Planning with Participation Constraints

Hanrui Zhang[1], **Yu Cheng**[2], **Vincent Conitzer**[3]
[1] Carnegie Mellon University
[2] University of Illinois at Chicago
[3] Duke University
hanruiz1@cs.cmu.edu, yucheng2@uic.edu, conitzer@cs.duke.edu

Abstract

We pose and study the problem of planning in Markov decision processes (MDPs), subject to participation constraints as studied in mechanism design. In this problem, a planner must work with a self-interested agent on a given MDP. Each action in the MDP provides an immediate reward to the planner and a (possibly different) reward to the agent. The agent has no control in choosing the actions, but has the option to end the entire process at any time. The goal of the planner is to find a policy that maximizes her cumulative reward, taking into consideration the agent's ability to terminate.

We give a fully polynomial-time approximation scheme for this problem. En route, we present polynomial-time algorithms for computing (exact) optimal policies for important special cases of this problem, including when the time horizon is constant, or when the MDP exhibits a "definitive decisions" property. We illustrate our algorithms with two different game-theoretic applications: the problem of assigning rides in ride-sharing and the problem of designing screening policies. Our results imply efficient algorithms for computing (approximately) optimal policies in both applications.

1 Introduction

In this paper, we consider the general problem of planning in a Markov decision process (MDP), when the actions are *chosen* by the planner, but must be *taken* by an agent that is separate from the planner. The agent cannot choose the actions, but can refuse to participate at any time. To illustrate the setting, consider the following scenario. A ride-hailing company is designing a system to assign tasks to drivers. The company monitors the location (or *state*) of each driver in real-time via a mobile app. The company then has to choose which ride to assign to a driver (the company's choice of *action*), with each possible assignment resulting in some value to the company. If drivers had no incentives of their own (e.g., were self-driving vehicles belonging to the company), this could be modeled straightforwardly as an MDP. However, the driver also obtains rewards from the rides, which may be different. For example, drivers generally prefer not to take rides that will lead them to isolated areas,[1] or that are likely to involve troublesome passengers (e.g., pick-up at a bar at closing time). The company still wants to assign these rides, but does not want the driver to become sufficiently unhappy that they *sign off* (stop participating) altogether.

It is therefore crucial to take into consideration drivers' interests in planning task assignments, and provide enough incentives for drivers to remain in the system even when they receive less profitable or even unprofitable tasks. One intuitive solution is to promise drivers who receive undesirable tasks high priority for receiving more profitable tasks (or other perks) when they finish the current task. However, implementing this solution in a way that is most beneficial to the company can be challenging — incentivizing drivers and maximizing profit can be in conflict with each other, and it is not immediately clear how to balance them optimally.

In fact, the company faces a *planning* problem with *participation constraints*: the company as the planner (henceforth the *principal*) wants to compute and *commit to* a policy that maximizes her cumulative reward, subject to the constraint that each driver (henceforth the *agent*) must be motivated to participate in the policy, and not just at the beginning but also from any point onward.[2]

In addition to ride-hailing, many other real-world scenarios can be modeled as planning with participation constraints. We provide a few such examples:

- Financial services, such as automated investment platforms ("robo-advisors") or traditional mutual funds. In the case of robo-advisors, when a customer puts in money, the platform automatically creates and maintains a portfolio for the customer, and trades on behalf of the customer without asking for confirmation. The customer can see the trades and has the option to withdraw their money (i.e., to quit) at any time. The customer wants to make as much money as possible in a given amount of time, while the platform wants to keep the customer and optimize their own objective (which partially aligns with

[1] Sometimes drivers can see the destination; sometimes they cannot but may still suspect the destination based on the origin.

[2] Restricting the focus to policies under which no agent ever drops out is without loss of generality by a revelation principle: if the optimal policy involves an agent quitting at some point, since the principal knows when the agent would quit, she can use an equivalent policy where the agent is told that he will not have to do anything else (e.g., take any further rides) at that point. (Throughout the paper, we use "she" for the principal and "he" for the agent.)

the customer's goal, but may also include other considerations, such as charging an advisor fee).

- Distributed computing platforms (e.g., MQL 5, LoadTeam, or Golem Network) where an agent can sell CPU hours. Typically, such platforms assign tasks to machines based on how much task providers are willing to pay and how much computing power is required. Agents are paid an amount of money that depends on the task when a task is finished. The agents want to make more money, while the platform may want to finish as many tasks as possible. If the agent is receiving too little payment, they may decide to quit the platform and make use of their CPU hours in other ways (e.g., mining cryptocurrencies).
- Crowdsourcing marketplaces (e.g., Amazon Mechanical Turk). Workers are paid after they finish a task (e.g., completing surveys, labeling images). Workers can quit the platform or switch to another platform if they are not making enough money relatively. The platform wants to assign tasks (or a list of possible tasks to choose from) to workers and decide how much to pay to retain the workers, while optimizing their own objectives at the same time (e.g., charging a commission fee).

In this paper, we pose and study the computational problem of finding an (approximately) optimal policy in MDPs subject to participation constraints. We focus on the setting where agents have no private information, because in all the aforementioned applications, we have a very good first-order approximation of the agent's reward function. For example, ride-sharing drivers want to make more money in a given amount of time, and applicants going through a screening process would like to pass. (See Section 1.2 for more discussion on related work that considers private information.)

1.1 Our Results

Our main result is a fully polynomial-time approximation scheme (FPTAS) for planning with participation constraints.

- *We can compute a policy satisfying participation constraints and losing at most an additive* $\varepsilon > 0$ *in the principal's utility compared to the optimal policy. Moreover, we can compute such a policy in time* $\mathrm{poly}(m, n, 1/\varepsilon)$*, where* m *is the total number of actions and* n *is the number of states.*

Theorem 4 is a formal version of the above claim. We obtain this FPTAS by first designing an exponential-time algorithm (Algorithm 1 in Section 3.1) which computes an exact optimal policy, and then carefully discretizing the algorithm without violating participation constraints (Section 3.2).

When the MDP has some additional structures, our exact algorithm (Algorithm 1) in fact runs in polynomial time.

- *There is an exact algorithm (Algorithm 1) for planning with participation constraints that runs in time* $\mathrm{poly}(m_0^n)$*, where* m_0 *is the maximum number of actions available at a state and* n *is the number of states.*
- *When the time horizon is* H*, the exact algorithm runs in time* $\mathrm{poly}(m_0^H, n^H)$.
- *When the MDP has a "definitive decisions" property (see Definition 5 in Section 3.3), the exact algorithm runs in time* $\mathrm{poly}(m, n)$*, where* m *is total number of actions.*

The formal statements of these results appear as Lemma 3 in Section 3.1 and Lemma 6 in Section 3.3.

Some applications, such as the ride-hailing model that we gave above, are straightforward to fit into our framework. Consequently, the above results immediately imply efficient approximation algorithms for the ride assignment problem in ride-hailing, as well as efficient exact algorithms when the instance admits certain additional structures.

Our algorithmic framework in fact allows more and somewhat surprising applications. To illustrate this, we consider the design of optimal screening policies (Section 4), where the goal is to separate "good" candidates from "bad" ones by observing their performance in a series of tests. The more outcomes we observe, the more confident we are — but candidates may drop out if we ask them to take so many tests that the cost of taking tests outweighs the expected utility from passing the screening. By casting the problem as planning with participation constraints and applying our algorithms, we obtain the following results (formal statements in Theorem 8) for this problem.

- *We can compute an optimal screening policy that uses at most* N *tests in time* $\mathrm{poly}(N)$*. Moreover, an approximately optimal policy losing at most an additive* $\varepsilon > 0$ *in the principal's utility can be computed in time* $\mathrm{poly}(1/\varepsilon)$.

1.2 Related Work

Our results are along the line of research on *planning in Markov decision processes* (MDPs). The basic problem of computing optimal policies in MDPs has been extensively studied, and numerous efficient methods have been proposed, e.g., (Bellman 1957; Howard 1960; Puterman and Shin 1978). The problem studied in this paper is closely related to *constrained MDPs (CMDPs)* (see (Altman 1999) for a comprehensive survey), where the goal is to find a reward-maximizing policy subject to *overall* constraints, such as that the expected cumulative cost of the policy must not exceed a predefined threshold. It is known that CMDPs can be solved using linear programming (Altman and Spieksma 1995; Altman 1996, 1998). The key difference between our model and CMDPs is that we consider *uniform* participation constraints, which must hold in all states throughout the execution of the policy, rather than just in the initial state. In particular, such participation constraints may require any (nearly) optimal policy to be history-dependent, whereas in CMDPs, one can focus on Markovian (i.e., history-independent) policies without loss of generality (Altman 1999). More recently, reinforcement learning in CMDPs has received increasing attention (Achiam et al. 2017; Tessler, Mankowitz, and Mannor 2018; Cheung 2019; Le, Voloshin, and Yue 2019; Brantley et al. 2020; Efroni, Mannor, and Pirotta 2020; Singh, Gupta, and Shroff 2020; Ding et al. 2021). These results are quite different from ours, because they tend to focus on exploration rather than the planning problem itself, and the model we study in this paper is different from CMDPs. A related model is *multi-objective MDPs (MOMDPs)* (see (Roijers et al. 2013) for a comprehensive survey), where the planner cares about multiple objectives simultaneously. Similar to CMDPs, the work on MOMDPs focuses on the overall cumulative reward vector, whereas the participation constraints in our model must hold throughout the process. Another line of research considers *multi-agent (partially observable) MDPs*, where individual agents act in a common environment based only on local information and/or beliefs about each other (Gmytrasiewicz and Doshi 2005; Oliehoek 2012; Hoang and Low 2013). One key difference between our setting and the above is that we consider an asymmetric environment where the principal has the exclusive power to commit to a policy.

From an economic point of view, our results can be interpreted as *dynamic mechanism design* (see (Bergemann and Välimäki 2010; Athey and Segal 2013; Pavan, Segal, and Toikka 2014; Pavan 2017; Bergemann and Välimäki 2019)) under only individual-rationality (i.e., participation) constraints. Roughly speaking, in dynamic mechanism design, the principal controls which actions to play, but relies on self-interested agents to report the current state. Since the agents' reward functions may not align perfectly with the principal's, the main challenge in dynamic mechanism design is to encourage agents to truthfully report the state, while maximizing the planner's cumulative reward as much as possible. In contrast, in our model, the principal always observes the true state[3] but does not have the power to force participation. From a computational perspective, the fact that the principal can observe the true state enables efficient algorithms for long-horizon environments, which are known to be hard when the state must be reported by an agent (Zhang and Conitzer 2021).

2 Preliminaries

In this section, we give a formal definition of the problem of planning with participation constraints, which generalizes planning in MDPs. Recall that in standard (finite-time) MDPs, the state of the world evolves stochastically depending on the actions taken. Each state-action pair has a corresponding reward, and the goal of the planner is to choose actions in each state to maximize the total reward collected before reaching a terminal state. In our model, there are two entities, a *principal* (which plays a similar role as the planner in standard MDPs) and an *agent*. Each state-action pair generally gives different rewards to the principal and the agent. The principal controls which action to choose in each state, but the agent has the power to end the entire process at any moment. So, in order to gain more rewards, the principal has to commit to taking actions that incentivize the agent to remain in the process. Below we formally state the problem.

States, actions, rewards, and transitions. There is a *state* space $\mathcal{S}$, which contains an initial state s_{init} and a terminal state s_{term}.[4] For each state $s \in \mathcal{S}$, there is an associated set of *actions* $\mathcal{A}_s$, where $\mathcal{A}_{s_{\mathrm{term}}} = \varnothing$, i.e., no actions can be played at the terminal state. The principal's and the agent's *rewards* are captured by two reward functions $r_P : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $r_A : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. That is, in state $s \in \mathcal{S}$, taking action $a \in \mathcal{A}_s$ gives the principal a reward of $r_P(s, a)$ and the agent a reward of $r_A(s, a)$.

When action $a \in \mathcal{A}_s$ is played in state $s \in \mathcal{S}$, the *probabilistic transition* to the next state is given by a distribution $P(s, a) \in \Delta(\mathcal{S})$ over the state space, where $P(s, a, s')$ is the probability that the next state is $s' \in \mathcal{S}$. Throughout the paper, we assume the transition operator P is *acyclic*,[5] meaning that the probability of reaching the same state twice is 0 no matter what actions are played. In other words, if a state s can reach another state s' with positive probability, then s' cannot reach s. Consequently, we can label the states in topological order. That is, $\mathcal{S} = [n]$ [6] where $s_{\mathrm{init}} = 1$ and $s_{\mathrm{term}} = n$, such that all transitions go from a state with smaller index to one with larger index.

Histories and policies. A *history* of length t is a sequence of interleaved states and actions $(s_1, a_1, s_2, a_2, \dots, s_t, a_t)$, where $s_i \in \mathcal{S}$ and $a_i \in \mathcal{A}_{s_i}$ for each $i \in [t]$, and $s_i < s_{i+1}$ for each $i \in [t-1]$. Such a history corresponds to a (partial) run of the MDP, in which the initial state is s_1 and action a_1 is played, and the second state is s_2 and action a_2 is played, etc. Let $\mathcal{H}_t$ be the set of all histories of length t, and $\mathcal{H}_0 = \{\varnothing\}$ where $\varnothing$ denotes the empty history. Let $\mathcal{H} = \bigcup_{t \in [n-1]} \mathcal{H}_t \cup \mathcal{H}_0$ be the set of all histories, and let

$$\mathcal{H}_{<s} = \{(s_1, a_1, \dots, s_t, a_t) \in \mathcal{H} \mid s_t < s\}$$

be the set of histories visiting only states before s. For a history $h = (s_1, a_1, \dots, s_t, a_t)$ and a state-action pair (s, a), let $h + (s, a)$ be the history given by appending (s, a) to h:

$$h + (s, a) = (s_1, a_1, \dots, s_t, a_t, s, a).$$

We say a history h' *extends* another history h, if h is a prefix of h', with the notation $h' \supseteq h$,

A *policy* π is a (probabilistic) mapping from history-state pairs to distributions over actions. For any $h \in \mathcal{H}$, $s \in \mathcal{S}$, and $a \in \mathcal{A}_s$, $\pi(h, s, a)$ is the probability that action a will be chosen when state s is reached with history h. Note that we allow policies to be randomized and history-dependent, which is necessary when we later consider participation constraints. We always require that $\pi(h, s, a) = 0$ if $a \notin \mathcal{A}_s$.

Fixing a policy π, the utility (or value) function of the principal $u_P^\pi : \mathcal{H} \times \mathcal{S} \to \mathbb{R}$ can be defined recursively.

$$u_P^\pi(h, s) = \sum_{a \in \mathcal{A}_s} \pi(h, s, a) \Bigg(r_P(s, a) + \sum_{s' \in \mathcal{S}} P(s, a, s') \cdot u_P^\pi(h + (s, a), s') \Bigg),$$

[3] In our model, it is without loss of generality to have the principal choose only one action, rather than several actions for the agent to choose from. This is because the principal knows which (if any) action the agent would choose — since the agent has no private information — and hence the principal may as well give this action as the only option (mindful that the agent can refuse).

[4] We note that this is without loss of generality, i.e., equivalent to having a distribution of initial states and multiple terminal states.

[5] In particular, this subsumes all finite-horizon environments.

[6] Throughout the paper, we use $[n]$ to denote the set $\{1, \dots, n\}$.

and $u_P^\pi(h, s) = 0$ if $s = s_{\text{term}}$. Note that $u_P^\pi(h, s)$ is well-defined given that transitions are acyclic. In particular, $u_P^\pi(\varnothing, s_{\text{init}})$ is the principal's expected cumulative reward under policy π.[7]

Similarly, the agent's utility function u_A^π under policy π can be defined as

$$u_A^\pi(h, s) = \sum_{a \in \mathcal{A}_s} \pi(h, s, a) \left(r_A(s, a) \right.$$
$$\left. + \sum_{s' \in \mathcal{S}} P(s, a, s') \cdot u_A^\pi(h + (s, a), s') \right),$$

and $u_A^\pi(h, s) = 0$ if $s = s_{\text{term}}$.

The planning problem. We take the principal's point of view and aim to find a policy π that maximizes the principal's expected total utility $u_P^\pi(\varnothing, s_{\text{init}})$. Unlike planning in standard MDPs, we have the additional constraint that the policy should always incentivize the agent to remain in the process. That is, we only consider policies π that satisfy:

$$u_A^\pi(h, s) \geq 0, \text{ for all } h \in \mathcal{H} \text{ and } s \in \mathcal{S}.$$

This condition is commonly known as *individual-rationality (IR)* (or *voluntary participation*, or just *participation*) in economic theory, and captures the intuition that the agent should not be worse off by participating in the process. It is worth noting that, while our model as defined above only allows the agent to quit *before* an action is chosen (i.e., ex-ante IR), it is easy to extend the model to allow the agent to quit *after* seeing the randomly selected action (i.e., ex-post IR). This can be done by inserting a dummy state to represent "ready to take action a in state s".

We remark that it is sometimes better for the principal to let the agent quit, if the cost of keeping the agent in the system is too high. We capture this by adding a "quit" action for the principal in each non-terminal state, which gives both parties a reward of 0 and transitions the state of the world to s_{term}. Having added this action, imposing the IR constraint is without loss of generality: if our solution relied on the agent at some point refusing to participate further, we could instead just select the "quit" action for the agent at that point.

3 Computing Optimal Policies with Participation Constraints

In this section we present our main result: an algorithm that computes an ε-approximate optimal policy with participation constraints in time $\text{poly}(n, m, 1/\varepsilon)$, where $n = |\mathcal{S}|$ is the number of states, and $m = \sum_s |\mathcal{A}_s|$ is the total number of actions. We first present an exact algorithm, which runs in exponential time in the worst case, and then discretize the algorithm to allow for efficient approximation. Moreover, we show that the exact algorithm itself is strongly polynomial time if the input MDP satisfies a "definitive decisions" property (Definition 5).

Algorithm 1: Algorithm for computing the Pareto frontier of the principal/agent's utility for all states.

```
Input: State space $\mathcal{S}$, sets of actions $\{\mathcal{A}\}_s$, reward
       functions $r_P$ and $r_A$, transition operator $P$.
Output: Pareto frontier functions $\{f_s\}_s$ for all states.
let $f_n = \{(0, 0)\}$;
/* we use $\{(x, f(x))\}$ to represent $f$; all these
   functions are piecewise linear, so we only store the
   turning points. */
for $s = n - 1, \ldots, 1$ do
    initialize $f_s = \varnothing$;
    for $a \in \mathcal{A}_s$ do
        call Algorithm 2 and let
          $g_{s,a} \leftarrow \mathsf{subcurve}(s, a)$;
        /* $g_{s,a}$ is the Pareto frontier after taking
           action $a$ in state $s$ */
        let $g_{s,a} \leftarrow \{(x + r_A(s, a), y + r_P(s, a)) \mid$
          $(x, y) \in g_{s,a}\}$;
        /* shift $g_{s,a}$ by adding $(r_A(s, a), r_P(s, a))$ to
           all its turning points */
    let $\hat{f}_s \leftarrow \mathsf{concenv}(\{g_{s,a}\}_a)$;
    /* call concenv to compute the upper concave
       envelope of $\{g_{s,a}\}_a$; concenv can be any
       algorithm for concave envelope */
    let $f_s \leftarrow \hat{f}_s \cap \{(x, y) \mid x \geq 0, y \in \mathbb{R}\}$;
    /* truncate $\hat{f}_s$ at $x = 0$ and keep the nonnegative
       part as $f_s$ */
return $\{f_s\}_s$;
/* Algorithm 3 requires additional bookkeeping
   information from Algorithms 1 and 2; for ease of
   presentation, we do not explicitly return it */
```

3.1 An Exact Algorithm

Overview of Algorithm 1. Algorithm 1 has the same high-level structure as backward induction in standard (finite-horizon) MDPs. Recall that states are labeled in topological order. Without participation constraints, we can simply process the states backward, and recursively compute the optimal onward utility and the corresponding sub-policy in every state. However, in our problem, the algorithm has to keep track of the cumulative reward of the principal *and* the agent, and enforce the participation constraints.

We compute the *Pareto frontier* of the principal's and the agent's utility at every state. The Pareto frontier at s is a function f_s such that $f_s(u)$ is the principal's maximum achievable onward utility, subject to the agent's onward utility is exactly u (while satisfying onward participation constraints). We show that the functions $\{f_s\}_s$ can be computed efficiently in a recursive way, and consequently, the principal's optimal overall utility can be read off from the Pareto frontier in the initial state, and is equal to $\max_{u \geq 0} f_{s_{\text{init}}}(u)$.

Algorithmic insights. Algorithm 1 heavily exploits the fact that feasible policies (i.e., those ensuring participation) are closed under convex combination, and as a result, the Pareto frontier function at each state is concave. In each state s, we

[7] Because we consider finite-horizon environments, no discount factor is required; if a discount factor γ is nevertheless *desired*, we can add this into our model by introducing an additional probability of $1 - \gamma$ of immediately transitioning to the terminal state.

Algorithm 2: subcurve(s, a): algorithm for computing the subcurve for a state-action pair.

```
Input: State $s \in \mathcal{S}$, action $a \in \mathcal{A}_s$, transition
       operator $P$, Pareto frontier functions
       $\{f_{s'}\}_{s'>s}$ for downstream states $s'$.
Output: Subcurve for state-action pair $(s, a)$.
/* we represent each $f_{s'}$ and the subcurves by their
   turning points; let $f_{s'} = (f_{s',0}, \ldots, f_{s',k_{s'}})$ where
   $k_{s'}$ is the number of pieces of $f_{s'}$ and each
   $f_{s',i} = (x_{s',i}, y_{s',i})$ is a point in $\mathbb{R}^2$ */
let $p_0 \leftarrow \sum_{s'>s} P(s, a, s') \cdot f_{s',0}$;
let the source point of $p_0$ from each onward state $s'$
  be $f_{s',0}$;
/* the source points are for bookkeeping and are used
   in Algorithm 3 to execute the policy */
let $k \leftarrow \sum_{s'>s} k_{s'}$, and $c_{s'} \leftarrow 0$ for all $s' > s$;
/* $c_{s'}$ is the number of pieces of $f_{s'}$ that we have
   already used */
for $i = 1, \ldots, k$ do
    let $L_i \leftarrow \{s' \mid s < s' \leq n, c_{s'} < k_{s'}\}$;
    /* $L_1$ is the set of states left to consider */
    let $s_i \leftarrow \text{argmax}_{s' \in L_i}(y_{s',c_{s'}+1} -
      y_{s',c_{s'}})/(x_{s',c_{s'}+1} - x_{s',c_{s'}})$;
    let $p_i \leftarrow p_{i-1} + P(s, a, s_i) \cdot (x_{s',c_{s'}+1} -
      x_{s',c_{s'}}, y_{s',c_{s'}+1} - y_{s',c_{s'}})$;
    let $c_{s_i} \leftarrow c_{s_i} + 1$;
    let the source point of $p_i$ from each onward state
      $s'$ be $f_{s',c_{s'}}$;
return $(p_0, p_1, \ldots, p_k)$;
```

first compute, for each action a that can be taken at that state, the Pareto frontier $g_{s,a}$ after taking that action, and then shift each $g_{s,a}$ by the principal/agent's reward for the state-action pair (s, a). Then, since the policy can randomize among actions, we simply take the upper concave closure of all these shifted curves and then truncate it to ensure participation, which gives the Pareto frontier of this state.

A key subroutine of Algorithm 1 is Algorithm 2, which computes the Pareto frontier function $g(s, a)$ after taking action a in state s — we call this the *subcurve* for state-action pair (s, a). Intuitively, to evaluate this subcurve at some u, one needs to determine the optimal way of distributing the agent's utility u into onward states, such that the resulting principal's utility is maximized. Because the Pareto frontier functions of all onward states are concave, one can construct the entire subcurve using a water-filling procedure, where the construction always "follows" the Pareto frontier function that has the largest slope. In other words, if we want to increase the principal's utility by an infinitesimal amount after (s, a), we should increase the principal's utility (and decrease the agent's utility) in exactly one of the onward states, namely the onward state that provides the best tradeoff.

Implementing the optimal policy. Algorithm 1 solves only half of the problem — it computes the principal's optimal utility, but not the corresponding policy. In fact, it could be the case that any optimal policy requires exponential space to explicitly describe (even approximately), so it is impossible to output an optimal policy in polynomial time. Nevertheless, given the Pareto frontier functions constructed in Algorithm 1, we present an algorithm (Algorithm 3) that can efficiently execute an optimal policy on the fly. At a high level, while there are exponentially many possible runs under the optimal policy, to execute the policy we need to follow only the one that actually happened.

More specifically, the optimal policy chooses a point p_1 on the Pareto frontier of s_{init} to realize. We can backtrack how Algorithm 1 achieved p_1, which tells us to take an action (say a_1) and gives us a point $\{q_s\}$ on the Pareto frontier of each onward state s reachable from (s_{init}, a_1). We then play action a_1 and transition to one of these states (say s_2) at random, and we can continue to execute the point q_{s_2} on the Pareto frontier of s_2 using the same process as above.

Correctness of Algorithms 1 and 3. Now we are ready to prove the correctness of our algorithms.

Lemma 1. *Algorithm 1 computes the Pareto frontier functions for all states, i.e., for all $s \in \mathcal{S}$, $h \in \mathcal{H}_{<s}$, and $u \geq 0$,*[8]

$$f_s(u) = \max\{u_P^\pi(h, s) \mid \pi : \ u_A^\pi(h, s) = u \textit{ and } \\ \forall a \in \mathcal{A}_s, \forall s' > s, \forall h' \in \mathcal{H}_{<s'} \textit{ with } \\ h' \supseteq h + (s, a), u_A^\pi(h', s') \geq 0\}.$$

Lemma 2. *Let $(s_1 = s_{\text{init}}, a_1, s_2, a_2, \ldots, a_{t-1}, s_t = s_{\text{term}})$ be any (random) output sequence of Algorithm 3. Then for each $i \in [t-1]$,*

$$a_i \sim \pi^*(h_i, s_i),$$

where π^* is an optimal policy subject to participation constraints, and $h_i = (s_1, a_1, \ldots, s_{i-1}, a_{i-1})$. In particular, π^* guarantees that

$$u_A^{\pi^*}(h_i, s_i) = u_i \quad \textit{and} \quad u_P^{\pi^*}(h_i, s_i) = f_{s_i}(u_i),$$

where u_i is computed in line 1 or line 12 in Algorithm 3.

Runtime of Algorithm 1. We remark that Algorithm 1 may take exponential time in the worst case. In fact, let $|f_s|$ be the number of pieces in f_s (recall that all f_s are piecewise linear). Then for each $s \in \mathcal{S}$, $|f_s|$ can be as large as $\Theta\left(|\mathcal{A}_s| \cdot \sum_{s'>s} |f_{s'}|\right)$, so $|f_s|$ may grow exponentially as s decreases. Formally, we have the following result regarding the time complexity of Algorithm 1.

Lemma 3. *Algorithm 1 runs in time $n \cdot (2m_0)^n$, where $n = |\mathcal{S}|$ is the number of states and $m_0 = \max_{s \neq s_{\text{term}}} |\mathcal{A}_s|$ is the maximum number of actions available in a single state. Moreover, suppose the state space can be partitioned into H sets $\{\mathcal{S}_1, \ldots, \mathcal{S}_H\}$, such that for any two states $s \in \mathcal{S}_i$ and $s' \in \mathcal{S}_j$, for all $a \in \mathcal{A}_s$, $P(s, a, s') > 0$ only if $j = i + 1$. Then Algorithm 1 runs in time $O((m_0 n)^H \log n)$. In particular, Algorithm 1 is polynomial-time if H is constant.*

[8] Note that the equation of $f_s(u)$ implies that the policy satisfies IR constraints at histories it would never reach; this is without loss of generality as the "quit" action could always be chosen.

Algorithm 3: Algorithm for executing the optimal policy on the fly (i.e., choosing actions on the fly)

```
Input: Problem instance $(\mathcal{S}, \{\mathcal{A}\}_s, r_P, r_A, P)$,
       Pareto frontier functions $\{f_s\}_s$ and other
       auxiliary variables computed by Algorithm 1.
Output: A random trajectory
        $(s_1 = s_{\text{init}}, a_1, \ldots, a_{t-1}, s_t = s_{\text{term}})$ of the
        optimal policy subject to participation
        constraints.
/* we require the following bookkeeping
   information: the source points (lines 2 and 9 of
   Algorithm 2), the Pareto frontier functions before
   truncation $\{\hat{f}_s\}_s$ (line 7 of Algorithm 1), and
   "which turning points come from which subcurve"
   (lines 7 and 8 of Algorithm 1) */
let $i \leftarrow 1$, $u_1 \leftarrow \operatorname{argmax}_{u'} f_{s_{\text{init}}}(u')$, $s_1 \leftarrow s_{\text{init}}$;
while $s_i \neq s_{\text{term}}$ do
    let $p_i^-, p_i^+ \in \hat{f}_{s_i}$ and $\alpha_i \in [0, 1]$ be such that
      $(u_i, \hat{f}_{s_i}(u_i)) = \alpha_i \cdot p_i^- + (1 - \alpha_i) \cdot p_i^+$;
    /* we view $f_{s_i}$ and $\hat{f}_{s_i}$ interchangeably as
       mappings and sets of turning points; note that
       $p_i^-$ may have a negative $x$-coordinate, but $p_i$
       defined below always has a nonnegative
       $x$-coordinate, and is therefore on the curve of
       $f_{s_{i+1}}$ */
    let $a_i^-, a_i^+ \in \mathcal{A}_i$ be such that $p_i^-$ and $p_i^+$ come
      from the (shifted) subcurves for $(s_i, a_i^-)$ and
      $(s_i, a_i^+)$ respectively;
    let $\text{rand}_i$ be a uniformly random real number
      between 0 and 1;
    if $\text{rand}_i \leq \alpha_i$ then
        let $a_i \leftarrow a_i^-$,
          $p_i \leftarrow p_i^- - (r_A(s_i, a_i), r_P(s_i, a_i))$;
    else
        let $a_i \leftarrow a_i^+$,
          $p_i \leftarrow p_i^+ - (r_A(s_i, a_i), r_P(s_i, a_i))$;
    play action $a_i$ and transition to next state
      $s_{i+1} \sim P(s_i, a_i)$;
    let $q_i = (x_i, y_i)$ be the source point of $p_i$ from
      onward state $s_{i+1}$ in the subcurve (before
      shifting) for $(s_i, a_i)$;
    let $u_{i+1} \leftarrow x_i$, $i \leftarrow i + 1$;
let $t \leftarrow i$;
return $(s_1 = s_{\text{init}}, a_1, \ldots, s_{t-1}, a_{t-1}, s_t = s_{\text{term}})$;
```

3.2 Achieving Efficiency by Approximation

Below we show how to make Algorithm 1 polynomial-time by sacrificing an arbitrarily small additive error ε. The idea is simple: since each Pareto frontier function is concave, when the number of pieces in each function is large, there must be many pieces that are essentially the same and therefore can be merged at little cost.

To make this concrete, instead of the exact Pareto frontier curves, we maintain their intersections with equally spaced horizontal lines, and use the upper concave closures of these intersections as proxy curves to approximate the actual Pareto frontier functions. Since the Pareto frontier curves are concave, each curve can intersect each horizontal line at most twice — once in the increasing phase, and once in the decreasing phase. Moreover, the error from this approximation is uniformly upper bounded by the distance between two neighboring horizontal lines. Formally, for any desired error $\varepsilon > 0$, we insert the following operation after line 9 of Algorithm 1:

$$\boxed{\begin{array}{c}\text{let } f_s \leftarrow \mathsf{concenv}\big(f_s \cap \{(x, y) \mid x \in \mathbb{R}, y \in \\ \{-n, -n + \frac{\varepsilon}{n}, -n + \frac{2\varepsilon}{n}, \ldots, n - \frac{2\varepsilon}{n}, n - \frac{\varepsilon}{n}, n\}\}\big);\end{array}}$$

The approximation guarantee and the runtime of the modified algorithms are stated in the following theorem.

Theorem 4. *Suppose the principal's reward function is bounded, i.e., for all $s \in \mathcal{S}$ and $a \in \mathcal{A}_s$,*

$$-1 \leq r_P(s, a) \leq 1.$$

Then, the modified Algorithm 1 with parameter $\varepsilon > 0$ together with Algorithm 3 implicitly computes a policy π_ε guaranteeing participation, which satisfies

$$u_P^{\pi_\varepsilon}(\varnothing, s_{\text{init}}) \geq u_P^{\pi^*}(\varnothing, s_{\text{init}}) - \varepsilon,$$

where π^* is the optimal policy guaranteeing participation. Moreover, the modified Algorithm 1 with parameter $\varepsilon > 0$ takes time $O((mn^3 \log n)/\varepsilon)$.

3.3 Achieving Efficiency with Definitive Decisions

Although Algorithm 1 can take exponential time in general, we show that it is in fact strongly polynomial-time for a special case of the problem, where all decisions are definitive.

Formally, we consider the model with the following restriction.

Definition 5 (Definitive decisions)**.** *We say an MDP has the "definitive decisions" property if, for each state $s \in \mathcal{S}$, the set of available actions $\mathcal{A}_s$ at s includes at most one "decide-later" action, for which there are no restrictions on its transition probabilities; all other actions in $\mathcal{A}_s$ are "decision" actions, which immediately terminate the process by taking the state deterministically to s_{term}.*

Notice that there are no restrictions on the rewards, so each action can induce arbitrary rewards for the principal and the agent. MDPs with definitive decisions property can capture many real-world scenarios, such as candidate screening and training programs (see Section 4).

We can show that our exact algorithm (Algorithm 1) runs in polynomial time if the input MDP satisfies the definitive decisions property.

Lemma 6. *In the case of definitive decisions, Algorithm 1 computes the (exact) Pareto frontier functions for all states in time $O(m^2 \log n)$.*

4 Application: Screening Policy Design

In this section, we illustrate the wide applicability of our results by examining a natural application scenario: the design of screening policies. The setting we consider is the following. Suppose we run a crowdsourcing task that requires very specific qualifications, so we would like all crowd workers to be screened before entering the pool. Whether a worker actually qualifies can only be observed during the job (even workers themselves do not know beforehand), but we know that workers who qualify tend to perform better in certain simple online tests. In order to estimate workers' qualifications, we provide these tests to all prospective workers, so they can potentially take as many as they wish.

Ideally, we would like to admit only workers who take a large number of tests and perform well on average, because in that case we are very confident that they qualify. However, if the time cost of repeatedly taking tests outweighs the benefit of getting the job, then workers will simply not try at all. Even if a worker starts trying, he may quickly give up after seeing a few negative results, deciding that his chance of eventually qualifying does not justify the time cost of keep trying. Taking this into consideration, our goal is to design a policy of admission that in expectation maximizes our gain (i.e., the amount of useful job done minus the total payment) by encouraging workers to take more tests.

The setup. Formally, there are two types of workers: "good" and "bad". Before taking tests, neither we nor the worker know whether the worker is good or bad, but there are commonly known prior probabilities for both types, p_g and p_b. A test has two possible results: pass (P) and fail (F). A good worker passes each test independently with probability q_g, and a bad worker passes with probability $q_b < q_g$. Without loss of generality, suppose our utility for accepting a good agent is $u_g > 0$, and that for accepting a bad one is $u_b < 0$; each worker receives utility 1 for being accepted, and $-c$ for the time cost of taking a test.

A policy maps each sequence of test outcomes — e.g., "PFPPF" — to a distribution over three possible actions: accept, reject, and decide later. Our goal is to find an optimal policy, taking into consideration that workers may drop out at any moment if their onward expected utility is negative. We remark that our formulation and algorithms can in fact handle much more general settings, but we choose to present this specific application, because it admits nice structures and can demonstrate several intriguing aspects of our results.

4.1 Screening Policy Design as Planning with Participation Constraints

We show how to cast the screening problem as planning with participation constraints. First observe that at any moment in the above process, our belief on the worker's type (i.e., the posterior probability that a worker is good) only depends on the sequence of test outcomes so far. In fact, this probability depends only on the numbers of Ps and Fs that a worker gets. Therefore, without loss of generality, we can consider an MDP where the states are $\mathcal{S} = \{(p, f) \mid p, f \in \mathbb{N}\} \cup \{s_{\text{term}}\} = \mathbb{N}^2 \cup \{s_{\text{term}}\}$, where $s_{\text{init}} = (0, 0)$. The posterior probability of being good in a state (p, f) is given by:

$$\Pr[\text{good} \mid (p, f)] = \frac{p_g \cdot q_g^p (1 - q_g)^f}{p_g \cdot q_g^p (1 - q_g)^f + p_b \cdot q_b^p (1 - q_b)^f}.$$

The set of actions associated with each state s has 3 elements: $\mathcal{A}_s = \{\text{accept}, \text{reject}, \text{decide-later}\}$. Among these actions, accept and reject are decision actions, in the sense that they terminate the process immediately. The rewards they induce for the agent and the principal are: $(1, \Pr[\text{good} \mid (p, f)] \cdot (u_g - u_b) + u_b)$ for accept, and $(0, 0)$ for reject. The decide-later action induces rewards $(-c, 0)$ and transition probabilities:

$$\begin{aligned} \Pr[(p+1, f) \mid (p, f)] &= \Pr[\text{good} \mid (p, f)] \cdot q_g \\ &\quad + (1 - \Pr[\text{good} \mid (p, f)]) \cdot q_b, \\ \Pr[(p, f+1) \mid (p, f)] &= 1 - \Pr[(p+1, f) \mid (p, f)]. \end{aligned}$$

Computing optimal policies. With the above formulation, we can apply our algorithms to find optimal policies. First consider the case where there is an upper limit N on the number of tests each worker can take. In such cases, the state space becomes $\mathcal{S} = \{(p, f) \mid p + f \le N\} \cup \{s_{\text{term}}\}$, where $|\mathcal{S}| = O(N^2)$. Since the formulation has definitive decisions, we can directly apply Algorithm 1 to compute the optimal policy in time $O(N^4 \log N)$.

Even in the idealized setting where the number of tests a worker can take is arbitrarily large (which is possible because workers may not consider sunk cost when making decisions), we can still approximate the optimal policy to arbitrary precision. More specifically, the following lemma states that there exists a policy that uses at most N tests and achieves a principal's utility at most $O(1/N)$ worse than the optimal policy.

Lemma 7. *Let* OPT *be the principal's optimal utility in screening policy design with no limit on the number of tests. For any N, there is a policy guaranteeing participation which plays reject for all (p, f) where $p + f > N$, and achieves expected utility* $\text{OPT} - O(1/N)$ *for the principal.*

Combining Lemma 6 with Lemma 7, we can compute an ε-approximately optimal policy in time $O(\log(1/\varepsilon)/\varepsilon^8)$ for any $\varepsilon > 0$. To summarize, we have the following theorem.

Theorem 8. *We can compute an optimal screening policy in time $O(N^4 \log N)$ when the maximum number of tests a worker is willing to take is N. Moreover, when workers may take arbitrarily many tests, we can compute an approximately optimal policy up to any additive error $\varepsilon > 0$ in time* $O(\log(1/\varepsilon)/\varepsilon^8)$.

5 Future Directions

We leave a number of open questions as interesting avenues for future work: Is there a poly-time algorithm (or a hardness result) for exactly solving MDP with participation constraints? What if the policy must be deterministic? What additional applications can fit in the MDP model in this paper?

Another future direction is the case with incomplete information, e.g., when the agent's reward function is only approximately known. One potential solution is to run a robust version of our algorithm that allows some slack in the IR constraints, where the slack corresponds to a confidence interval for our estimate of the agent's reward function.

Acknowledgments

Hanrui Zhang and Vincent Conitzer are supported by NSF award IIS-1814056. Yu Cheng is supported in part by NSF award CCF-2122628. The authors thank anonymous reviewers for their helpful feedback.

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained policy optimization. In *International Conference on Machine Learning*, 22–31. PMLR.

Altman, E. 1996. Constrained Markov decision processes with total cost criteria: Occupation measures and primal LP. *Mathematical methods of operations research*, 43(1): 45–72.

Altman, E. 1998. Constrained Markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Mathematical methods of operations research*, 48(3): 387–417.

Altman, E. 1999. *Constrained Markov decision processes*, volume 7. CRC Press.

Altman, E.; and Spieksma, F. 1995. The linear program approach in multi-chain Markov decision processes revisited. *Zeitschrift für Operations Research*, 42(2): 169–188.

Athey, S.; and Segal, I. 2013. An efficient dynamic mechanism. *Econometrica*, 81(6): 2463–2485.

Bellman, R. 1957. A Markovian decision process. *Journal of mathematics and mechanics*, 6(5): 679–684.

Bergemann, D.; and Välimäki, J. 2010. The dynamic pivot mechanism. *Econometrica*, 78(2): 771–789.

Bergemann, D.; and Välimäki, J. 2019. Dynamic mechanism design: An introduction. *Journal of Economic Literature*, 57(2): 235–74.

Brantley, K.; Dudik, M.; Lykouris, T.; Miryoosefi, S.; Simchowitz, M.; Slivkins, A.; and Sun, W. 2020. Constrained episodic reinforcement learning in concave-convex and knapsack settings. *arXiv preprint arXiv:2006.05051*.

Cheung, W. C. 2019. Regret minimization for reinforcement learning with vectorial feedback and complex objectives. *Advances in Neural Information Processing Systems*, 32: 726–736.

Ding, D.; Wei, X.; Yang, Z.; Wang, Z.; and Jovanovic, M. 2021. Provably efficient safe exploration via primal-dual policy optimization. In *International Conference on Artificial Intelligence and Statistics*, 3304–3312. PMLR.

Efroni, Y.; Mannor, S.; and Pirotta, M. 2020. Exploration-exploitation in constrained MDPs. *arXiv preprint arXiv:2003.02189*.

Gmytrasiewicz, P. J.; and Doshi, P. 2005. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24: 49–79.

Hoang, T. N.; and Low, K. H. 2013. Interactive POMDP Lite: Towards practical planning to predict and exploit intentions for interacting with self-interested agents. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press.

Le, H.; Voloshin, C.; and Yue, Y. 2019. Batch policy learning under constraints. In *International Conference on Machine Learning*, 3703–3712. PMLR.

Oliehoek, F. A. 2012. Decentralized pomdps. In *Reinforcement Learning*, 471–503. Springer.

Pavan, A. 2017. Dynamic mechanism design: Robustness and endogenous types. In *Advances in Economics and Econometrics: Eleventh World Congress*, 1–62.

Pavan, A.; Segal, I.; and Toikka, J. 2014. Dynamic mechanism design: A myersonian approach. *Econometrica*, 82(2): 601–653.

Puterman, M. L.; and Shin, M. C. 1978. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11): 1127–1137.

Roijers, D. M.; Vamplew, P.; Whiteson, S.; and Dazeley, R. 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48: 67–113.

Singh, R.; Gupta, A.; and Shroff, N. B. 2020. Learning in Markov decision processes under constraints. *arXiv preprint arXiv:2002.12435*.

Tessler, C.; Mankowitz, D. J.; and Mannor, S. 2018. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074*.

Zhang, H.; and Conitzer, V. 2021. Automated Dynamic Mechanism Design. *arXiv preprint arXiv:2105.06008*.