Principal-Agent Reward Shaping in MDPs

Omer Ben-Porat[1], **Yishay Mansour**[2, 3], **Michal Moshkovitz**[4], **Boaz Taitler**[1]
[1]Technion—Israel Institute of Technology, Israel
[2]Tel Aviv University, Israel
[3]Google Research
[4]Bosch Center for Artificial Intelligence
omerbp@technion.ac.il, mansour.yishay@gmail.com, michal.moshkovitz@mail.huji.ac.il, boaztaitler@campus.technion.ac.il

Abstract

Principal-agent problems arise when one party acts on behalf of another, leading to conflicts of interest. The economic literature has extensively studied principal-agent problems, and recent work has extended this to more complex scenarios such as Markov Decision Processes (MDPs). In this paper, we further explore this line of research by investigating how reward shaping under budget constraints can improve the principal's utility. We study a two-player Stackelberg game where the principal and the agent have different reward functions, and the agent chooses an MDP policy for both players. The principal offers an additional reward to the agent, and the agent picks their policy selfishly to maximize their reward, which is the sum of the original and the offered reward. Our results establish the NP-hardness of the problem and offer polynomial approximation algorithms for two classes of instances: Stochastic trees and deterministic decision processes with a finite horizon.

1 Introduction

The situation in which one party makes decisions on behalf of another party is common. For instance, in the context of investment management, an investor may hire a portfolio manager to manage their investment portfolio with the objective of maximizing returns. However, the portfolio manager may also have their own preferences or incentives, such as seeking to minimize risk or maximizing their own compensation, which may not align with the investor's goals. This conflict is a classic example of principal-agent problems, extensively investigated by economists since the 1970s (see, e.g., (Holmström 1979; Laffont 2003). The fundamental question in this line of work is how the principal should act to mitigate incentive misalignment and achieve their objectives (Zhang and Zenios 2008; Gan et al. 2022; Ross 1973; Zhuang and Hadfield-Menell 2020; Hadfield-Menell and Hadfield 2019; Xiao et al. 2020; Ho, Slivkins, and Vaughan 2014).

While the literature on principal-agent problems is vast, modern applications present new challenges in which recommendation systems are principals and their users are agents. To illustrate, consider a navigation app like Waze. While the app's primary function is to provide users with the fastest route to their destination, it also has internal objectives that may not align with those of its users. For example, the app may incentivize users to explore less frequently used roads or to drive near locations that have paid for advertising. Additionally, the app relies on user reports to identify incidents on roads, but users can be reluctant to report. In this scenario, the navigation app is the principal, and it can incentivize users to act in ways that align with its objectives through gamification or by offering coupons for advertisers' stores, among other strategies. The transition of the agent in the space and therefore the principal-agent interaction can be modeled as a Markov decision process (MDP). While some works consider principal-agent problems over MDPs (Zhang and Parkes 2008; Yu and Ho 2022; Zhang, Cheng, and Conitzer 2022a,b), the setting remains underexplored. The fundamental question of mitigating misalignment in MDP environments warrants additional research.

In this paper, we contribute to the study of this challenging setting. Specifically, we model this setting as a Stackelberg game between two players, Principal and Agent,[1] over a joint MDP. Principal and Agent each have a unique reward function, denoted R^P and R^A, respectively, which maps states and actions to instantaneous rewards. Principal receives rewards based on Agent's decision-making policy, and thus she seeks to incentivize Agent using a *bonus reward function* we denote R^B. We assume that Principal has a limited budget, modeled as a constraint on the norm of R^B. Agent is self-interested and seeks a policy that maximizes his own utility, which is the sum of his reward function R^A and the bonus reward function R^B offered by Principal. By offering bonus rewards to Agent, Principal motivates Agent to adopt a policy that aligns better with her (Principal's) objectives. The technical question we ask is *how should Principal structure the bonus function R^B to maximize her own utility given budget constraints?*

Our Contribution This paper is the first to propose efficient, near-optimal algorithms for the principal-agent reward shaping problem. This problem is considered in prior works and has shown to be NP-hard (see elaborated discussion in Subsection 1.1). Prior works therefore propose solutions ranging from mixed integer linear programming to

[1]For ease of exposition, third-person singular pronouns are "she" for Principal and "he" for Agent.

differentiable heuristics. In contrast, in this paper, we identify two broad classes of instances that, despite also being NP-hard, can be approximated efficiently.

The first class is MDPs with a tree layout, which we term *stochastic trees*. In stochastic trees, every state has exactly one parent state that leads to it, and several states can have the same parent. We emphasize that the dynamics are not deterministic, and given a state and an action, the next state is a distribution over the children of the state. Stochastic trees are well-suited for addressing real-world scenarios that involve sequential dependencies or hierarchical decision-making. For example, in supply chain management, upstream decisions like raw material procurement have cascading effects on downstream activities such as manufacturing and distribution. Likewise, in natural resource allocation, the initial extraction or harvesting choices create a pathway of subsequent decisions.

We devise Stochastic Trees principal-Agent Reward shaping algorithm (STAR), which is a fully polynomial-time approximation scheme. It uses a surprising *indifference* observation: Imagine two scenarios, one in which Principal grants no bonus, and another where Principal provides an *efficient* bonus, for a definition of efficiency that we describe in Subsection 2.3. In both cases, Agent gets the same utility if he best responds. This allows us to adopt a bottom-up dynamic programming approach (see Observation 1) and show that

Theorem 1 (Informal statement of Theorem 4)**.** *Let the underlying MDP be a k-ary tree of depth H and let $V_\star^P$ be the optimal utility of Principal's problem with budget B. Given any small positive constant α, our algorithm* STAR *guarantees a utility of at least $V_\star^P$ by using a budget of $B(1+\alpha)$ and its runtime is $O\left(|A||S|k(\frac{|S|}{\alpha})^3\right)$.*

The second class of problems is *finite-horizon deterministic decision processes* (DDP) (Castro 2020; Post and Ye 2015). Unlike stochastic trees, where uncertainty plays a central role, DDPs involve scenarios characterized by a clear cause-and-effect relationship between actions and outcomes. DDPs are suitable for many real-world applications, e.g., robotics and control systems that rely on planning. Importantly, the machinery we develop for stochastic trees fails here. We propose another technique that is based on approximating the Pareto frontier of all utility vectors. We devise the Deterministic Finite horizon principal-Agent Reward shaping algorithm (DFAR), and prove that

Theorem 2 (Informal statement of Theorem 5)**.** *Let the underlying MDP be a DDP with horizon H, and let ε be a small positive constant, $\varepsilon > 0$. Our algorithm* DFAR *has the following guarantees:*

- *(ε-discrete rewards) If the reward functions of Principal and Agent are multiples of ε,* DFAR *outputs an optimal solution.*
- *(general rewards) For general reward functions,* DFAR *requires $B + H\varepsilon$ budget to guarantee an additive $H\varepsilon$ approximation of Principal's best utility with budget B.*
- *(runtime) In both cases, executing* DFAR *takes runtime of $O(|S||A|H^2/\varepsilon \log(|A|H/\varepsilon))$.*

1.1 Related Work

Most related to this work are works on environment design and policy teaching (Zhang and Parkes 2008; Yu and Ho 2022; Zhang, Parkes, and Chen 2009). These works address scenarios in which the principal can incentivize the agent through an external budget and adopt the same model as we do. Zhang and Parkes (2008) propose a mixed integer linear programming to tackle this problem, while Yu and Ho (2022) modify the agent to have bounded rationality, thereby obtaining a continuous optimization problem. Additionally, Zhang, Parkes, and Chen (2009) employ the same model and study how to implement a predefined policy of the principal. Crucially, none of these works offers efficient approaches with provable approximation guarantees. In contrast, our approach targets instances where we can develop polynomial-time approximation algorithms.

More recent works on principal-agent interactions over MDPs include those of Zhang, Cheng, and Conitzer; Zhang, Cheng, and Conitzer (2022b; 2022a). They consider MDPs where the principal chooses the policy for both parties, but the agent can stop the execution at any time. They assume that each party has a different reward function, and thus the principal aims at finding a utility-maximizing policy with the constraint of a positive utility for the agent. In this paper, we assume the agent chooses the policy, and not the principal like in Zhang, Cheng, and Conitzer (2022a). In our work, the power of the principal is to *shape* the agent's reward under a limited budget to improve her utility; thus, the problems and treatment are different.

From a broader perspective, principal-agent problems have received significant attention (Laffont 2003). By and large, solutions are divided into monetary incentives (Zhang and Zenios 2008; Xiao et al. 2020; Ho, Slivkins, and Vaughan 2014) like contracts (Bolton and Dewatripont 2004; Dutting, Roughgarden, and Talgam-Cohen 2021) and non-monetary incentives (e.g., Bayesian persuasion (Wu et al. 2022; Kamenica 2019)). This work addresses the former, as we assume the principal can provide monetary rewards to the agent. The literature on monetary incentives in such problems addresses complex settings like dynamic interaction (Zhang and Zenios 2008; Battaglini 2005; Zhang and Conitzer 2021) and learning contracts adaptively (Ho, Slivkins, and Vaughan 2014), among others.

Our main optimization problem (that appears in Problem P1) work could be cast as both a constrained MDPs problem (Altman 1999; Xiao et al. 2019) and a Bi-level optimization problems (Stadie, Zhang, and Ba 2020; Wang, Wang, and Gong 2022; Hu et al. 2020; Chen et al. 2022, 2023; Chakraborty et al. 2023). In constrained MDP problems, the goal is to find a utility-maximizing policy under a global constraint. In Bi-level optimizations, the problem is typically decomposed into an inner optimization problem, the solution of which becomes a parameter for an outer optimization problem. However, our problem cannot benefit from conventional tools and techniques employed in Bi-level optimizations, as two infinitely close bonus allocations can result in arbitrarily different utilities for the principal.

Reward shaping (Ng, Harada, and Russell 1999; Wiewiora, Cottrell, and Elkan 2003; Devlin and Kudenko

2012; Grzes 2017) focuses on modifying the learner's reward function by incorporating domain knowledge. The main goals are accelerating the learning process and guiding the agent's exploration (Ng, Harada, and Russell 1999; Randløv and Alstrøm 1998; Devlin and Kudenko 2012; Hu et al. 2020). We also aim to shape a reward function but in a way that aligns incentives, which is overlooked in that line of work. Particularly, these works do not consider strategic interaction between two entities as we do. Other related works are papers on poisoning attacks (Banihashem et al. 2022; Rakhsha et al. 2020; Zhang, Parkes, and Chen 2009), wherein the designer aims to manipulate the environment to steer a learning agent from his originally optimal policy. Finally, our optimization problem also relates to Stacklberg games (Başar and Olsder 1998)) and inverse reinforcement learning (Arora and Doshi 2021).

2 Model

In this section, we present the model along with several properties. We begin by providing some background and notation on Markov decision processes (MDPs) (Sutton and Barto 2018; Mannor, Mansour, and Tamar 2022). An MDP is a tuple (S, A, P, R, H), where S is the state space, A is the action space where $A(s) \subseteq A$ is the subspace of actions available at state s. P is the transition function, $P : S \times A \times S \to [0, 1]$ and $\sum_{s' \in S} P(s, a, s') = 1$, which is the probability of reaching a state s' from a state s by acting a, and R is the (immediate) reward function, $R : S \times A \to \mathbb{R}$. H is the finite horizon, and we assume that there is a designated initial state s_0. A policy $\pi : S \to A$ is a mapping from states to actions.

Given an MDP (S, A, P, R, H) and a policy π, we let $V(\pi, S, A, P, R, H)$ denote the *expected utility* of π, which is the expected sum of immediate rewards; i.e., $V(\pi, S, A, P, R, H) = \mathbb{E}[\sum_{i=0}^{H-1} R(s_i, \pi(s_i))]$, where $s_{i+1} \sim P(s_i, \pi(s_i), \cdot)$. We also let $V_s(\pi, S, A, P, R, H)$ denote the reward in case we start from any state $s \in S$. For convenience, we denote the set of optimal policies by $\mathcal{A}(S, A, P, R, H) = \arg\max_\pi V(\pi, S, A, P, R, H)$. When the objects (S, A, P, H) are known from the context, we drop them and denote $V(\pi, R), V_s(\pi, R)$ and $\mathcal{A}(R)$. Finally, we adopt the Q function (see, e.g., (Wiering and Van Otterlo 2012)), defined as $Q^\pi(s, a, R) = R(s, a) + \sum_{s' \in S} P(s, a, s') V_{s'}(\pi, R)$. The Q function describes the utility from a state s when choosing action a and playing policy π afterward.

The Principal-Agent Reward Shaping MDPs problem (PARS-MDP) is a two-player sequential game between players that we term Agent and Principal. Formally, an instance of the PARS-MDP is a tuple $(S, A, P, R^A, R^P, H, B)$, where (S, A, P, H) are the standard ingredients of MDPs as we explain above. The additional ingredients of our model are the (immediate) reward functions R^A and R^P, representing the reward functions of Agent and Principal, respectively. These reward functions are typically different, reflecting the different goals and preferences of the two players. We assume that R^A and R^P are always bounded in the $[0, 1]$ interval. The last ingredient of our model is the budget B, $B \in \mathbb{R}_+$, which represents the total amount of resources available to Principal to distribute over the state-action space and is determined exogenously. The game is played sequentially:

1. Principal picks a *bonus reward function* R^B, where $R^B : S \times A \to \mathbb{R}_+$. Principal is restricted to non-negative bonus functions that satisfy the budget constraint, namely $R^B(s, a) \geq 0$ for every $s \in S, a \in A(s)$ and $\sum_{s,a} R^B(s, a) \leq B$.
2. Agent's strategy space is the set of deterministic policies that map S onto A. Agent receives the bonus function R^B on top of his standard reward R^A, and picks a policy as the best response to the modified reward $R^A + R^B$.

The policy Agent picks as a strategy determines Agent's and Principal's utility. Specifically, Agent's utility is the expected sum of rewards, with respect to $R^A + R^B$, received by following policy π in the MDP $(S, A, P, R^A + R^B, H)$, taken over the distribution of states and actions induced by policy π.[2] Principal's utility is the expected sum of rewards received by following the same policy π selected by Agent in the MDP (S, A, P, R^P, H). Here, the reward function used is Principal's own reward function R^P. Given a strategy profile (π, R^B), the utilities of Principal and Agent are $V(\pi, R^P)$ and $V(\pi, R^A + R^B)$, respectively. Both players wish to maximize their utilities.

2.1 PARS-MDP as an Optimization Problem

We propose describing PARS-MDP as Principal's optimization problem instead of a game formulation. Intuitively, Agent should best respond to the bonus function by using $\mathcal{A}(R^A + R^B) = \arg\max_\pi V(\pi, R^A + R^B)$. In case several policies maximize $V(\cdot, R^A + R^B)$, we break ties by assuming that Agent selects the one that most benefits Principal. Formally, $\pi = \arg\max_{\pi \in \mathcal{A}(R^A + R^B)} V(\pi, R^P)$. Notice that this is *with* loss of generality, although we discuss a straightforward remedy (see the appendix). Consequently, Agent's action is predictable, and the main challenge in computing equilibrium strategies is finding Principal's optimal action.

Next, we formulate the model as an optimization problem for Principal. Her goal is to choose the bonus function R^B such that the policy chosen by Agent, $\pi \in \mathcal{A}(R^A + R^B)$, maximizes Principal's utility. Formally,

$$\begin{aligned} &\max_{R^B} V(\pi, R^P) \\ &\sum_{s \in S, a \in A} R^B(s, a) \leq B \qquad \text{(P1)} \\ &R^B(s, a) \geq 0 \text{ for every } s \in S, a \in A(s) \\ &\pi \in \mathcal{A}(R^A + R^B) \end{aligned}$$

[2] While the bonus function alters the environment and may prompt Agent to adopt a different optimal policy, it does not guarantee that Agent will realize the full value of the additional utility offered through the bonus. This is akin to *money burning* (Hartline and Roughgarden 2008). Our machinery is also effective in case Principal's constraint in Problem P1 is on the *realized* budget. We discuss it further in the appendix.

Clearly, without R^B the optimization can be done efficiently, selecting an optimal policy for Agent that maximizes Principal's expected utility (namely, using the tie-breaking in favor of the Principal). When we introduce the variable R^B, the problem becomes computationally hard. Intuitively, we need to select both the bonus rewards R^B and Agent's best response policy π simultaneously. This correlation is at the core of the hardness; the following Theorem 3 shows that the problem is NP-hard.

Theorem 3. *Problem P1 is NP-hard.*

We sketch the proof of Theorem 3 in Example 2 below. The proof of this theorem, as well as other missing proofs of our formal statements, appear in the appendix.

2.2 Warmup Examples

To get the reader familiar with our notation and illustrate the setting, we present two examples.

Example 1. Consider the example illustrated in Figure 1a. The underlying MDP has an acyclic *layout*,[3] and the transition function is deterministic. The horizon is $H = 2$; therefore, the states s_3, s_4 and s_5 are *terminal*. At each non-terminal state, Agent chooses action from $\{left, right\}$. The rewards of Agent and Principal are colored (red for Agent, blue for Principal) and appear next to edges, which are pairs of (state, action). In this example, the state-action pairs of $(s_1, right)$ and $(s_2, left)$ share the same rewards and appear once in the figure. Furthermore, assume that the budget is limited to 1; i.e., $B = 1$.

Since the transitions are deterministic and so are Agent's policies, each policy corresponds to a path from s_0 to a leaf. For instance, the policy that always plays $left$ corresponds to the path s_0, s_1, s_3. It is thus convenient to have this equivalence in mind and consider paths instead of policies. The best path for Principal is $\tau^B = (s_0, s_2, s_5)$, with utilities of $V(\tau^B, R^P) = 5$ to Principal and $V(\tau^B, R^A) = 6$ to Agent. However, Agent has a better path: If he plays $\tau^A = (s_0, s_1, s_4)$, he gets $V(\tau^A, R^A) = 8$ while Principal gets $V(\tau^A, R^P) = 2$. Indeed, this is Agent's optimal path.

Assume Principal picks R^B such that $R^B(s_3, left) = 1$ and $R^B(s, a) = 0$ for every $s \in S \setminus \{s_3\}$ and $a \in A$. This is a valid bonus function since it satisfies the budget constraint. In this case, the path $\tau' = (s_0, s_1, s_3)$ generates Agent's utility of $V(\tau', R^A + R^B) = V(\tau', R^A) + V(\tau', R^B) = 7 + 1 = 8$. Furthermore, Principal's utility under τ' is $V(\tau', R^P) = 3.5$, which is better than her utility under Agent's default path, τ^A (recall $V(\tau^A, R^P) = 2$). Therefore, since Agent's optimal policies are τ^A, τ', our tie-breaking assumption from Subsection 2.1 suggests he plays τ'. In fact, the above R^B is the optimal solution to Principal's problem in Problem (P1).

Example 2. Consider the example illustrated in Figure 1b. From the initial state s_0, the system transitions uniformly at random to one of N gadgets and reaches the state s_i with probability $1/N$ for any $N \in \mathbb{N}$. In the gadget associated with s_i, Agent can choose deterministically whether to transition to the leaf state $s_{i,r}$ or $s_{i,l}$ by choosing $right$ or choosing $left$. The former results in a reward of zero for both players, while the latter yields a negative reward $-c_i$ for Agent and a positive reward v_i for Principal. Since all $left$ actions induce a negative reward to Agent, his default is always to play $right$. To incentivize Agent to play $left$ at s_i, Principal has to allocate her budget such that $R^B(s_i, left)$ is greater or equal to the loss Agent incurs, c_i. Hence, setting $R^B(s_i, left) = c_i$ suffices. Whenever $B < \sum_{i=1}^{N} c_i$, Principal must carefully decide which gadgets to allocate her budget. This example is a reduction from the Knapsack problem (see the proof of Theorem 3).

[3] We use the term layout to describe the underlying structure of the states, actions, and transition probabilities.

2.3 Implementable Policies

The following definition captures the set of feasible policies, namely policies that Principal can induce by picking a feasible bonus function.

Definition 1 (B-implementable policy)**.** *A policy π is B-implementable if there exists R^B such that $\sum_{s,a} R^B(s, a) \leq B$ and $\pi \in \mathcal{A}(R^A + R^B)$.*

For instance, in Example 1, a policy that induces the path $\tau' = (s_0, s_1, s_3)$ is 1-implementable but not $1/2$-implementable. We further highlight the minimal implementation of a policy.

Definition 2 (Minimal implementation)**.** *Let π be any B-implementable policy. We say that R^B is the minimal implementation of π if $\pi \in \mathcal{A}(R^A + R^B)$ and for every other bonus function R such that $\pi \in \mathcal{A}(R^A + R)$, it holds that $\sum_{s,a} R^B(s, a) \leq \sum_{s,a} R(s, a)$.*

Minimal implementations are budget efficient and refer to the minimal bonus that still incentivizes Agent to play π. Importantly, as we prove in the appendix, they always exist.

3 Stochastic Trees

In this section, we focus on instances of PARS-MDP that have a tree layout. To approximate the optimal reward, we propose the Stochastic Trees principal-Agent Reward shaping algorithm (STAR), which guarantees the optimal Principal utility assuming a budget of $B+\varepsilon|S|$. Here, B represents the original budget, $|S|$ is the number of states, and ε is a configurable discretization factor. Notably, STAR has a runtime complexity of $O(|A||S|k(B/\varepsilon)^3)$. Before proceeding, we assert that tree-based instances can still be computationally challenging even though they are a special case of the problem. To see this, recall that we used the shallow tree in Example 2 to prove the problem is NP-hard. Crucially, it is worth noting that the computational challenge does not stem from the tree structure itself but rather from the presence of randomness. In the case of *deterministic* trees, the trajectory of any policy starting from s_0 always leads to a leaf node. Consequently, the Principal's task simplifies to selecting a leaf node from at most $O(|S|)$ leaves.

To begin, we introduce some useful notations. We adopt the standard tree graph terminology of children and parents, where if there is $a \in A$ such that $P(s, a, s') > 0$, then s is a parent of s', and s' is a child of s. For convenience,

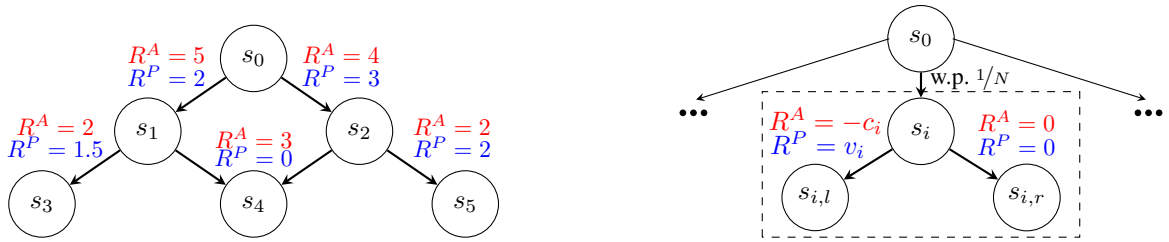

Figure 1: Instances for Examples 1 and 2. In both figures, Agent's (Principal's) reward is described in red (blue) next to each edge. Figure 1a describes an acyclic graph with deterministic transitions, and Figure 1b describes a stochastic tree of depth 2.

we define the set of children and parent states of state s as $Child(s)$ and $Parent(s)$, respectively. Since the depth of the tree is bounded by the horizon, we can use H to denote an upper bound on the tree's depth, which is defined as the longest path from state s_0 to a leaf state. Additionally, we denote (any arbitrary) optimal policy of Agent for $B = 0$, which we denote as $\pi^A \in \mathcal{A}(R^A)$. This policy represents the default actions of Agent when there are no bonus rewards.

We present STAR formally in the next subsection and sketch the high-level intuition here. It employs a dynamic programming approach, starting from leaf states and iterating toward the root s_0, while propagating almost all seemingly optimal partial solutions upstream. We use "almost", since it uses a form of discretization (recall that the problem is NP-hard). To explain why this dynamic programming is non-trivial, fix any arbitrary internal (not a leaf) state $s \in S$. Assume for the moment that Principal places no bonuses; hence, Agent chooses the action in s according to $\pi^A(s)$. Due to π^A's optimality w.r.t. R^A, we know that $\pi^A(s) \in \arg\max_{a \in A(s)} Q^{\pi^A}(s, a, R^A)$. To incentivize Agent to select an action $a' \in A(s), a' \neq \pi^A(s)$, Principal can allocate a bonus according to

$$R^B(s, a') = Q^{\pi^A}(s, \pi^A(s), R^A) - Q^{\pi^A}(s, a', R^A). \quad (1)$$

The term on the right-hand side of Equation (1), conventionally referred to as the *advantage function* and represented with a minus sign, plays a significant role in reinforcement learning (see, e.g., (Wiering and Van Otterlo 2012)). Note that this bonus includes only the (instantaneous) reward for the pair (s, a'). However, any dynamic programming procedure propagates partial bonus allocations; thus, when converting Agent from $\pi^A(s)$ to a', we must consider the bonus allocation in s's subtree. That is, to set $R^B(s, a')$ for converting Agent to playing a' at s, we should consider not only R^A but also any candidate bonus function R^B we propagate, and consider Agent's best response. This can result in a significant runtime blowup and budget waste due to discretization.

Fortunately, we can avoid this blowup. The next Observation 1 asserts that, under minimal implementation bonus functions, converting Agent to play another action in s at any state s is decoupled from the allocation at s's subtree.

Observation 1. *Let π be any B-implementable policy, and let R^B be its minimal implementation. For every $s \in S$, it holds that $V_s(\pi^A, R^A) = V_s(\pi, R^A + R^B)$.*

Algorithm 1: Stochastic Trees principal-Agent Reward shaping (STAR)

Input: $S, A, P, R^P, R^A, H, B, \varepsilon$
Output: R^B

1: let $\mathcal{B} = \{0, \varepsilon, 2\varepsilon, ..., B\}$
2: for every $s \in S$, $a \in A$, and $b \in \mathcal{B}$, set $U^P(s, b) \leftarrow 0$ and $U^P_a(s, b) \leftarrow 0$
3: $curr \leftarrow Leaves(S)$
4: **while** $curr \neq \emptyset$ **do**
5: pop $s \leftarrow curr$ with the highest depth
6: for every $a \in A(s)$, set $r(a) \leftarrow Q^{\pi^A}(s, \pi^A(s), R^A) - Q^{\pi^A}(s, a, R^A)$
7: for every $a \in A(s)$ and $b \in \mathcal{B}$, set $U^P_a(s, b) \leftarrow R^P(s, a) +$ OCBA(s, a, b)
8: for every $b \in \mathcal{B}$, set $U^P(s, b) \leftarrow \max_{a \in A(s)} \{U^P_a(s, b - \max_{\substack{r \in \mathcal{B} \\ r \leq r(a)}} \{r\})\}$
9: $curr \leftarrow curr \cup Parent(s)$
10: **end while**
11: extract R^B from U^P and U^P_a
12: **return** R^B

Observation 1 is non-intuitive at first glance. On the left-hand side, we have $V_s(\pi^A, R^A)$, which is what Agent gets in the absence of Principal and bonus rewards. On the right-hand side, $V_s(\pi, R^A + R^B)$ is the optimal utility of Agent when Principal picks R^B, which is a minimal implementation of the policy π. Equating the two terms suggests that by granting the bonus reward, Principal makes Agent indifferent between his default policy π^A and the B-implementable one; namely, the Agent's utility does not increase after placing the bonus reward. This is true recursively throughout the tree. Consequently, as long as we consider minimal implementations, we can use the instantaneous bonus proposed in Equation (1) to incentivize Agent to play a' at s regardless of the bonus allocation in s's subtree.

3.1 The STAR Algorithm

STAR is implemented in Algorithm 1. It gets the instance parameters as input, along with a discretization factor ε, and outputs an almost optimal bonus function R^B. Line 1 initializes the discrete set of bonuses $\mathcal{B}$, referred to as *budget units*. Line 2 initializes the variables U^P and U^P_a, which we

use to store partial optimal solutions for Principal. We then initialize $curr$ to $Leaves(S)$ in Line 3, where $Leaves(S)$ is the set of leaf states. The backward induction process is the while loop in Lines 4. Throughout the execution, $curr$ stores the states whose subtrees were already processed in previous iterations. We iterate while $curr$ is non-empty. Line 5, we pop a state s from $curr$. In Line 6, we compute for every action $a \in A(s)$ the minimal bonus needed to shift Agent from playing $\pi^A(s)$. According to Observation 1, this local deviation is decoupled from partial optimal solutions we computed for $Child(s)$ for minimal bonus functions.

Lines 7 and 8 are the heart of the dynamic programming process. In Line 7, we consider every action $a \in A(s)$ and every budget unit $b \in \mathcal{B}$. We set $U_a^P(s,b)$ to be the highest (expected) utility of Principal when starting from s, assuming Agent plays a and the budget is b. For that, we need to address two terms. The first term is the local $R^P(s,a)$, the reward Principal gets if Agents plays a at s. The second term in Line 7 is OCBA, which stands for Optimal Children Budget Allocation and is based on inductive computation. $\mathsf{OCBA}(s,a,b)$ is Principal's optimal utility if Agent plays action a in state s *and* we allocate the budget b optimally among the subtrees of $Child(s)$. $\mathsf{OCBA}(s,a,b)$ can be computed in $O(k(b/\varepsilon)^2)$ time via another (and different) dynamic programming process. Due to space limitations, we differ the implementation of OCBA for k-ary trees to the appendix and explain for binary trees. Let s_r, s_l be the right and left child of s, respectively. Then, $\mathsf{OCBA}(s,a,b)$ is

$$\max_{b' \in \mathcal{B}, b' \leq b} \left\{ P(s,a,s_r) U^P(s_r, b') + P(s,a,s_l) U^P(s_l, b-b') \right\}.$$

Namely, we consider all possible budget allocations between the subtrees of the children s_r and s_l, allocating b' to the former and $b-b'$ to the latter. The second step of the dynamic programming appears in Line 8, where we compute $U^P(s,b)$ inductively. Recall that the previously computed U_a^P assumed Agent plays a for $a \in A(s)$, but did not consider the required budget for that. To motivate Agent to play a, we need to allocate the bonus $r(a)$ to the pair (s,a) (recall Line 6). Hence, $U^P(s,b)$ is the maximum over U_a^P computed in the previous line, but considering that we must allocate $R^B(s,a) = r(a)$. Since we consider discrete budget, we assume we exhaust the minimal budget unit $r \in \mathcal{B}$ greater than $r(a)$, meaning we slash a slight budget portion in the inductive process.

After the inductive computation, Line 9 updates the set $curr$. Line 11 extracts the optimal bonus reward R^B from U^P and U_a^P. To that end, we use backtracking, which involves tracing the actions that led to the optimal value of U^P while considering the required bonus reward for Agent's deviations. The backtracking identifies the best sequence of actions and corresponding bonus rewards, which we formally claim later are the (approximately) best B-implementable policy and its corresponding minimal implementation. We end this subsection with the formal guarantees of STAR.

Theorem 4. *Let $I = (S, A, P, R^A, R^P, H, B)$ be a k-ary tree, and let $V_\star^P$ be the optimal solution for I. Further, let $\tilde{I}$ be the identical instance but with a budget $B + \varepsilon|S|$ for a small constant $\varepsilon > 0$, and let $\tilde{R}^B$ denote the output of $\mathsf{STAR}(\tilde{I})$. Then, executing $\mathsf{STAR}(\tilde{I})$ takes a run time of $O\left(|A||S|k(B/\varepsilon)^3\right)$, and its output $\tilde{R}^B$ satisfies $V(\pi, R^P) \geq V_\star^P$ for any $\pi \in \mathcal{A}(R^A + \tilde{R}^B)$.*

We note that STAR is a fully polynomial-time approximation scheme (FPTAS) despite the factor B in the runtime in Theorem 4. Let $\alpha > 0$ be any small constant. By setting $\varepsilon = \frac{B\alpha}{|S|}$, we use a budget of $B(1+\alpha)$ and the execution takes $O\left(|A||S|k(\frac{|S|}{\alpha})^3\right)$.

4 Deterministic Decision Processes with Finite Horizon

This section addresses PARS-MDP instances with a deterministic decision process layout and finite horizon. As we show in the appendix, this class of problems is still NP-hard. We propose the Deterministic Finite horizon principal-Agent Reward shaping algorithm (DFAR), which is implemented in Algorithm 2. In case R^A and R^P are ε-*discrete*, i.e., multiples of some small constant $\varepsilon > 0$, DFAR provides an optimal solution to Problem (P1) and runs in $O(|S||A|H^2/\varepsilon \log(|A|H/\varepsilon))$ time, where H is the horizon. As a corollary, we show that DFAR provides a bi-criteria approximation for general reward functions R^A and R^P. Namely, if the optimal solution for a budget B is $V_\star^P$, DFAR requires a budget of $B + H\varepsilon$ to guarantee a utility of at least $V_\star^P - H\varepsilon$.

For ease of readability, we limit our attention to acyclic DDPs and explain how to extend our results to cyclic graphs later in Subsection 4.2. Before presenting an approximation algorithm for this class of instances, we note that the STAR algorithm from the previous section is inappropriate for instances with an acyclic layout. One of the primary challenges in STAR is allocating the budget between children states for each action. In an acyclic layout, a state may have multiple parent states; therefore, if we follow the same technique for non-tree layouts, the STAR algorithm would assign multiple different budgets to the same state, one budget from each parent. As a result, a single state-action pair may have more than one bonus reward assigned to it. To handle acyclic layouts, we employ different techniques. To explain the intuition behind our algorithm, consider the set of all utility vectors $\mathcal{U}$, where Π is the set of all policies and $\mathcal{U} = \left\{ \left(V(\pi, R^A), V(\pi, R^P)\right) \in \mathbb{R}^2 \mid \pi \in \Pi \right\}$.

Every element in $\mathcal{U}$ is a two-dimensional vector, where the entries are the utilities of Agent and Principal, respectively. Ideally, we would like to find the best utility vector in $\mathcal{U}$: One that corresponds to a B-implementable policy and maximizes Principal's utility. However, we have two obstacles. First, constructing $\mathcal{U}$ is infeasible as its size can be exponential. We circumvent this by discretizing the reward functions of both players to be multiples of a small ε. Discretizing the reward function ensures that utilities will also be ε-discrete. Since H constitutes an upper bound on the highest utility (recall we assume R^A, R^P are bounded by 1 for every state-action pair), each player can have at most H/ε different utilities; thus, the ε-discrete set $\mathcal{U}$ can have at most $(H/\varepsilon)^2$ different vectors. The *Pareto frontier*, i.e., the set of Pareto efficient utilities, contains at most $2H/\varepsilon$. Our algorithm propagates the Pareto efficient utilities bottom-up.

The second obstacle is that the Pareto frontier we compute

Algorithm 2: Deterministic Finite horizon principal-Agent Reward shaping (DFAR)

Input: S, A, P, R^P, R^A, H, B, where R^P and R^A are assumed to be ε-discrete
Output: R^B

1: for every $s \in S$, set $U(s) \leftarrow \emptyset$
2: for all $s \in Terminal(S)$, let $U(s) \leftarrow \{(0,0)\}$
3: $S_{pass} \leftarrow Terminal(S)$
4: **while** $S_{pass} \neq S$ **do**
5: select a state $s \in Terminal(S \setminus S_{pass})$
6: for every $a \in A(s), s' \in Child(s,a)$ and $\mathbf{u} \in U(s')$, set $U(s) \leftarrow U(s) \cup \left((R^A(s,a), R^P(s,a)) + \mathbf{u}\right)$
7: $U(s) \leftarrow Pareto(U(s))$
8: $S_{pass} \leftarrow S_{pass} \cup \{s\}$
9: **end while**
10: let π such that $u^\pi \leftarrow \underset{\substack{u^\pi \in U(s_0) \\ V(\pi,R^A) \geq V(\pi^A,R^A)-B}}{\arg\max} V(\pi, R^P)$
11: extract R^B from π
12: **return** R^B

includes utility vectors corresponding to policies that are not B-implementable and thus infeasible. The following observation asserts that we can quickly distinguish utility vectors belonging to B-implementable policies.

Observation 2. *If the transition function is deterministic, then a policy π is B-implementable if and only if $V(\pi, R^A) \geq V(\pi_A, R^A) - B$.*

4.1 The DFAR Algorithm

The DFAR algorithm receives the instance parameters, where we assume R^A and R^P are ε-discrete for some constant $\varepsilon > 0$ (Corollary 1 explains how to relax this assumption), and outputs an *optimal* R^B. DFAR begins by initializing the set $U(s)$ to be the empty set for each state s in Line 1. In Line 2, we set $U(s)$ to include the zero vector for every terminal state. A state is *terminal* if it does not allow transitions to other states, and $Terminal(S')$ denotes the set of terminal states in the induced graph with states $S' \subseteq S$. Line 3 initializes S_{pass} to the set of $Terminal(S)$. Line 4 is a while loop that executes until S_{pass} contains all states.

In Line 5, we pick a state s we have not processed yet, namely, a terminal state of $S \setminus S_{pass}$. Due to the way we process states, $Child(s) \subseteq S_{pass}$. Further, since the graph is acyclic, such a state s must exist. In Line 6, we let $Child(s,a)$ denote the state we reach by acting $a \in A(s)$ in s (this state is unique since transitions are deterministic). Due to inductive arguments, $U(s')$ encompasses all attainable utilities when starting from state s', for every child $s' \in Child(s)$. In other words, every element in $U(s')$ has the form $(V_{s'}(\pi, R^A), V_{s'}(\pi, R^P))$ for some policy π. We update $U(s)$ to contain all vectors $(R^A(s,a), R^P(s,a)) + \mathbf{u}$ for $a \in A(s)$ and $\mathbf{u} \in U(s')$, where $s' = Child(s,a)$. After this update, elements in $U(s)$ represent the Q function vector $(Q^\pi(s,a,R^A), Q^\pi(s,a,R^P))$ for any $a \in A(s)$.

In Line 7, we remove Pareto inefficient utility vectors. We show in the appendix that this can be done in linearithmic time. Finalizing the while loop, Line 8 updates S_{pass}. By the time we reach Line 10, $U(s_0)$ contains all attainable utility vectors, including infeasible utility vectors that require a budget greater than B. Observation 2 assists in distinguishing utility vectors associated with B-implementable policies. We pick the utility vector u^π that maximizes Principal's utility, and work top-down to reconstruct the policy π that attains it. Finally, we reconstruct the minimal implementation R^B that induces π by setting the right-hand side of Equation (1) for every pair $(s, \pi(s))$. Next, we present the formal guarantees that DFAR provides.

Theorem 5. *Let $I = (S, A, P, R^A, R^P, H, B)$ be an acyclic and deterministic instance, and assume the reward function R^A and R^P are ε-discrete for a small constant $\varepsilon > 0$. Further, let $V^P_\star$ be the optimal solution for I. Then, executing DFAR(I) takes a run time of $O\left(\frac{|S||A|H}{\varepsilon} \log(\frac{|A|H}{\varepsilon})\right)$, and its output R^B satisfies $V(\pi, R^P) = V^P_\star$ for any $\pi \in \mathcal{A}(R^A + R^B)$.*

We also leverage Theorem 5 to treat reward functions that are not ε-discrete. According to Corollary 1 below, DFAR provides a bi-criteria approximation for general reward functions R^A and R^P. Namely, it requires a budget of $B + H\varepsilon$ to provide an additive approximation of $H\varepsilon$.

Corollary 1. *Let $I = (S, A, P, R^A, R^P, H, B)$ be an acyclic and deterministic instance, and let $V^P_\star$ be the optimal solution for I. Let $\tilde{I} = (S, A, P, \tilde{R}^A, \tilde{R}^P, H, B + H\varepsilon)$ be an instance with ε-discrete versions of R^A and R^P, $\tilde{R}^A$ and $\tilde{R}^P$, respectively, for a small constant ε. If DFAR$(\tilde{I})$ outputs $\tilde{R}^B$, then it holds that $V(\pi, R^P) \geq V^P_\star - H\varepsilon$ for any $\pi \in \mathcal{A}(R^A + \tilde{R}^B)$.*

4.2 Cyclic Deterministic Decision Processes

We develop DFAR assuming the underlying DDP has no cycles. In this subsection, we address the case of cyclic DDPs. Notice that any DDP with a finite horizon H can be cast as an *acyclic* layer graph with $|S| \cdot H$ states. Due to space limitations, we describe this construction formally in the appendix. Crucially, the resulting DDP is acyclic and has $|S|H$ states, compared to $|S|$ states in the original DDP. Consequently, every factor $|S|$ in the runtime guarantees of DFAR should be replaced with $|S|H$; thus, executing DFAR on the modified acyclic DDP instance takes $O(|S||A|H^2/\varepsilon \log(|A|H/\varepsilon))$.

5 Discussion

This paper introduces a novel approach to principal-agent modeling over MDPs, where the principal has a limited budget to shape the agent's reward. We propose two efficient algorithms designed for two broad problem classes: Stochastic trees and finite-horizon DDPs. We experimentally validate some of our theoretical findings via simulations in the appendix. Further, we also study several extensions that we relegate to the appendix in the interest of space. Future work can include better algorithms for our classes of instances as well as more general classes of instances. Another direction for future work is addressing learning scenarios, i.e., when the principal, agent or both have incomplete information.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882396), by the Israel Science Foundation, the Yandex Initiative for Machine Learning at Tel Aviv University and a grant from the Tel Aviv University Center for AI and Data Science (TAD).

References

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

Arora, S.; and Doshi, P. 2021. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297: 103500.

Banihashem, K.; Singla, A.; Gan, J.; and Radanovic, G. 2022. Admissible policy teaching through reward design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 6037–6045.

Başar, T.; and Olsder, G. J. 1998. *Dynamic noncooperative game theory*. SIAM.

Battaglini, M. 2005. Long-term contracting with Markovian consumers. *American Economic Review*, 95(3): 637–658.

Bolton, P.; and Dewatripont, M. 2004. *Contract theory*. MIT press.

Castro, P. S. 2020. Scalable methods for computing state similarity in deterministic markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 10069–10076.

Chakraborty, S.; Bedi, A. S.; Koppel, A.; Manocha, D.; Wang, H.; Huang, F.; and Wang, M. 2023. Aligning Agent Policy with Externalities: Reward Design via Bilevel RL. *arXiv preprint arXiv:2308.02585*.

Chen, S.; Wu, J.; Wu, Y.; and Yang, Z. 2023. Learning to Incentivize Information Acquisition: Proper Scoring Rules Meet Principal-Agent Model. *arXiv preprint arXiv:2303.08613*.

Chen, S.; Yang, D.; Li, J.; Wang, S.; Yang, Z.; and Wang, Z. 2022. Adaptive model design for Markov decision process. In *International Conference on Machine Learning*, 3679–3700. PMLR.

Devlin, S. M.; and Kudenko, D. 2012. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, 433–440. IFAAMAS.

Dutting, P.; Roughgarden, T.; and Talgam-Cohen, I. 2021. The complexity of contracts. *SIAM Journal on Computing*, 50(1): 211–254.

Gan, J.; Han, M.; Wu, J.; and Xu, H. 2022. Optimal Coordination in Generalized Principal-Agent Problems: A Revisit and Extensions. *arXiv preprint arXiv:2209.01146*.

Grzes, M. 2017. Reward shaping in episodic reinforcement learning. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*, 565–573. ACM.

Hadfield-Menell, D.; and Hadfield, G. K. 2019. Incomplete contracting and AI alignment. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 417–422.

Hartline, J. D.; and Roughgarden, T. 2008. Optimal mechanism design and money burning. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 75–84.

Ho, C.-J.; Slivkins, A.; and Vaughan, J. W. 2014. Adaptive contract design for crowdsourcing markets: Bandit algorithms for repeated principal-agent problems. In *Proceedings of the fifteenth ACM conference on Economics and computation*, 359–376.

Holmström, B. 1979. Moral hazard and observability. *The Bell journal of economics*, 74–91.

Hu, Y.; Wang, W.; Jia, H.; Wang, Y.; Chen, Y.; Hao, J.; Wu, F.; and Fan, C. 2020. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33: 15931–15941.

Kamenica, E. 2019. Bayesian persuasion and information design. *Annual Review of Economics*, 11: 249–272.

Laffont, J.-J. 2003. *The principal agent model*. Edward Elgar Publishing.

Mannor, S.; Mansour, Y.; and Tamar, A. 2022. *Reinforcement Learning: Foundations*. Online manuscript; https://sites.google.com/view/rlfoundations/home. Accessed March-05-2023.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, 278–287. Citeseer.

Post, I.; and Ye, Y. 2015. The simplex method is strongly polynomial for deterministic markov decision processes. *Mathematics of Operations Research*, 40(4): 859–868.

Rakhsha, A.; Radanovic, G.; Devidze, R.; Zhu, X.; and Singla, A. 2020. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *International Conference on Machine Learning*, 7974–7984. PMLR.

Randløv, J.; and Alstrøm, P. 1998. Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. In *ICML*, volume 98, 463–471.

Ross, S. A. 1973. The economic theory of agency: The principal's problem. *The American economic review*, 63(2): 134–139.

Stadie, B.; Zhang, L.; and Ba, J. 2020. Learning intrinsic rewards as a bi-level optimization problem. In *Conference on Uncertainty in Artificial Intelligence*, 111–120. PMLR.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Wang, L.; Wang, Z.; and Gong, Q. 2022. Bi-level Optimization Method for Automatic Reward Shaping of Reinforcement Learning. In *International Conference on Artificial Neural Networks*, 382–393. Springer.

Wiering, M. A.; and Van Otterlo, M. 2012. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3): 729.

Wiewiora, E.; Cottrell, G. W.; and Elkan, C. 2003. Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, 792–799.

Wu, J.; Zhang, Z.; Feng, Z.; Wang, Z.; Yang, Z.; Jordan, M. I.; and Xu, H. 2022. Markov Persuasion Processes and Reinforcement Learning. In *ACM Conference on Economics and Computation*.

Xiao, S.; Guo, L.; Jiang, Z.; Lv, L.; Chen, Y.; Zhu, J.; and Yang, S. 2019. Model-based constrained MDP for budget allocation in sequential incentive marketing. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 971–980.

Xiao, S.; Wang, Z.; Chen, M.; Tang, P.; and Yang, X. 2020. Optimal common contract with heterogeneous agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 7309–7316.

Yu, G.; and Ho, C.-J. 2022. Environment Design for Biased Decision Makers. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Zhang, H.; Cheng, Y.; and Conitzer, V. 2022a. Efficient Algorithms for Planning with Participation Constraints. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, 1121–1140.

Zhang, H.; Cheng, Y.; and Conitzer, V. 2022b. Planning with Participation Constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 5260–5267.

Zhang, H.; and Conitzer, V. 2021. Automated dynamic mechanism design. *Advances in Neural Information Processing Systems*, 34: 27785–27797.

Zhang, H.; and Parkes, D. C. 2008. Value-Based Policy Teaching with Active Indirect Elicitation. In *AAAI*, volume 8, 208–214.

Zhang, H.; Parkes, D. C.; and Chen, Y. 2009. Policy teaching through reward function learning. In *Proceedings of the 10th ACM conference on Electronic commerce*, 295–304.

Zhang, H.; and Zenios, S. 2008. A dynamic principal-agent model with hidden information: Sequential optimality through truthful state revelation. *Operations Research*, 56(3): 681–696.

Zhuang, S.; and Hadfield-Menell, D. 2020. Consequences of misaligned AI. *Advances in Neural Information Processing Systems*, 33: 15763–15773.