 30s	2807

Table 5: Mean objective value $\pm$ standard error as a function of different amounts of agent interaction achieved by agents without biases when unassisted, or assisted by the baselines and variations of AIAD (see sections 6.1 and 6.3). Bold shows a significant ($p < 0.05$) improvement – given the same number of interactions – over the unassisted agent and baselines. For *AIAD + autom.* bold additionally shows an improvement over standard AIAD.

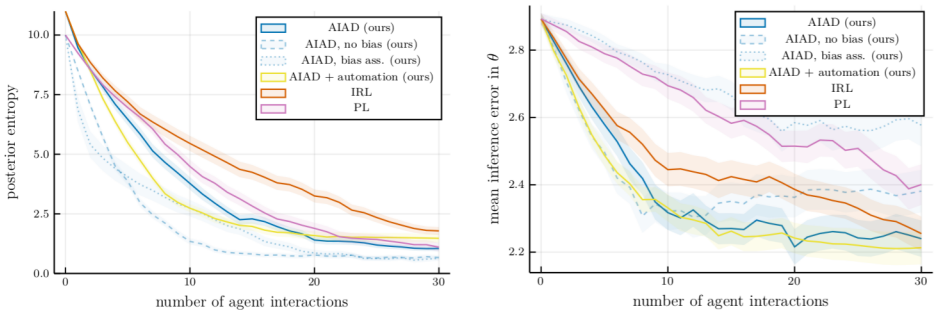
	number of agent interactions					
	5	10	15	20	25	30
unassisted agent	0.19 ± 0.01	0.33 ± 0.02	0.39 ± 0.02	0.41 ± 0.02	0.42 ± 0.02	0.43 ± 0.02
IRL + autom.	0.30 ± 0.03	0.35 ± 0.03	0.38 ± 0.02	0.36 ± 0.02	0.36 ± 0.02	0.37 ± 0.02
PL + autom.	0.26 ± 0.02	0.27 ± 0.02	0.29 ± 0.02	0.32 ± 0.03	0.29 ± 0.03	0.28 ± 0.03
AIAD	0.25 ± 0.01	$\mathbf{0.44 \pm 0.02}$	$\mathbf{0.50 \pm 0.02}$	$\mathbf{0.50 \pm 0.02}$	$\mathbf{0.51 \pm 0.02}$	$\mathbf{0.51 \pm 0.02}$
AIAD + autom.	0.21 ± 0.01	0.45 ± 0.02	0.48 ± 0.02	0.48 ± 0.02	0.48 ± 0.02	0.48 ± 0.02

B.7 Additional results for day trip design with anchoring bias

Here we present additional results from the second and third experiment, in which we considered agents that potentially had an anchoring bias. Table 6 presents the objective value achieved by agents assisted by the different methods tested. Figure 5 shows the posterior entropy and reward parameter estimation error of the different methods.

Table 6: Mean objective value $\pm$ standard error as a function of different amounts of agent interaction achieved by agents with potential anchoring bias when unassisted, or assisted by the baselines and variations of AIAD (see sections 6.2 and 6.3). Bold shows a significant ($p < 0.05$) improvement – given the same number of interactions – over the unassisted agent, baselines, *AIAD, no bias* and *AIAD, bias ass.*. For *AIAD + autom.* bold additionally shows an significant ($p < 0.05$) improvement over standard AIAD.

	number of agent interactions					
	5	10	15	20	25	30
unassisted agent	0.11 ± 0.01	0.24 ± 0.02	0.31 ± 0.02	0.34 ± 0.02	0.36 ± 0.02	0.38 ± 0.02
IRL + autom.	0.26 ± 0.02	0.33 ± 0.02	0.34 ± 0.02	0.35 ± 0.02	0.37 ± 0.02	0.40 ± 0.02
PL + autom.	0.25 ± 0.02	0.29 ± 0.02	0.29 ± 0.02	0.33 ± 0.02	0.29 ± 0.02	0.31 ± 0.02
AIAD, no bias	0.18 ± 0.02	0.33 ± 0.02	0.38 ± 0.03	0.42 ± 0.02	0.43 ± 0.02	0.42 ± 0.02
AIAD, bias ass.	0.16 ± 0.01	0.33 ± 0.02	0.40 ± 0.03	0.43 ± 0.02	0.45 ± 0.02	0.46 ± 0.02
AIAD	0.18 ± 0.02	0.35 ± 0.02	$\mathbf{0.43 \pm 0.02}$	$\mathbf{0.46 \pm 0.02}$	$\mathbf{0.47 \pm 0.02}$	$\mathbf{0.48 \pm 0.02}$
AIAD + autom.	$\mathbf{0.35 \pm 0.01}$	$\mathbf{0.49 \pm 0.02}$	$\mathbf{0.51 \pm 0.02}$	$\mathbf{0.51 \pm 0.02}$	$\mathbf{0.51 \pm 0.02}$	$\mathbf{0.51 \pm 0.02}$



(a) posterior entropy

(b) reward parameter estimate error

Figure 5: Additional results for the day trip design experiments with agents with potential anchoring bias (see sections 6.2 and 6.3). The shading and error bars in all plots show the standard error of the mean. **(a)** Posterior entropy as a function of the number of agent interactions of the methods considered. **(b)** Mean error between these same methods' posterior over reward parameters and the true reward parameters.