Approximating Solutions to Discrete State Space Markov Decision Processes in a Single Episode

Naveen Durvasula[†] and **Aravind Srinivasan**[‡] and **John P. Dickerson**[‡]
[†]Montgomery Blair High School [‡]University of Maryland
140.naveen.d@gmail.com, {srin,john}@cs.umd.edu

Abstract

Many optimization problems associated with decision making in reinforcement learning (RL) settings are naturally modeled via Markov decision processes (MDPs). In many settings, portions of the model are unknown (e.g., the state space, state-transition function, and so on). We consider one such constrained setting: an agent can "walk through" an MDP with known state and action space, but unknown transition and reward function, *exactly once*, and has to make transition decisions on the fly. We propose a principled Gaussian-Process-based approach to this single-episode control problem. Surprisingly, in exploratory experiments, our method performs well when compared to (more traditional, and substantially less-constrained) methods that can sample the transition and reward functions *multiple* times.

1 Introduction

In many real-world and virtual settings, agents learn by interacting with their environment. Autonomous vehicles interact first with high-fidelity simulators and later real-world environments to learn policies that map their perceived state to "successful" driving actions (Urmson et al. 2008). A biped robot learns how to lift itself off the ground and walk toward a door by way of physics-based simulation and, later, in a physical laboratory (Huang et al. 2001). Trading agents interact in strategic and uncertain environments so as to maximize their expected monetary reward (Hu, Wellman, and others 1998). The field of reinforcement learning (RL) addresses the problem of mapping states to actions such that some numerical reward-signal is maximized in expectation.

When the full model is known or can be learned via sampling, dynamic-programming-based approaches can compute optimal policies that map states to actions such that some cumulative expected reward is maximized, and scalable but approximate methods can be used when the number of states and/or actions grows. Model-free approaches do not explicitly build a representation of the underlying environment, but instead sample actions repeatedly to learn how to act in a state. Yet, in some settings, an agent must learn to act in an uncertain environment via only a *single* episode. For example, a graduate student wishes to maximize the reward from her Ph.D., but likely does not wish to repeatedly sample multiple Ph.D. processes. How should an agent act in such a "single-shot" constrained setting?

In this paper, we address the constrained setting where an agent can "walk through" a Markov decision process (MDP) with known state and action space, but with unknown transition and reward functions, *exactly once*, and has to make transition decisions on the fly. Total reward is accumulated throughout that single episode. We propose a principled approach based on Gaussian processes (GPs) to this single-episode control problem, and discuss its relation to model-based and model-free approaches to decision-making under uncertainty. Surprisingly, in exploratory experiments, our method performs well when compared to substantially less-constrained, and widely used, methods that can sample both the transition and reward functions *multiple* times.

The paper progresses as follows. Section 2 defines the environment in which we operate, and places our proposed model in the greater RL literature. Section 3 gives a high-level overview of our approach, and discusses the classic exploration-exploitation tradeoff in the context of our single-episode model, while Sections 4 and 5 define in depth our GP-based approach to learning a policy. Finally, Section 6 experimentally compares an implementation of our approach on a toy model to a less-constrained, state-of-the-art RL method.

2 Preliminaries

In this section, we formally define the "on-demand" environment in which we operate, as well as some mathematical constructs that will be referred to later on. We begin with a brief overview of the MDP paradigm (§2.1; for an in-depth overview, see (Sutton and Barto 1998; Kaelbling, Littman, and Moore 1996; Puterman 2014)). We then place our setting in a hierarchy of RL problems representable as MDPs, and compare briefly to model-based and model-free approaches to RL (§2.2).

2.1 Markov Decision Processes

We start with the definition of a *Markov decision process*, or MDP. An MDP $\mathbb{M}$ is defined by the tuple $(S, A, R_a(\cdot,\cdot), P_a(\cdot,\cdot))$, and consists of a Markov chain and an *agent* capable of traversing the chain. The nodes of the Markov chain are given by elements of S, otherwise known as the *state space*. On each time iteration, the agent, currently in some state $s \in S$, selects an action a from the *action space* A, transitions to another state $s' \in S$, and receives a real-valued "reward". For each potential future state s', the probability that the agent will transition from s to s' conditioned on the selection of action a is given by $P_a(s, s')$, the *state transition function*. Likewise, the reward earned from this transition step is given by $R_a(s, s')$, the *reward function*. The objective of the agent is to find a *policy function* $\pi : S \mapsto A$ such the expected total cumulative reward after T iterations

$$\mathrm{E}\left[\sum_{t=1}^{T} R_{\pi(s_t)}(s_t, s_{t+1})\right]$$

is maximized. For problems with an infinite time horizon, we introduce a parameter $\gamma \in (0, 1)$ which serves to prevent the cumulative reward from being infinite. In these cases, the expected *discounted* cumulative reward

$$\mathrm{E}\left[\sum_{t=1}^{T} \gamma^t R_{\pi(s_t)}(s_t, s_{t+1})\right]$$

is maximized. The parameter γ controls the "greediness" of the agent: a smaller γ will incentivize short-term gain. In other formulations, rather than defining a horizon, we define a set of *termination states* $S_\emptyset \subset S$. The agent must find a policy π such that the expected cumulative reward

$$\mathrm{E}\left[\sum_{s_{t+1} \notin S_\emptyset} R_{\pi(s_t)}(s_t, s_{t+1})\right]$$

is maximized.

2.2 Solvers and the Environment Poset

We now give structure to the set of all RL problems describable by a MDP and define a partial ordering of *hardness* over sets of problems. We use this structure to formally define general RL solvers. We then describe the *on-demand environment* and our solver, and discuss their relation to current literature.

We first define the set of all RL problems $\mathcal{P}$ describable by a MDP. We let the set of all MDPs given by the set of all tuples $(S, A, R_a(\cdot,\cdot), P_a(\cdot,\cdot))$ be $\mathcal{M}$. Intuitively, a problem $p \in \mathcal{P}$ consists of a MDP $\mathbb{M} \in \mathcal{M}$ and the set of elements in the tuple given by $\mathbb{M}$ that are known to the agent before its traversal. An *environment* is a tuple indicating the agent's level of knowledge of each of the elements of $\mathbb{M}$.

For each element $m \in \mathbb{M}$ for some MDP $\mathbb{M} \in \mathcal{M}$ we denote two levels of understanding:

1. the agent has complete knowledge of m: we denote this level of knowledge by m;
2. the agent has no knowledge of m: we denote this level of knowledge by $\emptyset$.

We make one exception—it is impossible for an agent to have no knowledge of the action space A. The agent's task is to construct a policy $\pi : S \mapsto A$. While the agent gains knowledge of S during its traversal, it is impossible for the agent to gain knowledge of A. If the set of actions that are visible to the agent is the empty set $\emptyset$, then it is impossible for the agent to construct a map $\pi : S \mapsto A$.

For the state transition function $P_a(\cdot,\cdot)$ and reward function $R_a(\cdot,\cdot)$, we define an intermediate stage of knowledge: being able to *sample* from the function—we denote this level of knowledge by P'_a or R'_a respectively. An agent has sampling access to P_a if it can draw from the distribution of future states conditioned on a given state and action. Likewise, an agent has sampling access to R_a if it can draw from the distribution of rewards conditioned on a given state and action. Note that the above requires an agent to have knowledge of S and A. Similarly, complete knowledge of $P_a(\cdot,\cdot)$ or $R_a(\cdot,\cdot)$ would imply complete knowledge of S and A.

With the above definitions, we define an environment $\mathbb{E}$ to be a length-four tuple indicating the amount of information known to the agent. For example, the environment (S, A, P'_a, R'_a) would indicate that the agent has access to the state and action spaces, but can only sample from the state transition function and reward function. The complete set of environments

$$\begin{aligned}
\mathcal{E} = \{&(\emptyset, A, \emptyset, \emptyset),\\
&(S, A, \emptyset, \emptyset),\\
&(S, A, P'_a, \emptyset), (S, A, \emptyset, R'_a),\\
&(S, A, P'_a, R'_a), (S, A, P_a, \emptyset), (S, A, \emptyset, R_a),\\
&(S, A, P_a, R'_a), (S, A, P'_a, R_a),\\
&(S, A, P_a, R_a)\}
\end{aligned} \tag{1}$$

is a graded poset with 6 rank levels. For some environments $\mathbb{E}, \mathbb{E}' \in \mathcal{E}$, we let $\mathbb{E} \succ \mathbb{E}'$ if and only if all elements of $\mathbb{E}$ are information states less than or equal to their corresponding counterparts in $\mathbb{E}'$, and there exists at least one element of $\mathbb{E}$ that is a strictly lesser information state than its counterpart in $\mathbb{E}'$. Each line of Equation (1) denotes one rank level of $\mathcal{E}$, with the "hardest" environment $(\emptyset, A, \emptyset, \emptyset)$ at rank 1. For some $\mathbb{E}, \mathbb{E}' \in \mathcal{E}$, we say that $\mathbb{E}$ is *harder* than $\mathbb{E}'$ if and only if $\mathbb{E} \succ \mathbb{E}'$.

We now formally define the set of problems $\mathcal{P} = \mathcal{M} \times \mathcal{E}$. For any problem $p \in \mathcal{P}$, we let $M : \mathcal{P} \mapsto \mathcal{M}$ denote the corresponding MDP and $E : \mathcal{P} \mapsto \mathcal{E}$ denote the corresponding environment. For all environments $\mathbb{E} \in \mathcal{E}$ we define the *environment class* $\mathbb{P}_\mathbb{E} = \{p \in \mathcal{P} \mid \mathbb{E} \succeq E(p)\}$.

We denote the set of all policies given a MDP $\mathbb{M} \in \mathcal{M}$ as $\Pi(\mathbb{M})$. We let the set of all policies (over all MDPs) be Π. A *solver* for an environment class $\mathbb{P}_\mathbb{E}$ is a map $\mathbb{S}_\mathbb{E} : \mathbb{P}_\mathbb{E} \mapsto \Pi$ such that $\forall p \in \mathbb{P}_\mathbb{E}, \mathbb{S}_\mathbb{E}(p) \in \Pi(M(p))$. We denote the set of all possible solvers, or *solver class*, for some environment class $\mathbb{P}_\mathbb{E}$ as $\mathcal{S}_\mathbb{E}$, and the set of all solver classes $\mathcal{S}_\mathbb{E}$ as $\mathcal{S}_\mathcal{E}$. The set $\mathcal{S}_\mathcal{E}$ is a graded poset as well, where $\mathcal{S}_\mathbb{E} \succ \mathcal{S}_{\mathbb{E}'} \iff \mathbb{E} \succ \mathbb{E}'$. We say that the solver class $\mathcal{S}_\mathbb{E}$ is more *powerful* than the solver class $\mathcal{S}_{\mathbb{E}'}$ if and only if $\mathcal{S}_\mathbb{E} \succ \mathcal{S}_{\mathbb{E}'}$. Note that by the definition of an environment class, $\mathcal{S}_\mathbb{E} \succ \mathcal{S}_{\mathbb{E}'} \iff \mathcal{S}_\mathbb{E} \supset \mathcal{S}_{\mathbb{E}'}$.

We now describe current literature in the context of this framework. The set of *model-based* reinforcement learning algorithms belong to the solver class $\mathcal{S}_{(S,A,P_a,R_a)}$, as the complete MDP must be accessible to the agent. These include early dynamic-programming methods such as value iteration (Bellman 1957) and policy iteration (Howard 1960). More recently, more-powerful solvers known as *model-free* methods belonging to the solver class $\mathcal{S}_{(S,A,P'_a,R'_a)}$ were developed. Notable examples of these methods include SARSA (Rummery and Niranjan 1994), Q-Learning

(Watkins 1989), Deep Q-Learning (Mnih et al. 2013; Mnih et al. 2015), and Policy Gradient methods (Sutton et al. 2000). Deep connections exist between model-free and model-based methods to solve this class of problem (Boyan 1999; Parr et al. 2008), involving space complexity, computational complexity, and sample complexity (Strehl et al. 2006). Recent methods even connect both approaches (Gu et al. 2016; Racanière et al. 2017). Yet, while recent advances in RL have made computationally tractable problems with massive (and even continuous) state and action spaces, these problems still belong to the same solver class.

In this paper, we present what we believe to be the first general solver more powerful than those of the class $\mathcal{S}_{(S,A,P'_a,R'_a)}$ for discrete (and finite) state-space problems. We refer to the set of environments $(S, A, \emptyset, *)$ as *on-demand environments* as the agent does not have the ability to plan in advance due to lack of knowledge of the state transition function. Our solver belongs to the highly constrained class $\mathcal{S}_{(S,A,\emptyset,\emptyset)}$. The only problems that it is unable to solve are those in which the agent knows only what actions it can take. We next give an overview of its functioning.

3 Simulated Problems, Subsolvers, & Subpolicies

We are given a problem $p \in \mathbb{P}_{(S,A,\emptyset,\emptyset)}$. We first define an *observation*. Given state and action spaces S and A respectively, an observation $o \in O(S, A)$ is a tuple (s, a, r, s') that denotes a state transition. The initial state is given by $s \in S$, the action taken is given by $a \in A$, the reward received is given by $r \in \mathbb{R}$, and the next state is given by $s' \in S$. Thus, the set of all observations $O(S, A)$ is $S \times A \times \mathbb{R} \times S$.

We denote the set of state-transition functions $P_a(\cdot, \cdot)$ for a given a state space S and action space A as $\mathcal{T}(S, A)$. Likewise we denote the set of reward functions $R_a(\cdot, \cdot)$ for a given a state space S and action space A as $\mathcal{R}(S, A)$.

We introduce more general notation. We let $\Sigma(X, Y)$ denote the set of all mappings $M : X \mapsto P$ such that P is the set of probability measures on the Borel algebra[1] on Y. The set $\Sigma(X, Y)$ can be thought of as the set of stochastic functions that map X to Y. As shorthand, we represent maps $M \in \Sigma(X, Y) \times \Sigma(X, Z)$ as maps $M \in \Sigma(X, Y \times Z)$ such that $M(x) = M_Y(x) * M_Z(x)$ for $M = (M_Y, M_Z)$ and $x \in X$. We note that $\Sigma(X, Y \times Z) \neq \Sigma(X, Y) \times \Sigma(X, Z)$ – in fact, the latter is only a subset of the former. This is true for the same reason as why one cannot define a joint distribution from only a set of marginal distributions. We can formally describe the set $\mathcal{R}(S, A) = \Sigma(S \times A, \mathbb{R})$ and the set $\mathcal{T}(S, A) = \Sigma(S \times A, S)$.

Our solver makes use of a function approximator $\aleph : O(S, A)^{t-1} \mapsto \mathcal{T}(S, A) \times \mathcal{R}(S, A)$ which takes as input the observations the agent has seen after t iterations and returns an approximated state transition function and reward function.

On time iteration t, we compute using the function approximator $\aleph(o) = (P_a(\cdot, \cdot)|_t, R_a(\cdot, \cdot)|_t)$ where $o \in O(S, A)^{t-1}$ is our running list of observations. We then construct a *simulated problem* $p|_t$ such that $M(p|_t) = (S, A, P_a(\cdot, \cdot)|_t, R_a(\cdot, \cdot)|_t)$ and $E(p|_t) = (S, A, P_a, R_a)$. We set the initial state s^t_1 of the simulated problem $p|_t$ to be s_t, the current state.

We then use a *subsolver* $\mathbb{S} \in \mathcal{S}_{(S,A,P_a,R_a)}$ to generate a *subpolicy* $\pi|_t$. Note that $\mathbb{S}$ can be *any* solver as (S, A, P_a, R_a) is the least hard environment. We let $\pi(s_t) = \pi|_t(s^t_1)$. Upon executing $\pi(s_t)$, we gain a new observation. We repeat this process to generate π. Note that in our solver, π is defined only at states that the agent visits.

3.1 Exploration Versus Exploitation

When we have only few observations, $\aleph$ will return poor approximations of $P_a(\cdot, \cdot)$ and $R_a(\cdot, \cdot)$. Likewise, when we have many observations, for the sake of computational tractability, we would like to execute multiple actions from $\pi|_t$ without re-running the solver $\mathbb{S}$.

We resolve our first conundrum with an ϵ-greedy approach. We define an *exploration constant* $\epsilon \in (0, 1)$ and an *exploration distribution* $e : A \mapsto \mathbb{R}$. On time iteration t, with probability ϵ^t, we draw $\pi(s_t)$ from e. A good choice of e allows the agent to explore the action space A while preventing the agent from taking clearly non-optimal actions.

We resolve the second issue with the use of a *trust factor* $\alpha \in [1, \infty)$. On time iteration t, we let $\pi(s_t), \ldots, \pi(s_{t+\lfloor \alpha^t \rfloor}) = \pi|_t(s^t_1), \ldots, \pi|_t(s^t_{\lfloor \alpha^t \rfloor})$. A good choice of α has $\lfloor \alpha^t \rfloor$ grow with the prediction accuracy of $\aleph$.

The hyperparameters ϵ, e, and α cannot be directly optimized for – they must be set based on problem-specific knowledge or intuition as there is no way to "test" the effect of changing them since we do not have the ability to sample from $P_a(\cdot, \cdot)$. We direct the reader to Figure 1 in the Appendix for a pictorial representation of our solver.

4 Defining the Function Approximator

We now describe $\aleph$, the backbone of our solver that predicts the state transition and reward functions given some observations. We let $\aleph(o) = (\aleph_P(o), \aleph_R(o))$ where $\aleph_P : O(S, A)^t \mapsto \mathcal{T}(S, A)$ returns $P_a(\cdot, \cdot)|_{t+1}$ and $\aleph_R : O(S, A)^t \mapsto \mathcal{R}(S, A)$ returns $R_a(\cdot, \cdot)|_{t+1}$.

We begin with a high-level overview of how $\aleph_P$ functions. We will define a bijective state-decomposition function which will transform states into a set of simplices, each paired with a natural number. We will then use a Gaussian-Process-based scheme to learn the distribution over the decomposed state given an action and our current state. We use the predicted distribution along with the inverse of our decomposition map to reconstruct $P_a(\cdot, \cdot)|_{t+1}$.

The approximator $\aleph_R$ functions similarly. Again, we use a Gaussian Process scheme. However, in this case, the Gaussian Process directly learns the distribution over the reward given an action and our current state to predict $R_a(\cdot, \cdot)|_{t+1}$.

4.1 Decomposing States

We now describe the bijective *state decomposition* $\mathcal{D} : S \mapsto \mathfrak{S}$. Any given state $s \in S$ can be described as a set of vectors – in practice, this is how states are stored. We can therefore represent the state space S as the Cartesian product of a series

[1]A Borel algebra on a topological space Y is a σ-algebra generated by the open sets of Y.

of vector spaces. As our state space is discrete, each of these vector spaces is isomorphic to $\mathbb{N}^k$ for some $k \in \mathbb{N}$. However, as we enforce that our state space is finite, we instead write S as a product of finite subsets (not subspaces) $V^k \subset \mathbb{N}^k$. Thus, for some sequence of natural numbers $\{m_k\}_{k=1}^n$, we can write

$$S = \bigcup_{x=0}^{\kappa} (V^{m_1})^x \times \bigcup_{x=0}^{\kappa} (V^{m_2})^x \times \cdots \times \bigcup_{x=0}^{\kappa} (V^{m_n})^x$$

where $\kappa \in \mathbb{N}$ is an arbitrarily large constant. We let S_k denote the kth vector space in the product, V^{m_k}. We define a sequence of vectors $\{\mathbf{l}_k\}_{k=1}^n$ such that each vector $\mathbf{l}_k \in \mathbb{N}^{m_k}$. The ith component of vectors in S_k can take on l_k^i distinct values, where l_k^i is the ith component of $\mathbf{l}_k$. It follows that

$$|S_k| = \prod_{i=1}^{m_k} l_k^i$$

We partition the state s into blocks $[s]_1, \dots, [s]_n$ such that some vector $\mathbf{v} \in s$ belongs to a block $[s]_k$ if and only if $\mathbf{v} \in S_k$. We now describe a bijective *block decomposition* $D : (S_k)^r \mapsto \Delta^{|S_k|-1} \times \mathbb{N}$ where

$$\Delta^c \equiv \left\{ \mathbf{x} \in \mathbb{R}^{c+1} \mid \sum_{i=1}^{c+1} x_i = 1, x_i > 0 \right\}$$

denotes the *c-simplex*, and $r, k \in \mathbb{N}$ are arbitrary constants. The map D decomposes arbitrary blocks $[s]_k$ into a $(|S_k|-1)$-simplex $\mathbf{X}_k$ that we call the *joint probability mass tensor* and a natural number c_k that we call the *size parameter*. The dimensions of $\mathbf{X}_k$ are given by $\mathbf{l}_k$. Just as we indexed values of $\mathbf{l}_k$ with the notation l_k^i, we will index values of $\mathbf{X}_k$ with the notation $\mathbf{X}_k^{\mathbf{v}}$ where $\mathbf{v} \in \mathbb{N}^{m_k}$.

We describe D with an example. Suppose that our block

$$[s]_k = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\}$$

consists of dimension 2, Boolean-valued vectors. In this case,

$$\mathbf{l}_k = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

and $|S_k| = 4$, and $D : (S_k)^4 \mapsto \Delta^3 \times \mathbb{N}$. We define a random vector $\mathbf{V} \in [s]_k$ which is distributed uniformly among the elements of $[s]_k$. For all $\mathbf{v} \in S_k$ we let $\mathbf{X}_k^{\mathbf{v}} = \Pr[\mathbf{v} = \mathbf{V}]$. However, with just $\mathbf{X}_k$ alone, we cannot reconstruct $[s]_k$. We let the size parameter $c_k = |[s]_k|$. One can reconstruct $[s]_k$ using both $\mathbf{X}_k$ and c_k as the number of instances of some vector $\mathbf{v} \in [s]_k$ is precisely equal to $c_k \mathbf{X}_k^{\mathbf{v}}$. In our example

$$[s]_k = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} \mapsto \left(\begin{bmatrix} .25 & 0 \\ .5 & .25 \end{bmatrix}, 4 \right)$$

We now formally define

$$\mathcal{D}(s) \equiv (D([s]_1), D([s]_2), D([s]_3), \dots, D([s]_n))$$

It follows that $\mathfrak{S} = \prod_{k=1}^n (\Delta^{|S_k|-1} \times \mathbb{N})$. We abuse notation and allow $\mathcal{D}$ to act on observations as well, transforming states from S to $\mathfrak{S}$. More formally, we let $\mathcal{D} : O(S, A)^t \mapsto O(\mathfrak{S}, A)^t$.

4.2 Predicting State Transitions via Observations

We now describe how $\aleph_P$ functions. We work under the assumption that the probability distribution over the quantity $(D([s_{t+1}]_k) - D([s_t]_k))$ conditioned on the action take $a \in A$ remains approximately invariant over time for all $k \in \{1 \dots, n\}$, $s_t \in S$. We later show that our method can be easily extended to the case where the probability distribution over $D([s_{t+1}]_k)$ conditioned on $s_t \in S$ and the action take $a \in A$ remains approximately invariant over time for all $k \in \{1 \dots, n\}$. Although these two cases may not hold for the state space S, our solver will still function so long as we can construct a *time invariant map* (see Appendix B for definition). Time invariant maps can also make large problems more computationally tractable.

We start by characterizing the set of possible $(D([s_{t+1}]_k) - D([s_t]_k))$. Recall that the block decomposition $D : (S_k)^r \mapsto \Delta^{|S_k|-1} \times \mathbb{N}$. We let

$$\hexagon^c \equiv \{\mathbf{u} - \mathbf{v} \mid \mathbf{u}, \mathbf{v} \in \Delta^c\}$$

(we choose the symbol $\hexagon$ as this is the shape of space in $\mathbb{R}^3$). It follows that

$$\hexagon^{|S_k|-1} \times \mathbb{Z} = \{D([s_{t+1}]_k) - D([s_t]_k)\}$$

where $s_t, s_{t+1} \in S$. Our approximator $\aleph_P$ will function by first mapping our observation $o' = \mathcal{D}(o)$. We now define a map

$$\delta : S^t \times (\Delta^c \times \mathbb{N})^{2t} \times A^t \mapsto (\hexagon^c \times \mathbb{Z} \times A \times I_P)^t$$

where $I_P \subseteq \{V^{m_k}\}_{k=1}^n$ is the set of *state-dependent transition factors*. Formally, I_P is the subset of $\{V^{m_k}\}_{k=1}^n$ of lowest cardinality such that it can be guaranteed that the state transition function $P_a(\cdot, \cdot) \in \Sigma(I_P \times A, S)$. Intuitively, I_P is the smallest subset of state elements that affect the distribution over future states.

We now describe the map δ. From a high level, δ will, for a specific block k, subtract the successive decompositions of this block, and pair the difference with the action taken on that time iteration as well as the state dependent transition factors from that time iteration.

The input to δ comes from the mapped observation o': it is given by the tuple

$$[(s_\ell, D([s_\ell]_k), D([s_{\ell+1}]_k), a_\ell)]_{\ell=1}^t$$

where $k \in \{1, \dots, n\}$. The output of δ given the above is given by

$$\left[(D([s_\ell]_k) - D([s_{\ell+1}]_k), a_\ell, i_P^\ell)\right]_{\ell=1}^t$$

where $i_P^\ell \in I_P$ is a subset of $\{[s_\ell]_k\}_{k=1}^n$. We now present an alternative map δ for the case in which the probability distribution over $D([s_{t+1}]_k)$ conditioned on $s_t \in S$ and the action take $a \in A$ remains approximately invariant over time. With only this change, the remainder of our method functions the same for both cases. We do not change the domain and codomain of δ. However, we change the output given the input above to be

$$\left[(D([s_\ell]_k) - \mathbf{v}), a_\ell, i_P^\ell)\right]_{\ell=1}^t$$

where $\mathbf{v} \in \Delta^{|S_k|-1}$ is an arbitrary element of the $(|S_k|-1)$-simplex.

We now again abuse notation and allow δ to act on observations. Formally, we let $\delta : O(\mathfrak{S}, A)^t \mapsto \mathbb{X}$ where

$$\mathbb{X} \equiv \prod_{k=1}^{n} \left(\hexagon^{|S_k|-1} \times \mathbb{Z} \times A \times I_P \right)^t \tag{2}$$

We let the kth term in the product of equation (2) be $\mathbb{X}_k$. The set $\mathbb{X}_k$ refers to that in $\mathbb{X}$ that was mapped from our observations of the kth block $[s]_k$. We let the domain of δ be $O(\mathfrak{S}, A)$ instead of $O(\mathfrak{S}, A) \times O(S, A)$ even though original states $s_1, \ldots, s_t$ are required because the original states can be reconstructed from the decomposed observations using $\mathcal{D}^{-1}$.

For each $\mathbb{X}_k$, we now define an *approximator map* $F_k : \mathbb{X}_k \mapsto \Sigma(I_P \times A, \hexagon^{|S_k|-1}) \times \Sigma(I_P \times A, \mathbb{Z})$ using a Gaussian Process-based approach. In doing so, we define a map

$$F : \mathbb{X} \mapsto \prod_{k=1}^{n} \Sigma(I_P \times A, \hexagon^{|S_k|-1}) \times \Sigma(I_P \times A, \mathbb{Z})$$

We note that in the above, we are using the shorthand notation defined in §3. We let

$$F_k^{\hexagon} : \left(\hexagon^{|S_k|-1} \times A \times I_P \right)^t \mapsto \Sigma(I_P \times A, \hexagon^{|S_k|-1})$$

and

$$F_k^{\mathbb{Z}} : (\mathbb{Z} \times A \times I_P)^t \mapsto \Sigma(I_P \times A, \mathbb{Z})$$

We consequently let $F_k \equiv (F_k^{\hexagon}, F_k^{\mathbb{Z}})$.

4.3 Gaussian Process-based Approximator Maps

We use *Gaussian Processes* (GP) to define $F_k^{\hexagon}$ and $F_k^{\mathbb{Z}}$. We start by defining $F_k^{\mathbb{Z}}$, the more straightforward of the two. We first define the GP.

A GP is a map $\mathcal{GP} : (\mathbb{R}^c \times \mathbb{R})^k \mapsto \Sigma(\mathbb{R}^c, \mathbb{R})$ that aims to approximate a stochastic function $f : \mathbb{R}^c \mapsto \mathbb{R}$ where $c, k \in \mathbb{N}$ are arbitrary constants. The constant k denotes the number of data points $(\mathbf{x}, y)$ we have, where $\mathbf{x} \in \mathbb{R}^c$ and $y \in \mathbb{R}$.

More specifically, the map $\mathcal{GP}$ places a joint normal prior over the values of f

$$f \sim \mathcal{N}\left(\boldsymbol{\mu}(f), K(\cdot, \cdot)\right)$$

where the function $K : \mathbb{R}^c \times \mathbb{R}^c \mapsto \mathbb{R}$ is a *kernel function* that defines the covariance matrix of the joint normal distribution. Given sample points $(\mathbf{x}, y)$ for $\mathbf{x} \in \mathbb{R}^c$ and $y \in \mathbb{R}$, the map $\mathcal{GP}$ updates the prior using Bayes' law.

We use $\mathcal{GP}$ to describe the approximator map $F_k^{\mathbb{Z}}$. We first note that $I_P \cong \mathbb{N}^p \subset \mathbb{R}^p$ where $p \in \mathbb{N}$ is an arbitrary constant. As the action space A can be continuous, we note that $A \cong \mathbb{R}^q$ where $q \in \mathbb{N}$ is an arbitrary constant. It follows that the set $I_P \times A$ can be mapped to $\mathbb{R}^{p+q}$. Thus, we can let

$$F_k^{\mathbb{Z}} : (\mathbb{R}^{p+q} \times \mathbb{Z})^t \mapsto \Sigma(\mathbb{R}^{p+q}, \mathbb{Z})$$

We now define

$$F_k^{\mathbb{Z}}\left([(\mathbf{v}_i, z_i)]_{i=1}^t \right) \equiv \left\langle \mathcal{GP}\left([(\mathbf{v}_i, z_i)]_{i=1}^t \right) \right\rangle$$

where the notation $\langle \cdot \rangle : \Sigma(X, \mathbb{R}) \mapsto \Sigma(X, \mathbb{Z})$ denotes a rounding operation on the GP.

We now describe the map $F_k^{\hexagon}$. Using the map from $I_P \times A$ to $\mathbb{R}^{p+q}$ as described above, we similarly let

$$F_k^{\hexagon} : \left(\mathbb{R}^{p+q} \times \hexagon^{|S_k|-1} \right)^t \mapsto \Sigma\left(\mathbb{R}^{p+q}, \hexagon^{|S_k|-1} \right)$$

We abuse notation and expand the domain of the GP to allow approximations of vector-valued stochastic functions, letting

$$\mathcal{GP} : (\mathbb{R}^c \times \mathbb{R}^d)^k \mapsto \prod_{i=1}^{d} \Sigma(\mathbb{R}^c, \mathbb{R})$$

where

$$\mathcal{GP}\left([(\mathbf{x}_i, \mathbf{y}_i)]_{i=1}^k \right) \equiv \left[\mathcal{GP}\left(\left[(\mathbf{x}_i, y_i^j) \right]_{i=1}^k \right) \right]_{j=1}^d$$

for $\{\mathbf{x}_i\}_{i=1}^k \subset \mathbb{R}^c$ and $\{\mathbf{y}_i\}_{i=1}^k \subset \mathbb{R}^d$, adopting the vector index notation defined in §4.1.

Just as we did with $F_k^{\mathbb{Z}}$, we would like to describe $F_k^{\hexagon}$ directly with the $\mathcal{GP}$ transformation. However, we cannot do this as the codomain of $F_k^{\hexagon}$ is $\Sigma(\mathbb{R}^{p+q} \times \hexagon^{|S_k|-1}, \hexagon^{|S_k|-1})$ while the $\mathcal{GP}$ transformation will return an element of $\Sigma(\mathbb{R}^{p+q} \times \mathbb{R}^d, \mathbb{R}^d)$ where $d \in \mathbb{N}$ is an arbitrary constant. Thus, we require a bijective map

$$T : \hexagon^{|S_k|-1} \mapsto \mathbb{B}^d$$

where the set $\mathbb{B} \subset \mathbb{R}$, in order to use the $\mathcal{GP}$ transformation. We specifically let

$$\mathbb{B}^d = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \max(|x_i|) \le 1 \right\}$$

We describe such a map in §5, and generalize our approach to allow GPs to approximate functions in any constrained space such that certain properties of the space hold.

5 Predictions in Constrained Spaces with Warped Gaussian Processes

We begin by defining the *Warped Gaussian Process* (WGP). The WGP was first defined in (Snelson, Ghahramani, and Rasmussen 2004) as a means of generalizing the GP to approximate stochastic functions that are not modeled well by standard GPs. As it is defined by Snelson et al., a WGP performs the same task as a standard GP. Formally, they define $\mathcal{WGP} : (\mathbb{R}^c \times \mathbb{R})^k \mapsto \Sigma(\mathbb{R}^c, \mathbb{R})$. However, in a WGP, we *warp* the observation space by mapping it to a latent space. We then assume that a standard GP can appropriately approximate the stochastic function mapping the input space to the latent space. Finally, we transform the output of the GP back into the original observation space.

We use a similar approach to generalize the GP framework to approximate stochastic functions with certain constrained codomains. We formally define

$$\mathcal{WGP} : (\mathbb{R}^c \times Y)^k \mapsto \Sigma(\mathbb{R}^c, Y)$$

where the set Y is restricted to be homeomorphic to $\mathbb{B}^d$ for some $d \in \mathbb{N}$. We let

$$\mathcal{WGP}\left([(\mathbf{x}_i, \mathbf{y}_i)]_{i=1}^k \right) \equiv U\left(\mathcal{GP}\left([(\mathbf{x}_i, T(\mathbf{y}_i))]_{i=1}^k \right) \right)$$

where $\{\mathbf{y}_i\}_{i=1}^k \subset Y$, the map $T : Y \mapsto \mathbb{B}^d$ denotes the homeomorphism, and $U : \Sigma(\mathbb{R}^c, \mathbb{B}^d) \mapsto \Sigma(\mathbb{R}^c, Y)$. We now define the map U as

$$[U(M_{\mathbb{B}}(\mathbf{u}))]\,(\mathbf{y}) = [M_{\mathbb{R}}(\mathbf{u})]\,(T(\mathbf{y}))$$

where $M_{\mathbb{B}} \in \Sigma(\mathbb{R}^c, \mathbb{B}^d)$, $\mathbf{y} \in Y$, an $\mathbf{u} \in \mathbb{R}^c$. As $M_{\mathbb{B}}(\mathbf{u})$ and $U(M_{\mathbb{B}}(\mathbf{u}))$, probability measures on Borel algebras, are maps themselves, we use the bracket notation to denote that the returned probability measure is acting on the following parameter.

We enforce the homeomorphism condition in order to ensure that elements of the range of U maintain properties of their preimages. If T was not continuous, elements in the range of U may not map to continuous probability measures even if their preimages did. Likewise, if T^{-1} was not continuous, elements in the range of U may not map to probability measures with connected typical sets even if their preimages did.

Theorem 1. *Suppose we are given two closed sets $X, Y \subset V$, and a vector space $\mathbb{V} = (V, \mathbb{R}, +, \cdot)$ defined over the field of real numbers. Furthermore, suppose that $\mathbf{0} \in X \cap Y$. If there exist continuous functions $\mathrm{M}_X, \mathrm{M}_Y : V \mapsto \mathbb{R}$ such that for all limit points $\mathbf{x} \in X$ and $\mathbf{y} \in Y$, $\mathrm{M}_X(\mathbf{x}) = \mathrm{M}_Y(\mathbf{y}) = 1$, and for all $c \in \mathbb{R}, \mathbf{v} \in V$, $\mathrm{M}_X(c\mathbf{v}) = c\mathrm{M}_X(\mathbf{v})$ and $\mathrm{M}_Y(c\mathbf{v}) = c\mathrm{M}_Y(\mathbf{v})$, then X and Y are homeomorphic.*

We direct the reader to Appendix C for the proof. We show that this result can be a powerful tool for modeling variables in constrained spaces by using it to show a homeomorphism between the sets $\hexagon^d$ and $\mathbb{B}^d$.

Recall that $\hexagon^d \subset \mathbb{R}^{d+1}$. Vectors $\mathbf{v} \in \hexagon^d$ must satisfy the conditions $\sum_{i=1}^{d+1} v_i = 0$ and $\sum_{i=1}^{d+1} |v_i| \leq 2$. We first show a homeomorphism between $\hexagon^d$ and $\mathbb{W}^d \subset \mathbb{R}^d$ which we call the *weight space*. We define the map $W : \hexagon^d \mapsto \mathbb{W}^d$ such that for some $\mathbf{v} \in \hexagon^d$,

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \mapsto \begin{bmatrix} -v_2 \\ \vdots \\ -v_n \end{bmatrix} = \mathbf{w} \in \mathbb{W}^d$$

The name "weight space" comes from the fact that $\mathbf{w}$ corresponds to the linear weights of the basis for the hyperplane given by the linear constraint $\sum v_i = 0$. We note that the map W is a homeomorphism as it is continuous and bijective, and its inverse is also continuous.

By applying the map W, we have removed the constraint $\sum v_i = 0$. It follows that in weight space, only one constraint holds: the mapped version of the constraint $\sum |v_i| \leq 2$. For some $\mathbf{v} \in \hexagon^d$, $v_1 = \sum w_i$ where $\mathbf{w} = W(\mathbf{v})$. With this, we get

$$\mathbb{W}^d = \left\{ \mathbf{w} \in \mathbb{R}^d \mid \sum |w_i| + \left| \sum w_i \right| \leq 2 \right\}$$

We now apply Theorem 1 to show the homeomorphism between $\mathbb{W}^d$ and $\mathbb{B}^d$ by defining the functions

$$\mathrm{M}_{\mathbb{W}}(\mathbf{v}) = \frac{\sum_{i=1}^n |v_i| + |\sum_{i=1}^n v_i|}{2}$$

$$\mathrm{M}_{\mathbb{B}}(\mathbf{v}) = \max(|v_i|)$$

These functions satisfy the conditions of M_X and M_Y as defined in the statement of Theorem 1 (the homeomorphism is shown explicitly in Appendix C). Letting T be the homeomorphism between $\hexagon^{|S_k|-1}$ and $\mathbb{B}^{|S_k|-1}$, we let

$$F_k^{\hexagon}\left([(\mathbf{x}_i, \mathbf{v}_i)]_{i=1}^t\right) \equiv \mathcal{WGP}\left([(\mathbf{x}_i, \mathbf{v}_i)]_{i=1}^t\right)$$

for $\{\mathbf{x}_i\}_{i=1}^t \subset \mathbb{R}^{p+q}$ and $\{\mathbf{v}_i\}_{i=1}^t \subset \hexagon^{|S_k|-1}$. In doing so, we have now defined the approximator map F_k, and thus the full map $F \equiv [F_k]_{k=1}^n$. At this point, we have constructed a map

$$O(S, A)^t \mapsto \Sigma\left(S \times A, \prod_{i=1}^n \left(\hexagon^{|S_k|-1} \times \mathbb{Z}\right)\right)$$

To complete $\aleph_P$, we construct a map Υ to $\Sigma(S \times A, S) = \mathcal{T}(S, A)$ that functions exactly in the same way as the map U, except for the requisite map T (which in this case we denote as Z) is given by either adding $D([s_{t+1}]_k)$ or the arbitrary element $\mathbf{v} \in \Delta^{|S_k|-1}$ back to the outputs of each of the F_k, and using the map $\mathcal{D}^{-1}$.

To define $\aleph_R$, we define the *state-dependent reward factors* $I_R \subset \{V^{m_k}\}_{k=1}^n$ such that $R_a(\cdot, \cdot) \in \Sigma(I_R \times A, \mathbb{R})$. We let $I_R \cong \mathbb{N}^r \subset \mathbb{R}^r$. We again map $I_R \times A \mapsto \mathbb{R}^{q+r}$, letting $\aleph_R : (\mathbb{R}^{q+r} \times \mathbb{R})^t \mapsto \Sigma(\mathbb{R}^{q+r}, \mathbb{R})$. We directly use a GP

$$\aleph_R\left([(\mathbf{x}_i, y_i)]_{i=1}^t\right) \equiv \mathcal{GP}\left([(\mathbf{x}_i, y_i)]_{i=1}^t\right)$$

for $\{\mathbf{x}_i\}_{i=1}^t \subset \mathbb{R}^{r+q}$, and $\{y_i\}_{i=1}^t \subset \mathbb{R}$. With this, we have fully defined the map $\aleph$. We direct the reader to Figure 2 in the Appendix for a commutative diagram that describes the maps that define $\aleph$.

6 Experiments

We experimentally validate our solver on a toy problem (described in detail in Appendix D.1). In our problem, a fishing person P catches fish from a lake on a daily basis and sells the entirety of his/her catch from the previous day. The number of fish that P catches each day is given by a positive-truncated normal distribution that is then discretized with a rounding operation (we justify this choice in Appendix D.1). The fish that P catches belong to one of T *types*. Each type of fish tends to reside at a different depth of the lake. More specifically, we uniformly split the lake into depth strata, and let the typical fish of type k reside in the kth stratum. Each day, P chooses which depth to fish at. The distribution over fish types in P's catch is thus a function of the depth that P chooses to fish at. Furthermore, the prices of each of the T types of fish vary periodically over time. We aim to find the depth that P should fish at each time iteration in order to maximize his/her total profit after D days.

We also consider this problem under the following variants:

1. We let the center of each stratum be uniformly distributed over all depths—we call this the "random means" variant
2. We let the number of fish that P catches be a well-behaved (sinusoidal) function of the depth—we call this the "size by depth" variant.

We note that these two variants are not mutually exclusive—that is, we also test our model on the combination of these two variants.

6.1 Experimental Setup

We use GPy (GPy 2014) to implement our solver. In our setup, we let $T = 5$. We also test our solver with $D = 75$ and $D = 365$. We test this problem as an element of the environment class $\mathbb{P}_{(S,A,\emptyset,R'_a)}$[2]. We use Deep Q-Learning (Mnih et al. 2013; Mnih et al. 2015) as our subsolver (recall that this solver belongs to the solver class $\mathcal{S}_{(S,A,P'_a,R'_a)}$). We run our solver on this problem 50 times per variant combination. Other hyperparameters to the problem and additional experimental details are given in Appendix D.1.

We also consider this problem within the environment class $\mathbb{P}_{(S,A,P'_a,R'_a)}$. For each variant combination, we also test the Deep Q-Learning solver 50 times. We optimize the learning rate of that solver and let both our subsolver and the Deep Q-Learning solver share the same set of hyperparameters (described in detail in Appendix D.1). For all experiments, we let the exploration constant $\epsilon = 0.95$, the trust factor $\alpha = 1.02$, and the exploration distribution $e = \text{unif}[0, 1]$. We note that ϵ and α are not optimized. However, in a real-world scenario, it is impossible to directly optimize for these constants as no information about $P_a(\cdot, \cdot)$ or $R_a(\cdot, \cdot)$ is known.

6.2 Results

We now compare the performance of our solver in the on-demand environment with that of the Deep Q-Learning solver in the less constrained environment (S, A, P'_a, R'_a).

D	RM	SBD	$\mathbb{S}_{(S,A,\emptyset,R'_a)}$	$\mathbb{S}_{(S,A,P'_a,R'_a)}$	%
75	0	0	[3908, 3991]	[4162, 4227]	94.2
75	0	1	[4493, 4592]	[4598, 4687]	97.8
75	1	0	[4239, 4710]	[5158, 5525]	83.7
75	1	1	[4572, 5128]	[5192, 5748]	88.6
365	0	0	[21278, 21471]	[22008, 22079]	96.7
365	0	1	[21449, 21623]	[22222, 22273]	96.8
365	1	0	[22506, 23224]	[23441, 24220]	95.9
365	1	1	[22309, 23199]	[22804, 23581]	98.1

Table 1: Comparative results on the fishing problem between our solver and the Deep Q-Learning solver

In Table 1, we describe the comparative performance between our solver (denoted $\mathbb{S}_{(S,A,\emptyset,R'_a)}$) and the Deep Q-Learning solver (denoted $\mathbb{S}_{(S,A,P'_a,R'_a)}$). The column D denotes the time horizon of the MDP. The columns "RM" and "SBD" denote whether or not we used the Random Means and Size by Depth variants respectively. The intervals below each solver denote the rounded 95% confidence interval (assuming the Student's t-distribution) for the mean cumulative reward attained by the solver. The column "%" denotes the mean ratio of our solver's cumulative reward to that of the Deep Q-Learning solver.

[2]We are in the process of compiling results for the $\mathbb{P}_{(S,A,\emptyset,\emptyset)}$ case, but believe that our current results are sufficient as a proof of concept for the more complex map $\aleph_P$.

6.3 Discussion

We begin by noting that in this relatively simple problem and its variants, our solver, in a single episode, is consistently capable of performing at a comparable level to a state-of-the-art RL method that samples from $P_a(\cdot, \cdot)$ multiple times—even when the time horizon is relatively small. In other words, the map $\aleph_P$ is capable of accurately approximating $P_a(\cdot, \cdot)$ even with relatively few observations. At a more general level, this experimentally validates the claim that at least well-behaved stochastic functions can be accurately modeled in a latent space homeomorphic to the original. This motivates the idea that Theorem 1 can be a powerful tool in the general field of stochastic modeling.

We further note that when the time horizon D increases, our solver's performance relative to that of the Deep Q-Learning solver tends to increase. This behavior occurs because, at some point, the approximation of $P_a(\cdot, \cdot)$ given by $\aleph_P$ becomes very accurate. Thus, at this point, our agent begins to act optimally. As the time horizon increases, our agent's initial errors become insignificant with respect to the cumulative reward.

7 Conclusions & Future Research

In this paper, we presented a principled method for approximating the optimal policy of discrete state space MDPs in the highly constrained—yet often realistic—environment in which the agent has no knowledge of the state transition or reward functions. Our method is computationally tractable—capable of running on commodity hardware—and has been shown experimentally to demonstrate comparable performance to state-of-the-art methods that are capable of sampling from the state transition function.

We believe that the natural and highly impactful application of this work is in the field of automated mechanism design (Conitzer and Sandholm 2002; Conitzer and Sandholm 2004; Sandholm and Likhodedov 2015; Albert, Conitzer, and Stone 2017). A mechanism design problem can be written as a MDP where the reward $R_a(\cdot, \cdot)$ is given by the central clearinghouse's (i.e., mechanism designer's) utility. This is nonstandard, as current solvers are incapable of solving these problems—the stochastic behavior of agents is often unknown or difficult to simulate. However, we believe that in most scenarios, a mechanism designer with enough intuition can define a time invariant map over the problem's state space. Thus, our method could potentially be used to solve for the mechanism designer's actions. The power of our solver in this context comes from the lack of assumptions that it imposes—other than the time invariant map, we make no assumptions about the behavior of the agents, including assumptions regarding the necessity of individual rationality in the mechanism, or even the capability to act rationally. Thus, our solver is capable of optimizing over the potentially even highly stochastic behavior of irrational agents. We direct the reader to Appendix D.2 for an example of a problem of this type (where different countries strategically set tariffs on imports), and see research in this direction as promising.

References

[Albert, Conitzer, and Stone 2017] Albert, M.; Conitzer, V.; and Stone, P. 2017. Automated design of robust mechanisms. In *AAAI Conference on Artificial Intelligence (AAAI)*, 298–304.

[Bellman 1957] Bellman, R. 1957. A Markovian decision process. *Journal of Mathematics and Mechanics* 679–684.

[Boyan 1999] Boyan, J. A. 1999. Least-squares temporal difference learning. In *International Conference on Machine Learning (ICML)*, 49–56.

[Conitzer and Sandholm 2002] Conitzer, V., and Sandholm, T. 2002. Complexity of mechanism design. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 103–110.

[Conitzer and Sandholm 2004] Conitzer, V., and Sandholm, T. 2004. Self-interested automated mechanism design and implications for optimal combinatorial auctions. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, 132–141.

[GPy 2014] GPy. 2014. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`.

[Gu et al. 2016] Gu, S.; Lillicrap, T.; Sutskever, I.; and Levine, S. 2016. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning (ICML)*, 2829–2838.

[Howard 1960] Howard, R. 1960. Dynamic programming and Markov processes.

[Hu, Wellman, and others 1998] Hu, J.; Wellman, M. P.; et al. 1998. Multiagent reinforcement learning: theoretical framework and an algorithm. In *International Conference on Machine Learning (ICML)*, volume 98, 242–250. Citeseer.

[Huang et al. 2001] Huang, Q.; Yokoi, K.; Kajita, S.; Kaneko, K.; Arai, H.; Koyachi, N.; and Tanie, K. 2001. Planning walking patterns for a biped robot. *IEEE Transactions on Robotics and Automation* 17(3):280–289.

[Kaelbling, Littman, and Moore 1996] Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

[Mnih et al. 2013] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

[Parr et al. 2008] Parr, R.; Li, L.; Taylor, G.; Painter-Wakefield, C.; and Littman, M. L. 2008. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 752–759. ACM.

[Puterman 2014] Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[Racanière et al. 2017] Racanière, S.; Weber, T.; Reichert, D.; Buesing, L.; Guez, A.; Rezende, D. J.; Badia, A. P.; Vinyals, O.; Heess, N.; Li, Y.; et al. 2017. Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 5690–5701.

[Rummery and Niranjan 1994] Rummery, G. A., and Niranjan, M. 1994. On-line Q-learning using connectionist systems. Technical report, Cambridge University, Engineering Department.

[Sandholm and Likhodedov 2015] Sandholm, T., and Likhodedov, A. 2015. Automated design of revenue-maximizing combinatorial auctions. *Operations Research* 63(5):1000–1025.

[Snelson, Ghahramani, and Rasmussen 2004] Snelson, E.; Ghahramani, Z.; and Rasmussen, C. E. 2004. Warped gaussian processes. In *Advances in neural information processing systems*, 337–344.

[Strehl et al. 2006] Strehl, A. L.; Li, L.; Wiewiora, E.; Langford, J.; and Littman, M. L. 2006. PAC model-free reinforcement learning. In *International Conference on Machine Learning (ICML)*, 881–888. ACM.

[Sutton and Barto 1998] Sutton, R. S., and Barto, A. G. 1998. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge.

[Sutton et al. 2000] Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 1057–1063.

[Urmson et al. 2008] Urmson, C.; Anhalt, J.; Bagnell, D.; Baker, C.; Bittner, R.; Clark, M.; Dolan, J.; Duggins, D.; Galatali, T.; Geyer, C.; et al. 2008. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics* 25(8):425–466.

[Watkins 1989] Watkins, C. J. C. H. 1989. *Learning from delayed rewards*. Ph.D. Dissertation, King's College, Cambridge.

Appendix for Paper ID: 7043

A Pictorial representation of an on-demand solver

Figure 1 gives a pictorial representation of the on-demand solver described in Section 3, along with an example of the exploration versus exploitation tradeoffs made based on total progress.

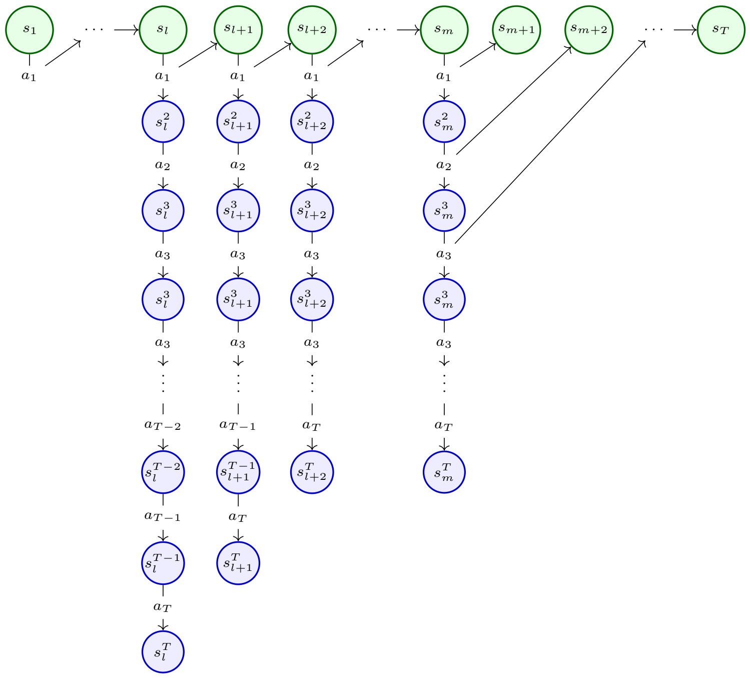

Figure 1: **Approximating the optimal policy in an on-demand environment.** The columns denote the subpolicies $\pi|_1, \pi|_2, \pi|_3, \ldots$ and the top row of actions denotes the returned values of π. When the current time t is "small", we choose to explore often, as ϵ^t is large. As we progress further, we stop exploring and instead choose to act by taking the first action $\pi|_t(s_1^t)$ from the optimal policy of the simulated problem $p|_t$. As our trust in $P_a(\cdot,\cdot)|_t$ (given by $\lfloor \alpha^t \rfloor$) increases, we take more actions from $\pi|_t$ before optimizing again.

B Time-invariant maps

We note that the domain of the function approximators $\aleph_P$ and $\aleph_R$ can be massive to the point in which it becomes computationally intractable to compute them for larger problems. Furthermore, the conditions stated in §4.2 can be restrictive. We introduce the notion of a *time-invariant map* as means of combating these issues.

A P-time invariant map $\nu_P : S \mapsto N_P$ is a map such that $\exists \{m_k\}_{k=1}^{n'}, \{\mathbf{l}_k\}_{k=1}^{n'}$ for $\kappa \in \mathbb{N}$ is an arbitrarily large constant where

$$N_P = \bigcup_{x=0}^{\kappa} (V^{m_1})^x \times \bigcup_{x=0}^{\kappa} (V^{m_2})^x \times \cdots \times \bigcup_{x=0}^{\kappa} (V^{m_{n'}})^x \tag{3}$$

for V^{m_k} is a finite subset of $\mathbb{N}^{m_k}$, the ith component of vectors in V^{m_k} can take on l_k^i distinct values, and one of the following two properties hold

- The probability distribution over $D([\nu_P(s_{t+1})]_k)$ conditioned on $\nu_P(s_t) \in N_P$ and the action take $a \in A$ remains approximately invariant over time for all $k \in \{1 \ldots, n'\}$
- The probability distribution over the quantity $(D([\nu_P(s_{t+1})]_k) - D([\nu_P(s_t)]_k))$ conditioned on the action take $a \in A$ remains approximately invariant over time for all $k \in \{1 \ldots, n'\}, s_t \in S$

In the above, $s_t \in S$ denotes the current state on time iteration t.

Likewise, a R-time invariant map $\nu_R : S \mapsto N_R$ is a map such that $\exists \{m_k\}_{k=1}^{n''}, \{\mathbf{l}_k\}_{k=1}^{n''}$ where

$$N_R = \bigcup_{x=0}^{\kappa} (V^{m_1})^x \times \bigcup_{x=0}^{\kappa} (V^{m_2})^x \times \cdots \times \bigcup_{x=0}^{\kappa} (V^{m_{n''}})^x \tag{4}$$

$$\begin{tikzcd}
\Sigma(S \times A, \mathbb{R}) & & & & & \Sigma(I_P \times A, \mathbb{Z}) \\
O(S,A)^t \arrow[u, mapsto, "\aleph_R"] \arrow[r, mapsto, "\mathcal{D}"] \arrow[dr, mapsto, "\aleph_P"'] & O(\mathfrak{S},A)^t \arrow[r, mapsto, "\delta"] & \mathbb{X} \arrow[d, mapsto, "F=[F_k]_{k=1}^n"] & & \mathbb{X}_k \arrow[ur, mapsto, "F_k^{\mathbb{Z}}"] \arrow[dr, mapsto, "F_k^{\hexagon}"'] & \\
& \Sigma(S \times A, S) & \Sigma\left(S \times A, \prod_{i=1}^n \left(\hexagon^{|S_k|-1} \times \mathbb{Z}\right)\right) \arrow[l, mapsto, "\Upsilon"] & & & \Sigma(I_P \times A, \hexagon^{|S_k|-1})
\end{tikzcd}$$

Figure 2: A full commutative diagram describing the function approximator $\aleph$.

for V^{m_k} is a finite subset of $\mathbb{N}^{m_k}$, the ith component of vectors in V^{m_k} can take on l_k^i distinct values, and the probability distribution over the reward on time t $r_t \in \mathbb{R}$ conditioned on $\nu_R(s_t) \in N_R$ and the action taken $a \in A$ remains approximately invariant over time for all $k \in \{1 \ldots, n''\}$.

We call N_P and N_R the *natural state spaces*. We impose constraints (3) and (4) in order to guarantee that the state decomposition function $\mathcal{D}$ can act on elements of the sets N_P and N_R. We abuse notation and also define $\nu_P : O(S,A)^t \mapsto O(N_P,A)^t$, $\nu_P^{-1} : O(N_P,A)^t \times O(S,A)^t \mapsto O(S,A)^t$, and $\nu_R : O(S,A)^t \mapsto O(N_R,A)^t$, allowing the time invariant maps to transform observations into the natural state spaces.

The maps ν_P and ν_R allow us to potentially reduce the state space and more tractably compute the maps $\aleph_P^N : O(N_P,A)^t \mapsto \mathcal{T}(N_P,A)$ and $\aleph_R^N : O(N_R,A)^t \mapsto \mathcal{R}(N_R,A)$. Alternatively, it may be possible for us to construct natural state spaces such that the conditions in §4.2 hold, even if they do not hold for the original state space. We note that the set $\mathcal{R}(N_R,A)$ can easily be mapped to the set $\mathcal{R}(S,A)$ under the transformation

$$R_a(s,s') \equiv R_a^N(\nu_R(s), \nu_R(s'))$$

where $s, s' \in S$. Thus, we let $\aleph_R^N : O(N_R,A)^t \mapsto \mathcal{R}(S,A)$. Likewise, we can map $\mathcal{T}(S,A)$ to $\mathcal{T}(N_P,A)$ with the transformation

$$P_a(s,s') \equiv P_a^N(\nu_P(s), \nu_P(s'))$$

and let $\aleph_P^N : O(N_P,A)^t \mapsto \mathcal{T}(S,A)$. We let $P_a^N(\cdot,\cdot)$ denote the element of $\mathcal{T}(N_P,A)$ that maps to the true state transition function $P_a(\cdot,\cdot) \in \mathcal{T}(S,A)$ under the above map. We similarly define $R_a^N(\cdot,\cdot)$ as that which maps to the reward function $R_a(\cdot,\cdot)$. It follows that

$$\aleph(o) = (\aleph_P(o), \aleph_R(o)) = (\aleph_P^N(\nu_P(o)), \aleph_R^N(\nu_R(o)))$$

for all observations $o \in O(S,A)^t$.

C Proof of Theorem 1

We restate the theorem below.

Theorem 1. *Suppose we are given two closed sets $X, Y \subset V$, and a vector space $\mathbb{V} = (V, \mathbb{R}, +, \cdot)$ defined over the field of real numbers. Furthermore, suppose that $\mathbf{0} \in X \cap Y$. If there exist continuous functions $\mathrm{M}_X, \mathrm{M}_Y : V \mapsto \mathbb{R}$ such that for all limit points $\mathbf{x} \in X$ and $\mathbf{y} \in Y$, $\mathrm{M}_X(\mathbf{x}) = \mathrm{M}_Y(\mathbf{y}) = 1$, and for all $c \in \mathbb{R}, \mathbf{v} \in V$, $\mathrm{M}_X(c\mathbf{v}) = c\mathrm{M}_X(\mathbf{v})$ and $\mathrm{M}_Y(c\mathbf{v}) = c\mathrm{M}_Y(\mathbf{v})$, then X and Y are homeomorphic.*

Proof. We first show that $\mathrm{M}_X(\mathbf{0}) = \mathrm{M}_Y(\mathbf{0}) = \mathbf{0}$. This comes from the fact that for all $c \in \mathbb{R}, \mathbf{v} \in V$, $\mathrm{M}_X(c\mathbf{v}) = c\mathrm{M}_X(\mathbf{v})$ and $\mathrm{M}_Y(c\mathbf{v}) = c\mathrm{M}_Y(\mathbf{v})$. Thus, $\mathrm{M}_X(\mathbf{0}) = \mathrm{M}_X(0\mathbf{v}) = 0\mathrm{M}_X(\mathbf{v}) = 0$. The same applies to M_Y. As M_X and M_Y are continuous, and equal 1 at the limit points of X and Y respectively, $\mathrm{M}_X(\mathbf{v}) = 0 \iff \mathrm{M}_Y(\mathbf{v}) = 0 \iff \mathbf{v} = 0$. We now show an explicit bijection between X and Y. We show the map $T : X \mapsto Y$. We let the transformation T scale points $\mathbf{x} \in X$ such that $\mathrm{M}_X(\mathbf{x}) = \mathrm{M}_Y(T(\mathbf{x}))$. Thus, we get that

$$\begin{aligned}
\mathrm{M}_X(\mathbf{x}) &= \mathrm{M}_Y(k\mathbf{x}) \\
\mathrm{M}_X(\mathbf{x}) &= k\mathrm{M}_Y(\mathbf{x}) \\
k &= \frac{\mathrm{M}_X(\mathbf{x})}{\mathrm{M}_Y(\mathbf{x})}
\end{aligned}$$

This gives us the transformation $T(\mathbf{x}) = \frac{\mathrm{M}_X(\mathbf{x})}{\mathrm{M}_Y(\mathbf{x})}\mathbf{x}$, and likewise, $T^{-1}(\mathbf{y}) = \frac{\mathrm{M}_Y(\mathbf{x})}{\mathrm{M}_X(\mathbf{x})}\mathbf{y}$. However, T and T^{-1} are undefined at $\mathbf{0}$. We fix this by letting $T(\mathbf{0}) = T^{-1}(\mathbf{0}) = \mathbf{0}$. Together, T and T^{-1} give us a bijection. We now show that both transformations are continuous. As M_X and M_Y are continuous, and $\mathrm{M}_X(\mathbf{v}) = 0 \iff \mathrm{M}_Y(\mathbf{v}) = 0 \iff \mathbf{v} = 0$, both T and T^{-1} are continuous on the domain $X \setminus \mathbf{0}$ and $Y \setminus \mathbf{0}$ respectively. We now prove continuity by showing that

$$\lim_{c\to 0}(T(c\mathbf{x})) = \mathbf{0} \quad \forall \mathbf{x} \in X$$

$$\begin{aligned}\lim_{c\to 0}(T(c\mathbf{x})) &= \lim_{c\to 0}\left(\frac{\mathrm{M}_X(c\mathbf{x})}{\mathrm{M}_Y(c\mathbf{x})}c\mathbf{x}\right)\\ &= \lim_{c\to 0}\left(\frac{\mathrm{M}_X(\mathbf{x})}{\mathrm{M}_Y(\mathbf{x})}c\mathbf{x}\right)\\ &= \lim_{c\to 0}(c)\frac{\mathrm{M}_X(\mathbf{x})}{\mathrm{M}_Y(\mathbf{x})}\mathbf{x}\\ &= \mathbf{0}\end{aligned}$$

The same logic applies to show that $\lim_{c\to 0}(T^{-1}(c\mathbf{y})) = \mathbf{0}$ for all $\mathbf{y} \in Y$. As we have shown a continuous bijection between X and Y with continuous inverse, X and Y are homeomorphic. □

D Experiments

We test the on-demand solver on a toy problem, which, however, captures a general exploration/exploitation problem with limited feedback. We compare the performance of the on-demand solver with a standard solver (such as Q learning).

D.1 Fishing

Suppose you are a person P, who fishes on a daily basis. Furthermore, suppose that you aim to maximize your cumulative D-day reward. You have access to a lake where you can trawl for fish. The lake consists of T different types of fish, each of which tend to reside at a different depth in the lake. The daily prices of each type of fish fluctuate over time. Every day, P sells his/her catch from the previous day, accumulating a profit, then selects a depth at which to fish for the current day's catch. We define P's *catch* on day t, C, as the set of fish that P *sells* on day t – that is, the set of fish that P catches on day $t-1$. We formally define C as the set containing the types of each of the fish caught.

We now formally define the state and action space. We let the current state $s_t = (C(t), t)$. It follows that the state space

$$S \equiv \left(\bigcup_{i=0}^{\infty} \{1, \dots, T\}^i\right) \times \{0, \dots, d\}$$

as we let $|C|$ be a random variable over the set of natural numbers. We formally let

$$|C| \sim [\mathcal{N}](\mu_F, \sigma_F) \tag{5}$$

where $[\mathcal{N}]$ denotes the discretized, truncated normal distribution. More specifically, modify the support of the normal distribution to be $\mathbb{N}$ by first truncating it to $\mathbb{R}^{\geq 0}$, then rounding values to the nearest natural number. We choose this distribution over a binomial distribution as it allows us to finely control the standard deviation, and thus the stochasticity in the system.

We now define our action space as the set of depths at which P can fish at. For simplicity, we let

$$A \equiv [0, 1]$$

We define the state transition function $P_a(\cdot, \cdot)$ in terms of a set of random variables $f_1, \dots, f_T$, where each f_k denotes the depth of an individual fish of type k within the lake. We let the distribution over depth f_k for an individual fish of type $k \in \{1, \dots, T\}$ be normally distributed, given by

$$f_k \sim \mathcal{N}\left(\frac{k}{T-1}, \frac{1}{6(T-1)}\right) \tag{6}$$

This choice of standard deviation comes from the fact that we want populations of fish to be stratified. As 99.7% of all fish lie within 3 standard deviations of the mean, we get that $6T\sigma = 1$. However, the fish types that reside at mean depth 0 and depth 1 only need half the spread as we can only fish on the interval $[0, 1]$. Thus, we get that $6(T-1)\sigma = 1$.

Once P chooses a depth $a \in A$ that they will trawl at, they pass a net of radius r through the lake. The probability p_k that they catch a fish of type k given that they trawl at depth a is proportional to ρ_k, where

$$\rho_k(a, r) \equiv \Phi_k(a + r) - \Phi_k(a - r),$$

and Φ_k is the cumulative distribution function for f_k. We normalize to get p_k

$$p_k(a, r) = \frac{\rho_k(a, r)}{\sum_i \rho_i(a, r)}$$

For some $s' = (C', t')$, our state transition function $P_a(\cdot, \cdot)$ given by

$$P_a(s, s') = \delta_{t+1,t'} \left(\prod_{c \in C'} p_c(a, r) \right) [\mathcal{N}] \left(|C'| \mid \mu_F, \sigma_F \right)$$

where δ refers to the *Kronecker delta function*.

We now define the reward function $R_a(\cdot, \cdot)$. The person P sells the entirety of their catch on each day. However, the prices of different fish vary over time. We let the prices of different types of fish vary in much the same way as the relative densities in P's catch vary over choice of a – in fact, we use the functions p_k to define the reward. Suppose that fish prices vary periodically with period τ. Furthermore, suppose that the lowest-priced fish will be sold for l dollars and the highest priced fish will be sold for h dollars. The price R_k of fish type k on day t is given by

$$R_k(t, K) = l + (h - l)p_k \left(.5 + \cos \left(\frac{2\pi t}{\tau} \right), K \right)$$

where K is a sort of *spread factor* that determines how varied the prices of different fish types are at a given time (analogous to the radius when we were defining the p_k). For some $s' = (C', t')$, the reward is given by

$$R_a(s, s') = \sum_{c \in C'} R_c(t', K)$$

Interestingly, our fishing problem is a simple generalization of the multi-armed bandit problem; on each time iteration t, we have a set of (uncountably) infinite arms that we can pull (one per value of a). Furthermore, the distribution over rewards for each lever changes based on the time iteration t.

We define two variants on the above problem that we will test our solver with.

Random means Our first variant involves letting the means of each of the f_k as defined in Equation (6) to be generated uniformly at random on the interval $[0, 1]$.

Size by depth Our second variation on the problem is letting μ_F as defined in Equation (5) be a function of our chosen depth $a \in A$. We let $\mu_F = \mu + \eta \sin(2\pi a)$ where $\mu, \eta \in \mathbb{R}$.

We test the (unmodified) problem under the following conditions

- $T = 5$
- $\mu_F = 30$ and $\sigma_F = 2$
- $r = 0.05$ and $K = 0.2$
- $l = 1$ and $h = 3$
- $\tau = 365$

For the size by depth variant, we let $\eta = 5$ and $\mu = 30$.

We now describe the hyperparameter optimization we performed for the Deep Q-Learning solver. We applied the same hyperparameters to the Deep Q-Learning subsolver of our solver instead of performing hyperparameter optimization each iteration in order to maintain computational tractability when running many trials. Again, due to constraints of computational tractability, we the number of training episodes to be 200. We performed 20 iterations of (uniform) randomized hyperparameter search over the learning rate and exploration decay rate. We restricted the search domain for the learning rate to be $[0, 0.1]$ and the search domain for the exploration decay rate to be $[0.75, 1)$. We set the initial exploration probability to be 1.

D.2 Tariffs

We describe an example of a mechanism design problem that will test the ability of our solver to manipulate other intelligent agents.

Suppose that we are government officials of some nation N_0 and we are in a sort of trade war with some other nations $N_1, \ldots, N_n$ over some product P. Furthermore, suppose that there are k different companies in N_0 that sell P internationally (for the sake of simplicity, these companies sell to all nations $N_0, \ldots, N_n$).

We let our action space

$$A = [0, 1]^n$$

For any $\mathbf{a} \in A$, we let a_i denote the strength of the tariff that N_0 places on N_i. We let $\mathbf{a}_t$ be the action that we take on time t. Let $a_0 = n + 1 - \sum_i a_i$.

More formally, suppose that companies from N_i would make D_i dollars in N_0 selling P if there were no tariffs in place. To apply the tariff, we change the price of P imports from N_i such that these companies instead make $D_i a_i$ dollars.

Let φ_t^i be the response from nation N_i on time iteration t to our action a_t^i. We let φ_t^i be a list of dimension 1 vectors, each element of which denotes a shipment of P sold from some N_0 company $c \in \{1, \ldots, k\}$. Furthermore,

$$\varphi_t^i = \max_{\varphi}(N_i(\varphi, \overline{a}_i(\varphi), \sim))$$

where $N_i(\varphi, \theta, \sim)$ is the utility function of N_i (the symbol $\sim$ denotes that N_i can be a function of other arbitrary factors), and $\overline{a}_i(\varphi)$ is a *learning function* which N_i uses to predict N_0's response to their action. Let $\boldsymbol{\varphi}_t = \varphi_t^1, \ldots, \varphi_t^n$. We let the current state on time t

$$s_t = (\boldsymbol{\varphi}_t, t)$$

It follows that our state space

$$S = \left(\bigcup_{i=0}^{\infty} \mathbb{N}^i \right) \times \mathbb{N}$$

We define our utility in terms of a reward function

$$R_{\mathbf{a}}(s_t, s_{t+1}) = \left[\sum_{i=1}^{n} R(\varphi_t^i) \right] + a_0^j D_0$$

where D_0 is the amount of money the k companies of N_0 would make in N_0 if there were no tariffs in place and $R(\varphi_t^i)$ gives the amount of money we would make from the shipment sales φ_t^i.

By modifying the learning functions $\overline{a}_1, \ldots, \overline{a}_n$, we can alter the *rationality* of an agent. Our solver has the challenging (yet realistic) goal of manipulating the outputs of the learning functions in order to maximize N_0's profit. We aim to validate our claim that our solver has strong application in the field of mechanism design through this complex toy problem.