Chapter 6

Automated Environment Design

In the previous chapters, we introduced a number of designs for crowdsourcing complex tasks that are effective in recruiting individuals with relevant expertise to join in problem solving and enabling coordination and collaboration. To promote desired behaviors and outcomes, we focused on reasoning about the crowd's abilities, limitations, and work processes in order to construct workflows, interfaces, and incentive mechanisms that are tailored to the characteristics of the crowd.

While we have focused thus far on the design of human computation systems, understanding participants and their behavior is crucial for designing any social or economic system. Participants have varied knowledge and abilities, interests and motivations, availability, and decision-making processes. Together with the decision environment, these elements influence participants' decisions on what actions to take. Designers can draw on what they know, but do not typically have a complete understanding of participants and cannot always predict their behavior. For this reason, solving a computational environment design problem may require experimenting with

alternative designs, and iterating to improve designs over time to better promote desired behaviors.

The Internet provides a number of tools for designers that support a data-driven, iterative design process. Frameworks, style sheets, and content management systems make it easier to modify or extend existing designs. Web analytics software tracks individual and group behaviors over time, and provides information on trends and patterns in the data. Tools for A/B testing allow designers to put hypotheses to the test, by measuring the performance of competing designs against defined objectives.

But despite having a rich set of tools, the process of discovering effective designs is still largely manual, tedious and ad hoc. Designers spend significant time and effort coming up with alternative designs, that may consist of small modifications geared towards making immediate improvements. Without particular regard to gaining a deeper understanding of participants or of potential interactions among design elements, this may lead to an experimentation process that tries to hill-climb toward a solution at a local, rather than global, maximum. Designers may miss out on parts of the design space where better solutions exist, and ultimately fail to promote desired behaviors and outcomes.

A more principled and automated approach to experimentation may lead to more effective designs more quickly, while requiring less manual effort. Such an approach may use observations of participant behaviors to not only evaluate competing designs, but also to refine our understanding of participants' abilities, motivations, and decision-making processes. This knowledge may allow us to reason about the design space more globally, and to discover designs that we would otherwise have missed. To

reduce the amount of manual effort required, and to discover effective designs more quickly, automated procedures can be employed to seamlessly combine domain knowledge with machine-driven processes that optimize the choice of experiments and refine existing models based on observed behavior. From the perspective of the designer, an automated system may simply take as input a set of available interventions, the objective of the designer, and a model of participants, and provide as output an intervention that promotes actions and outcomes meeting the objective whenever such interventions exist, or otherwise learn something new about participants.

In this chapter, we introduce a general approach for *automated environment design*. Section 6.1 presents a formal model of the automated environment design problem. Section 6.2 provides an *active, indirect elicitation* framework that automatically drives an objective-oriented, iterative design process in which a system indirectly learns about participants based on observations of participant behavior in response to experiments chosen based on current knowledge. Section 6.3 introduces the problem of *policy teaching* as a case study, in which an interested party aims to provide limited rewards to induce an agent in a sequential-decision setting to follow a desired policy. We construct an active, indirect elicitation algorithm, that without prior knowledge of the agent's reward function, is guaranteed to discover rewards in a constrained reward space that elicit the desired policy after few interactions, as long as such rewards exist. Section 6.4 describes how our methods and results may generalize to other automated environment design problems, and discusses our assumptions as well as alternative models and approaches for automated environment design.

6.1 Model for Automated Environment Design

We consider situations in which an automated system, which we refer to as an *interested party*, seeks to design or modify aspects of a social or economic system on the Internet with the intent of eliciting desired actions and outcomes. For simplicity of notation and without loss of generality, we model participants in a system as if they were a single *agent*.[1] A model for an automated environment design problem consists of a decision environment, an agent model, an environment change, an admissibility condition, an environment transition function, and a goal function. Below we define these components, and present static and dynamic formulations of the problem.

Consider an agent who acts in a *decision environment* $e \in \mathcal{E}$ based on his agent model $\mathcal{M} = \{\theta, f, \Lambda\}$, which consists of the *model parameters* $\theta \in \mathcal{I}$; the *agent function* $f : \mathcal{I} \times \mathcal{E} \to 2^{\mathcal{X}}$, where $\mathcal{X}$ is the decision space; and the *actuation function* $\Lambda : \mathcal{X} \times \mathcal{E} \to \mathcal{O}$, where O is the output space. The model parameters represent the agent's preferences and capabilities, and contains information private to the agent. The agent function takes the model parameters and environment as input and identifies (perhaps multiple, equivalent) decisions, which describe how the agent plans to act in the environment. The actuation function takes the agent's decision and the environment and provides an output representing the agent's actual actions in the environment. Although described here as deterministic for expositional clarity, the actuation function need not in general map a decision to an output deterministically, and may instead sample from a distribution over actions. Furthermore, while

[1] Interactions among participants can be captured by having the agent model take into account how participants in a decision environment may interact and make decisions based on other participants' actions.

the agent's actions may sometimes reveal the agent's exact decision, we assume that decisions are not directly observable.

We make a couple of assumptions about the agent model. First, we assume that the agent fully perceives the decision environment and makes decisions with respect to that knowledge.[2] Second, we assume that f and Λ are fixed and known to the interested party. This abstraction implies that if the interested party had full knowledge of the agent's model parameters, he would be able to predict the agent's decisions and a distribution over agent actions in the designed environment. Third, we assume the agent can compute f on any input he encounters, such that any computational limitations of the agent is embedded within f. Lastly, we assume that the agent makes a single decision $x \in f(\theta, e)$ when f returns a non-singleton set of decisions, with this tie-breaking rule *a priori* unknown to the interested party.

Having described the agent model, we turn to consider the interested party's problem. We assume the presence of a *base environment* e^0, which the interested party can modify via an *environment change* $\Delta \in \mathbf{\Delta}$. The *environment transition function* $\mathcal{F} : \mathcal{E} \times \mathbf{\Delta} \to \mathcal{E}$ takes the base environment e^0 and an environment change as input and outputs a modified environment. We assume this function is deterministic and known to the interested party. Furthermore, we assume that once the environment is modified, the agent acts with respect to a decision in the modified environment. Since the environment enters as input into the agent function, modifying the environment may influence the agent's decision and actions. We assume that the interested party fully perceives the environment, and can observe the agent's actions.

[2] Alternatively, one can define f based on the agent's perceptual inputs as opposed to the environment. For sake of exposition we do not explicitly model the agent's perception.

We assume that the agent is myopic with respect to environment changes. That is, the agent follows his agent function and does not reason about future changes to the environment when making current decisions. This seems reasonable in social and economic systems on the Web, in which there are large numbers of users, most of whom tend to use services as desired without reasoning about how systems may change in the future. Furthermore, as design decisions tend to be guided by the behaviors of many users, a single individual's actions are unlikely to affect a system's (re)design. That said, it is generally possible for users to take actions with the intent of influencing environment changes; we elaborate on this issue later in the chapter.

Given a set $X \in 2^{\mathcal{X}}$ of agent decisions that may result from an environment change, the admissible set $\mathit{admissible}(X) \subseteq \mathbf{\Delta}$ characterizes the space of allowable environment changes. Admissibility conditions can model the interested party's design costs and constraints, both of which may potentially depend on the agent's decisions. For example, an environment change that rewards user actions may be infeasible if agent decisions in the modified environment lead to actions that require the interested party to issue more rewards than he has available. We assume the admissible set always contains a null element Φ, corresponding to no environment change.

Finally, we define the goal of the interested party. The *goal function* $\mathcal{G} : \mathcal{X} \times \mathbf{\Delta} \times \mathcal{I} \times \mathcal{E} \rightarrow \Re$ takes the agent's decision under the modified environment, the environment change, the agent's model parameters, and the modified environment as input and outputs the value to the interested party.[3] The goal may depend on

[3]Since the agent's decision is not directly observable, in practice an interested party may use samples of observed actions to evaluate admissibility and goal conditions. Since the agent's model parameters are also private to the agent, the interested party may need to evaluate the goal function with respect to beliefs about the actual model parameters.

Environment	a Web 2.0 site
Agent model parameters	preferences over site modules; time available to spend online
Agent function	decision on what to do on the site based on interest and availability
Actuation function	actual user actions on the site based on user decisions
Environment change	adding, removing, and moving modules in the user interface
Admissibility condition	limit to changes within template; keep main components centered and visible
Environment transition function	describes how the user interface changes
Goal function	retention rate among new users; the volume of content contributed

Table 6.1: An example showing the various components of a computational environment design problem in which an interested party wishes to redesign the user interface of a Web 2.0 site to improve retention rate and increase the volume of user contributions.

(a) the agent's decision because it determines (the distribution over) agent actions and outcomes; (b) the environment change because this may come at a cost; (c) the model parameters because the designer may wish to consider the value to the agent; and (d) the modified environment because the interested party may value the agent's decisions differently in different environments.

To get a sense of how the model applies to computational environment design problems we may encounter in practice, see Table 6.1, which illustrates the various components of a computational environment design problem in which an interested party wishes to design the user interface of a Web 2.0 site to improve retention and increase the volume of user contributions.

6.1.1 Static Formulation

As a special case, we first present the *static formulation* of the automated environment design problem, in which we assume that the agent's model parameters are known to the interested party. The goal is to find an admissible Δ such that the agent's elicited behavior in the modified environment maximizes the goal function $\mathcal{G}$. Since the interested party already knows the agent function and the environment, we can think of the interested party's problem as a one-shot optimization problem.

In the case of multiple possible decisions in the range of the agent function, the agent may not select the one desired by the interested party. To be certain that the agent selects decisions desired by the interested party, our formulation assumes that the agent selects the worst possible decision for the interested party's goal function:

Definition 6.1. *Given an environment e, the static computational environment design problem is an optimization problem to find an environment change Δ that maximizes the interested party's goal function in the worst case:*

$$\max_{\Delta}[\min_{x_T} \mathcal{G}(x_T, \Delta, \theta, e')] \tag{6.1a}$$

$$\textit{subject to:} \quad e' = \mathcal{F}(e, \Delta) \tag{6.1b}$$

$$x_T \in f(\theta, e') \tag{6.1c}$$

$$\Delta \in \mathit{admissible}(f(\theta, e')) \tag{6.1d}$$

In the case that the agent function outputs singleton decision sets, the objective of the optimization simplifies to $\max_{\Delta} \mathcal{G}(x_T, \Delta, \theta, e')$.

The constraints ensure that e' is the modified environment (6.1b), that the model parameters and modified environment induce some decision x_T (6.1c), and that the

environment change is admissible with respect to all possible agent decisions (6.1d) consistent with the new environment.

6.1.2 The Dynamic Formulation

In the more interesting case, and the focus of this chapter, the agent's model parameters will initially be, at least partially, unknown to the interested party. Since the agent function depends on both the environment and the model parameters, the interested party may not be able to immediately identify admissible environment changes that promote the desired behavior. To address this, the interested party can experiment with alternative designs and have repeated interactions with the agent. In each interaction, the interested party can modify the environment and observe the agent's actions in the modified environment.

Observations and measurements can inform which experiments to conduct in subsequent interactions, and the goal is to arrive at effective designs quickly. An example objective may be to induce desired decisions after few interactions, without being concerned about the cost of experimentation. Given a target goal value $\underline{\mathcal{G}}$, we can represent this objective as minimizing the number of rounds until we find an admissible Δ that induces a decision environment e' in which the agent's decision x_T satisfies $\mathcal{G}(x_T, \Delta, \theta, e') \geq \underline{\mathcal{G}}$.

More generally, we can imagine that in the midst of experimentation, the interested party is (in a separate process) using the results of experimentation to deploy environment changes. Deployed designs may be designs from past experiments or new designs that are computed using currently available information. Viewed this way,

the interested party may wish to maximize one of several objectives that represent the exploration and exploitation tradeoff of having effective designs to deploy now versus later. For any point along this spectrum, the goal is to design experiments that maximize some measure of the expected goal value derived from deploying environment changes now and in the future. Different objectives weigh the value derived from experimentation differently, depending on when particular designs are discovered and deployed.

6.2 An Active, Indirect Elicitation Framework

Solving the dynamic formulation requires discovering effective designs quickly. To make efficient use of experiments, we can draw on observations and measurements to not only evaluate competing designs, but to refine our understanding of model parameters guiding the agent's behavior. For example, one can infer from observing consumer purchases and worker performance on tasks information about the underlying preferences and abilities that guide the person's decisions and actions. As an agent makes decisions in different environments with respect to his *actual* model parameters, we can use observed behavior to make inferences about the space of model parameters consistent with observations. Even without identifying the agent's actual model parameters, such information and knowledge may allow us to better predict how an agent will respond to different designs. This enables us to reason more effectively about the design space.

Taking advantage of this insight, we introduce an *active, indirect elicitation framework* that drives an automated, iterative design process that interleaves optimiza-

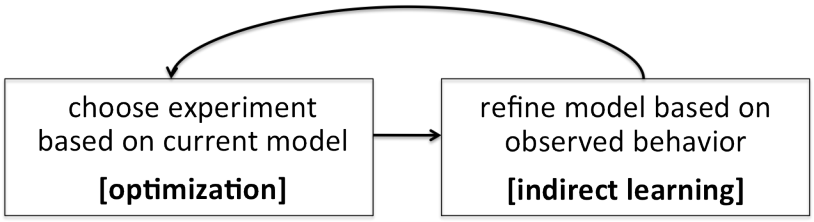

Figure 6.1: The active, indirect elicitation framework combines optimizing experiments based on current knowledge of model parameters with indirect learning of model parameters based on observed behavior.

tion of appropriate experiments with indirect learning of model parameters (see Figure 6.1). In each round, an experiment is designed using knowledge of the agent's model parameters, and seeks to derive new information from observing potential agent actions in the modified environment. Following an interaction, the knowledge of model parameters is refined by making inferences based on observed behavior. Since the goal is ultimately to elicit desired actions, experiments should be selected with the interested party's objective in mind, and not just for the sake of learning about the agent's underlying model parameters.

An algorithm based on the active, indirect elicitation framework contains two components: an inference procedure and an elicitation strategy. An *inference procedure* updates the interested party's beliefs about the actual model parameters, by incorporating new observations from experiments. Let $\mathcal{H}$ denote the history of past elicitation rounds, such that $(o^t, e^t) \in \mathcal{H}$ denotes observed actions o^t in environment e^t in round t. For all observations $(o^t, e^t) \in \mathcal{H}$, the agent's actual model parameters θ^* must satisfy $f(\theta^*, e^t) = x^t$, where x^t is the agent's decision in round t that, through the actuation function $\Lambda(x^t, e^t)$, led to the observed output o^t. By making inferences based on the relationships among these components, the inference procedure allows

us to refine our beliefs about θ^* over time. Indirectly, this enables us to better predict the agent's decisions and actions in response to different environment changes.

The *elicitation strategy* optimizes for experiments based on our beliefs, as provided by the inference procedure using the history $\mathcal{H}$. Depending on the interested party's objective, the elicitation strategy may focus on obtaining information that would most immediately lead to an improved design, or be more forward looking by taking into consideration the potential value that can be derived in the future from information learned now.

6.3 Case Study: Policy Teaching

For an algorithm based on the active, indirect elicitation framework to be practically useful, the inference procedure and elicitation function must be computationally tractable and help to discover effective designs quickly. To illustrate how the active, indirect elicitation framework can be applied to a specific automated environment design problem, we consider as a case study the problem of *policy teaching*.

Policy teaching considers a Markov Decision Process (MDP) setting in which an interested party can associate rewards with world states to affect an agent's policy. The interested party can observe the agent's decisions in response to provided incentives, but generally does not know the agent's reward function. The interested party can interact multiple times with the agent, but cannot directly impose actions on the agent. The goal of the interested party is to quickly identify feasible incentives (i.e., rewards from a constrained reward space) that induce the agent to follow a desired behavior or policy, when this is possible.

Policy teaching models situations on the Web in which an interested party can modulate costs and rewards in attempt to elicit desired actions. For example, a retailer such as Amazon may want customers to make frequent purchases and write product reviews, and may be willing to provide discounts on products and recognize top reviewers. Question-and-answer sites such as Yahoo! Answers and Stack Overflow may want users to answer lingering questions and generally spend time on the site, and can tweak their interfaces to make it easier to contribute (thus reducing the cost of effort) and offer points and badges as social rewards. Ad networks such as Google AdSense may want publishers to design their web sites to facilitate effective advertising, and can offer a share of the ad revenue to entice a publisher to choose a particular web layout.

We focus on the policy teaching problem in which the goal is to induce a fixed, *prespecified desired policy*. Section 6.3.1 provides a model of this automated environment design problem. Section 6.3.2 shows that in the static case, the problem can be formulated as a linear program. Section 6.3.3 considers the more likely case where the agent's reward function is unknown, and introduces an active, indirect elicitation algorithm that is guaranteed to converge after a few rounds to discover rewards to apply to states that induce the desired policy. To make the algorithm tractable, we apply results from sampling in convex spaces [6] to arrive at a polynomial time algorithm that maintains the same convergence guarantees with arbitrarily high probability. Section 6.3.4 summarizes our results and discusses a few extensions.

6.3.1 Model

The policy teaching problem considers an agent performing a sequential decision task with respect to an infinite horizon MDP $M = \{S, A, R, P, \gamma\}$, where S is a finite set of states, A is a finite set of possible actions, $R : S \rightarrow \Re$ is the reward function, $P : S \times A \times S \rightarrow [0, 1]$ is the transition function, and $\gamma \in (0, 1)$ is the discount factor. Given M, the agent's decision problem is to choose actions for each state to maximize the expected sum of discounted rewards. Let π denote a stationary policy, such that $\pi(s)$ is the action the agent executes in state s. Given a policy π, the value function $V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P(s, \pi(s), s')V^\pi(s')$ captures the expected sum of discounted rewards from state s. Similarly, the Q function captures the value of taking an action a and following the policy π in future states, such that $Q^\pi(s, a) = R(s) + \gamma \sum_{s' \in S} P(s, a, s')V^\pi(s')$. By Bellman optimality [76], an optimal policy π^* maximizes the Q function in every state, such that $\pi^*(s) \in \arg\max_{a \in A} Q^{\pi^*}(s, a)$. We assume the agent can compute an optimal policy of his MDP, and that his inherent reward function R is persistent.[4]

We consider an interested party whose goal is to induce a prespecified target policy π_T. The interested party knows S, A, P, and γ, but not the agent's reward function R. We assume that the interested party can observe the agent's actions, and that observed actions completely reveal the agent's policy (decision). The interested party can influence the agent's reward function by providing incentives $\Delta : S \rightarrow \Re$. We assume that Δ affects the agent's reward function linearly, such that the agent plans

[4]Mapping back to the general model, the agent function in this setting forms the agent's decision by computing the optimal policy with respect to the MDP model M, which captures aspects of both the environment and the agent's model parameters.

with respect to $M' = \{S, A, R + \Delta, P, \gamma\}$ in the modified environment. Following our base assumption that the agent is myopic with respect to environment changes, we assume the agent is *myopically rational* and follows the optimal policy in the modified environment.

To capture the idea that the interested party may only be able to provide limited incentives, we define a notion of admissibility:[5]

Definition 6.2. *An incentive function $\Delta : S \rightarrow \Re$ is admissible given budget D_{max} and Δ_{max} with respect to a policy π_T if it satisfies the following linear constraints, denoted $\Delta \in admissible(\pi_T)$:*

$$V_\Delta^{\pi_T}(s) = \Delta(s) + \gamma P_{s,\pi_T(s)} V_\Delta^{\pi_T}, \forall s \in S \qquad \textit{Incentive value.} \qquad (6.2)$$

$$V_\Delta^{\pi_T}(start) \leq D_{max} \qquad \textit{Limited spending.} \qquad (6.3)$$

$$0 \leq \Delta(s) \leq \Delta_{max}, \forall s \in S \qquad \textit{No punishments.} \qquad (6.4)$$

The incentive value $V_\Delta^{\pi_T}(s)$ in Definition 6.2 captures the total sum of expected discounted incentives provided to an agent following policy π_T starting from state s. The limited spending constraint limits the total incentives provided to D_{max} when the agent performs π_T from the *start* state.[6] The "no punishment" condition ensures that only bounded, positive incentives are provided, which seems quite fitting in many of the web domains that motivate this work.[7] We focus primarily on finding admissible

[5]The general model allows admissibility conditions to be defined over a set of decisions, but here we define it with respect to a single decision π_T. Given that the interested party's goal is to induce a single target policy, it is reasonable to assume that he would only be interested in discovering and deploying incentives Δ that strictly induce π_T and are admissible with respect to π_T.

[6]The use of a single start state is without loss of generality, since it can be a dummy state whose transitions represent a distribution over possible start states.

[7]Alternative definitions of admissibility are possible as well. Our methods are not specific to a particular admissibility definition, so we will not pursue the issue further.

incentives to elicit the desired policy quickly, and only consider minimizing cost as a secondary objective.

6.3.2 The Known Rewards Case

To develop intuition, we first consider the static formulation in which the interested party knows the agent's reward function. The policy teaching problem is to find minimal admissible incentives that induce the desired policy π_T. To capture the space of rewards that are consistent with a particular policy, we first define the concept of *inverse reinforcement learning* (IRL) [68]:

Definition 6.3. *Given a policy π and $M_{-R} = \{S, A, P, \gamma\}$, let $\{R : R \in IRL^{\pi}\}$ denote the set of reward functions for which π is optimal for the MDP $M = \{S, A, R, P, \gamma\}$. Furthermore, for $\epsilon > 0$, let $\{R : R \in IRL^{\pi}_{\epsilon}\}$ denote the set of rewards for which π is uniquely optimal for M by a slack of at least ϵ, such that $Q^{\pi}(s, \pi(s)) - Q^{\pi}(s, a) \geq \epsilon$ for all $s \in S$, $a \in A \backslash \pi(s)$.*

The policy teaching problem then aims to find incentives leading to a reward function that is consistent with the desired policy:

Definition 6.4. ***Policy teaching with known rewards.*** *Given an agent MDP $M = \{S, A, R, P, \gamma\}$, target policy π_T, incentive limits D_{max} and Δ_{max}, and $\epsilon > 0$, if there exists admissible Δ such that $(R + \Delta) \in IRL^{\pi_T}_{\epsilon}$, find such a Δ to minimize $V^{\pi_T}_{\Delta}(start)$.*

The definition requires that the provided incentives strictly induce the desired policy. This avoids scenarios in which an agent is indifferent among multiple optimal

policies and may choose a policy other than that which is desired by the interested party.

To solve this problem, we need to (1) locate the space of reward functions under which π_T is uniquely optimal and (2) find an admissible incentive Δ that maps the agent's reward into this space. We apply a well-known result from inverse reinforcement learning, which shows that the space of rewards consistent with a particular (uniquely) optimal policy is given by a set of linear constraints:

Theorem 6.1. (Ng and Russell [68]) *Given a policy π written as $\pi(s) \equiv a_1$ and $M_{-R} = \{S, A, P, \gamma\}$, $R \in IRL^{\pi}$ satisfies:*

$$(\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_{a_1}})^{-1}\mathbf{R} \succeq \mathbf{0} \quad \forall a \in A \backslash a_1 \tag{6.5}$$

Furthermore, for $\epsilon > 0$, $R \in IRL^{\pi}_{\epsilon}$ satisfies:

$$(\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_{a_1}})^{-1}\mathbf{R} \succeq \epsilon \quad \forall a \in A \backslash a_1 \tag{6.6}$$

where $\mathbf{P_a}$ is the transition function with respect to action a written in matrix form, $\mathbf{R}$ is the reward function written in matrix form, and $\mathbf{I}$ is the identity matrix.

This theorem leads directly to our first result:

Theorem 6.2. *The following linear program solves policy teaching with known rewards:*

$$\min_{\Delta} V^{\pi_T}_{\Delta}(start) \tag{6.7}$$

$$R_T(s) - \Delta(s) = R(s) \qquad \forall s \tag{6.8}$$

$$((\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_{a_1}})^{-1}\mathbf{R_T})[\mathbf{s}] \succeq \epsilon \qquad \forall s, a \in A \backslash a_1 \tag{6.9}$$

$$\Delta \in admissible(\pi_T) \tag{6.10}$$

where $a_1 \equiv \pi_T(s)$ denotes the actions of the target policy, $\mathbf{P_a}$ is the transition function with respect to action a written in matrix form, and $\mathbf{R_T}$ is a reward function that strictly induces π_T written in matrix form.

6.3.3 The Unknown Rewards Case

In most situations, the interested party will not know the agent's reward function. This leads to the following problem definition:

Definition 6.5. ***Policy teaching with unknown agent reward.*** *Consider an agent following a policy π with respect to an MDP $M = \{S, A, R, P, \gamma\}$. An interested party observes the agent's policy, and knows $M_{-R} = \{S, A, P, \gamma\}$ but not R. Given target policy π_T, incentive limits D_{max} and Δ_{max}, and $\epsilon > 0$, if there exists an admissible Δ for which $(R + \Delta) \in IRL_\epsilon^{\pi_T}$, find an admissible Δ and observe agent policy π' such that $\pi' = \pi_T$ after few interactions.*

We assume that direct queries about the agent's preferences are unavailable and that preference information must be inferred from observations of agent behavior. This is often true on the Web. While firms such as Amazon and Facebook can observe user actions, it may be considered intrusive for them to directly ask their users for preference information. Doing so may disrupt from the user experience, and users may question their motives.

We develop an algorithm based on the active, indirect elicitation framework, wherein the space of potential agent rewards is narrowed by drawing additional IRL constraints based on observations of agent behavior in response to provided incentives. We assume the agent's reward function is bounded in absolute value by R_{max}

in every state. Within these bounds, we maintain an "IRL space" of reward functions that are consistent with observations *and* that have associated admissible incentive functions that can strictly induce the desired policy with some minimal slack $\epsilon > 0$.

At every iteration, the *elicitation function* makes a guess $\widehat{R}$ at the agent's true reward by choosing a point in the IRL space. If the guess is correct, providing the associated incentives $\widehat{\Delta}$ will strictly induce π_T. If instead the agent performs a policy $\pi' \neq \pi_T$, we know that $\widehat{R}$ must not be the agent's true reward R. Furthermore, we know that $R+\widehat{\Delta}$ induces π', which allows the *inference procedure* to add the following IRL constraints to the IRL space:

$$(\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_{a_1}})^{-1}(\mathbf{R} + \widehat{\mathbf{\Delta}}) \succeq \mathbf{0} \quad \forall a \in A \backslash a_1 \tag{6.11}$$

where $a_1 \equiv \pi'(s)$ denotes the actions of the observed policy, $\mathbf{P_a}$ is the transition function with respect to action a written in matrix form, $\widehat{\Delta}$ is the incentive provided, and $\mathbf{R}$ is the agent's reward function written in matrix form.

IRL constraints contain $|S||A|$ constraints on R and restrict the space of possible rewards to the intersection of the previous IRL space and the convex polytope implied by the added constraints. Since we are only interested in the agent's reward for the purpose of solving the policy teaching problem, we can stop the elicitation process as soon as we observe the desired policy or as soon as the IRL space becomes empty (declaring the problem impossible).

We use the following notation. All constraints are added to a constraint set K, such that instantiations of variables satisfy all constraints in K. An instantiation of a variable R is denoted as $\widehat{R}$. Algorithm 6.1 gives the elicitation method.

Algorithm 6.1 Active indirect elicitation for policy teaching

Consider agent policy π, desired policy π_T, $\epsilon > 0$
Variables R, R_T, Δ; constraint set $K = \emptyset$
Add $R \in \text{IRL}^{\pi}$, $|R(s)| \leq R_{max}\ \forall s \in S$ to K
Add $R_T \in \text{IRL}_{\epsilon}^{\pi_T}$, $\Delta = R_T - R$ to K
Add $\Delta \in admissible(\pi_T)$ to K
loop
    Find $\widehat{\Delta}$, $\widehat{R}$, $\widehat{R_T}$ satisfying all constraints in K
    if no such values exist **then**
        return FAILURE
    else
        Provide agent with incentive $\widehat{\Delta}$
        Observe π'
        if $\pi' = \pi_T$ **then**
            return $\widehat{\Delta}$
        else
            Add $(R + \widehat{\Delta}) \in \text{IRL}^{\pi'}$ to K

Theorem 6.3. *Algorithm 6.1 terminates in a finite number of steps with a solution to the policy teaching problem with unknown rewards or returns FAILURE if no solution exists, regardless of the elicitation function's choice of $\widehat{R}$ and $\widehat{\Delta}$ from K.*

Proof. (sketch) The minimal slack ϵ over the target policy ensures that all points within a closed hypercube of side length $\delta = \frac{\epsilon(1-\gamma)}{\gamma} - \kappa$ centered at $\widehat{R}$ are eliminated

by IRL constraints whenever π_T is not observed, for some arbitrarily small $\kappa > 0$.[8] Since the true reward is consistent with IRL constraints, by a pigeonhole argument, only a finite number of such hypercubes of eliminated points can fit in the IRL space before elicitation converges. □

While convergence is a desirable property, in practice the algorithm is only useful if it can induce the desired policy after few interactions. We develop an elicitation strategy that guarantees fast convergence and can be computed tractably.

A Centroid-based Approach

Consider the IRL space at any round of the elicitation process. Since this set of reward functions is characterized by linear constraints, it is convex. We can apply the following result on cutting convex sets:

Theorem 6.4. (Grünbaum [28]) *Any halfspace containing the centroid of a convex set in $\Re^{|S|}$ contains at least $\frac{1}{e}$ of its volume.*

By choosing the centroid of the IRL space of rewards for $\widehat{R}$, any added IRL constraint will cut off at least a constant fraction of the IRL space's volume:

Lemma 6.1. *Let B_K^t denote the IRL space of reward functions implied by the constraints in K before the t-th iteration of Algorithm 6.1. Let c_t denote the centroid of B_K^t. Consider an elicitation strategy that picks $\widehat{R} = c_t$ and any corresponding admissible $\widehat{\Delta}$ for which $(\widehat{R} + \widehat{\Delta}) \in IRL_\epsilon^{\pi_T}$. Providing $\widehat{\Delta}$ will either induce π_T, or lead to adding IRL constraints that eliminate at least $\frac{1}{e}$ of the volume of B_K^t, such that $vol(B_K^{t+1}) \leq (1 - \frac{1}{e})vol(B_K^t)$.*

[8] Throughout this section, a hypercube refers to a closed, axis-aligned hypercube.

Lemma 6.1 implies that after a number of iterations logarithmic in the volume of the IRL space, this volume can be made arbitrarily small. If we can provide conditions under which the desired policy is elicited before the volume of the IRL space falls below some threshold, we can guarantee logarithmic convergence.

One condition that leads to logarithmic convergence is to ensure that all points within a small hypercube centered at the true reward are contained in the initial IRL space and never removed by added IRL constraints (in cases where a solution exist). If points within this hypercube are chosen for $\widehat{R}$, the minimal slack over the target policy ensures that π_T is elicited. Assuming this condition is satisfied, we can stop the elicitation process after logarithmic rounds because we will either elicit the desired policy before the volume of the IRL space drops below the volume of the hypercube, or discover that the true agent reward must not be contained in the initial IRL space and thus there are no possible solutions.[9]

Unfortunately, Algorithm 6.1 may not satisfy this condition because IRL constraints may potentially eliminate some points in the small hypercube centered at the true reward R_{true}. For a reward guess $\widehat{R}$ and associated incentive $\widehat{\Delta}$ that does not induce the target policy, the observed policy π' will be optimal for R_{true} but need not be optimal for all reward functions in the hypercube centered at R_{true}.

Nevertheless, we can modify our current algorithm to ensure that a hypercube of points centered at R_{true} is never eliminated. Since Theorem 6.3 ensures that all points within a closed hypercube of side length δ centered at $\widehat{R}$ are eliminated by added IRL constraints, by convexity there exists a separating hyperplane between

[9]Bertsimas and Vempala [6] used this general observation to formulate an algorithm for finding a point in a convex set specified by a separation oracle with logarithmic queries.

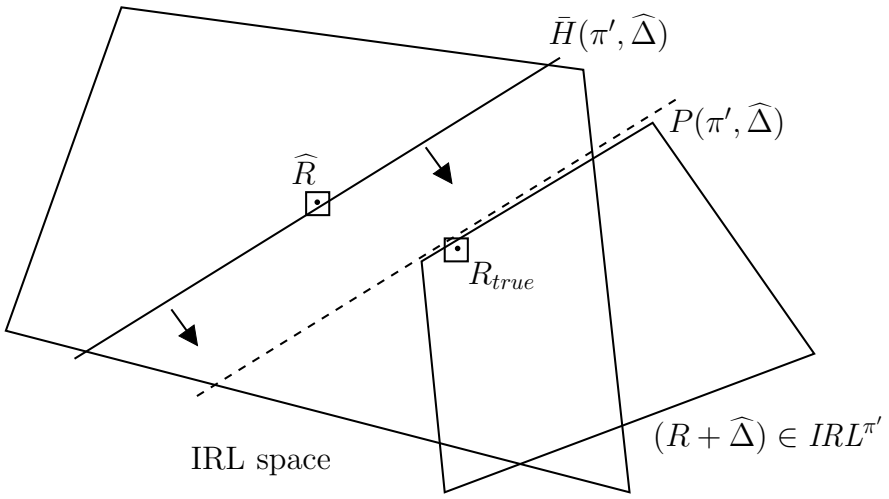

Figure 6.2: A condition that ensures logarithmic convergence requires maintaining a hypercube of points around the true reward R_{true} throughout the elicitation process. The larger polyhedron in the figure represents the IRL space of rewards that have yet to be falsified. Given an observation π' based on incentives $\widehat{\Delta}$, the IRL constraints $(R + \widehat{\Delta}) \in IRL^{\pi'}$ represented by the smaller polyhedron may eliminate some points within the hypercube of points centered at R_{true}. To avoid this, we find a separating hyperplane $P(\pi', \widehat{\Delta})$ between the hypercube centered at $\widehat{R}$ and the IRL constraints, and shift $P(\pi', \widehat{\Delta})$ towards $\widehat{R}$ until it is arbitrarily close to $\widehat{R}$. The resulting hyperplane $\bar{P}(\pi', \widehat{\Delta})$ separates $\widehat{R}$ and the hypercube centered at R_{true}. Adding the corresponding halfspace $\bar{H}(\pi', \widehat{\Delta})$ instead of the IRL constraints ensures logarithmic convergence.

this hypercube and the IRL constraints. Following Figure 6.2, let $P(\pi', \widehat{\Delta})$ be such a separating hyperplane, and let $\bar{P}(\pi', \widehat{\Delta})$ denote a hyperplane that results from relaxing $P(\pi', \widehat{\Delta})$ in the direction perpendicular to itself until it is arbitrarily close to $\widehat{R}$. Let $\bar{H}(\pi', \widehat{\Delta})$ be the halfspace not containing $\widehat{R}$ that is defined by $\bar{P}(\pi', \widehat{\Delta})$. Since $P(\pi', \widehat{\Delta})$ separates R_{true} from a hypercube of side length δ centered at $\widehat{R}$, $\bar{P}(\pi', \widehat{\Delta})$ will separate $\widehat{R}$ from a hypercube of side length δ centered at R_{true}. This ensures that finding $\bar{H}(\pi', \widehat{\Delta})$ and adding it instead of IRL constraints is a sufficient condition for guaranteeing logarithmic convergence.

Since the hypercube of points centered at $\widehat{R}$ and the IRL constraints are both characterized by linear constraints, we can find the separating hyperplane $P(\pi', \widehat{\Delta})$ by solving a simple linear program (e.g., see Theorem 10.4 in Vanderbei [93]). We can easily find $\bar{P}(\pi', \widehat{\Delta})$ by relaxing $P(\pi', \widehat{\Delta})$ until it almost passes through $\widehat{R}$, and define $\bar{H}(\pi', \widehat{\Delta})$ accordingly.

We define a modified version of Algorithm 6.1, denoted Algorithm 6.1*, where: (i) line 3 of Algorithm 6.1* adds $\bar{H}(\pi', \Phi)$ instead of $R \in \mathit{IRL}^{\pi}$ to K (where Φ corresponds to no environment change), (ii) Algorithm 6.1* returns FAILURE if it has not returned after $1 + |S|\lceil \log_b \lceil \frac{R_{max}}{\delta} \rceil \rceil$ rounds, where $b = \frac{1}{1-k}$ for some k such that $0 < k < \frac{1}{e}$, and (iii) given observed policy π' based on $\widehat{\Delta}$, Algorithm 6.1* does not add $(R + \widehat{\Delta}) \in \mathit{IRL}^{\pi'}$ to K and instead finds $\bar{H}(\pi', \widehat{\Delta})$ and add it to K.

Theorem 6.5. *Assume the agent's true reward is bounded by $R_{max} - \delta$ in every state, where $\delta = \frac{\epsilon(1-\gamma)}{\gamma} - \kappa$ for some arbitrarily small $\kappa > 0$. Let B_K^t denote the IRL space of reward functions implied by the constraints in K before the t-th iteration of Algorithm 6.1*, and let $b = \frac{1}{1-k}$ for some k such that $0 < k < \frac{1}{e}$. For any elicitation strategy that picks the centroid of B_K^t for $\widehat{R}$, Algorithm 6.1* terminates with a solution to the policy teaching problem with unknown rewards or returns FAILURE if no solution exists after at most $1 + \lceil \log_b \lceil (\frac{R_{max}}{\delta})^{|S|} \rceil \rceil$ iterations.*

Since the modifications to the algorithm allow us to eliminate the centroid of the IRL space while preserving a closed hypercube of points centered at the agent's true reward, the condition required for logarithmic convergence is satisfied and Theorem 6.5 follows. Here $(\frac{R_{max}}{\delta})^{|S|}$ is the number of non-overlapping hypercubes with side length δ that fit within the bounded space of rewards considered. This can be viewed

as the size of the elicitation problem, and the bound given by Theorem 6.5 is logarithmic in this dimension. This logarithmic bound is still linear in the number of states though, because only one of the constraints added at each iteration is guaranteed to cut off a constant fraction of the volume.

Although computing the centroid exactly is #P-hard [77], polynomial time, randomized algorithms exist and extend Grünbaum's result to the case of the approximate centroid. Bertsimas and Vempala [6] showed that any halfspace containing the average of $O(n)$ uniform samples from a convex set in $\Re^n$ will cut off a constant fraction of its volume with arbitrarily high probability. Using this result, we can construct an elicitation strategy that allows $\widehat{R}$ to be computed in polynomial time while guaranteeing logarithmic convergence with arbitrarily high probability:

Theorem 6.6. *Assume the agent's true reward is bounded by $R_{max} - \delta$ in every state. Let B_K^t denote the IRL space of reward functions implied by the constraints in K before the t-th iteration of Algorithm 6.1*, and let $b = \frac{1}{1-k}$ for some k such that $0 < k < \frac{1}{e}$. For any elicitation strategy that picks the average of $O(|S|)$ points sampled uniformly from B_K^t for $\widehat{R}$, with arbitrarily high probability, Algorithm 6.1* terminates with a solution to the policy teaching problem with unknown rewards or returns FAILURE if no solution exists after at most $1 + \lceil \log_b \lceil (\frac{R_{max}}{\delta})^{|S|} \rceil \rceil$ iterations.*

Theorem 6.7. *Each iteration of Algorithm 6.1* with the elicitation strategy from Theorem 6.6 is solvable in time polynomial in the number of states and actions.*

Sampling $O(|S|)$ points uniformly takes $O(|S|^4)$ steps of a random walk that requires $O(|S|^2)$ operations per step, so computing $\widehat{R}$ this way is $O(|S|^6)$ [6]. One can then find $\widehat{\Delta}$ satisfying $(\widehat{R} + \widehat{\Delta}) \in IRL_\epsilon^{\pi'}$ by solving a simple linear program.

6.3.4 Summary

We study the problem of policy teaching, in which the goal is to elicit a desired policy from an agent by providing rewards from a constrained space. Given unknown agent rewards, we constructed an algorithm that applies the active, indirect elicitation framework to quickly narrow down the space of possible rewards consistent with observed behavior. A centroid-based elicitation strategy guarantees convergence to a solution after few interactions, and is made tractable by applying appropriate sampling techniques.

Our analysis on policy teaching can be extended in a number of ways. Zhang et al. [108] considered a heuristic elicitation strategy based on maximizing the slack in IRL constraints. This approach does not provide logarithmic convergence guarantees, but is simpler (the elicitation strategy only requires solving a linear program), and achieved good empirical performance in simulation. Zhang et al. [108] also extended the elicitation algorithm to handle situations in which the interested party only observes the agent's actions instead of his policy, and in which the interested party only wishes to influence the agent's policy in a subset of the states.

Zhang and Parkes [107] considered the problem of *value-based policy teaching*, in which the goal is to provide limited rewards to elicit a policy that maximizes the interested party's value with respect to the *unknown* agent rewards. With this objective, computing the optimal incentives becomes NP-hard. The IRL space is no longer convex; while a similar active, indirect elicitation algorithm ensures convergence, logarithmic convergence cannot be guaranteed. Nevertheless, Zhang and Parkes [107] proposed a mixed-integer program for solving modest-sized instances, and presented

simulation results showing that slack-based elicitation heuristics were still effective and elicited the best possible policy after few interactions.

6.4 Discussion

Our solution to the policy teaching problem demonstrates how designing experiments by reasoning about participants based on current models, and learning about participants based on observed behaviors, can form an automated, iterative design process that effectively solves automated environment design problems. Using the active, indirect elicitation framework, the elicitation strategy sets up a hypothesis about an agent's model parameters, and designs an experiment that either produces a desired outcome (e.g., the agent follows the desired policy) or rejects the hypothesis. If the hypothesis is rejected, the inference procedure refines the knowledge of model parameters, to eliminate not only the particular parameter values being tested but any model parameters that are inconsistent with observed behavior. Zhang et al. [103] showed how to generalize Algorithm 6.1 and the centroid-based elicitation strategy for other automated environment design problems, and extended the theoretical results about logarithmic convergence to any setting with observable decisions for which the space of model parameters considered during the elicitation process is convex.

While we assumed in the policy teaching setting that the agent's decision or policy is directly observable through his actions, in practice we may only have access to samples of agent actions. This implies, for example, that we cannot always set up a hypothesis that directly proves that a particular set of parameter values is not the

agent's actual model parameters. In general, active, indirect elicitation algorithms may need to adopt a more probabilistic framework, where observed actions and outcomes are used to update beliefs over the underlying model parameters, but may never completely eliminate certain model parameters from consideration. As an example, Chapter 8 provides an active, indirect elicitation framework for automatically synthesizing crowdsourcing workflows, that adopts probabilistic beliefs.

Implicit in the active, indirect elicitation framework is the assumption that observations of behavior can be used to infer the agent's model parameters, and thus allow designers to better understand how participants make decisions based on which to more effectively design using learned models. In practice, models may be inaccurate and imprecise. The environment may be dynamic and involve changing factors that are outside of a designer's control but that nevertheless affect participant behavior. Some of these issues are explored in the next chapter, in which we consider an application of the active, indirect elicitation framework for automatically designing human computation tasks.

The active, indirect elicitation framework extends to settings with multiple participants, for which information about how participants may interact or affect one another's decisions can also be captured by an agent model and can likewise be refined by learning from observed behavior in response to well-chosen experiments. But with multiple participants there are new challenges, particularly in modeling the interaction among participants and how participants' individual actions can lead to complex outcomes. For example, a designer may need to reason about how the varied interests and abilities of participants can enable effective collaborative problem

solving, or reason about how network effects may affect the adoption of a new feature. Considering multiple agents also brings into focus a broader range of elicitation processes, which includes the ability to select particular groups of users on which to conduct an experiment.

While we assume that participants in social and economic systems on the Web are myopically rational with respect to environment changes, participants can be forward looking and take actions that aim to induce the designer to select more desirable environment changes. For example, such situations have been observed in traditional labor markets, in which paid for performance workers purposely reduced their output to prevent the employer from using output measures to infer their actual ability and increase quotas or reduce pay.[10] When this occurs, the interested party cannot make inferences based on observed actions under the assumption that agents are acting straightforwardly, because the revealed information may not truthfully represent the agent's model parameters.

In certain settings, the interested party may be able to avoid such issues by committing to a goal (e.g., eliciting behaviors leading to a goal value that is above a set threshold) and by only exploring environment changes that benefit both the agent and the interested party. The interested party may undertake an active, indirect elicitation process, and either discover an environment change under which the agent behaves as desired, or if not then give up and reset to the base environment. If the agent prefers a potential environment change over the base environment, he may nevertheless reveal sufficient information through actions to ensure that a change

[10]This is often referred to as the *ratchet effect* in economics, and occurs when an employer cannot commit to *not* using revealed information to exploit a worker over time.

benefitting both parties is made.

In general, handling such issues requires reasoning carefully about the incentives of participants and the interested party. Whenever possible, an automated environment design procedure should aim to discover designs that create additional value and benefit both parties, and in the process mitigate concerns about non-straightforward behavior. As an example in which we try to achieve this goal, we consider in the next chapter the problem of automatically designing a human computation task, where we seek to identify task designs that lead to higher quality output at a fixed unit rate of pay.