The Platform Design Problem

Christos Papadimitriou, Kiran Vodrahalli, Mihalis Yannakakis
Computer Science Department
Columbia University

September 13, 2020

Abstract

On-line firms deploy suites of software platforms, where each platform is designed to interact with users during a certain activity, such as browsing, chatting, socializing, emailing, driving, etc. The economic and incentive structure of this exchange, as well as its algorithmic nature, have not been explored to our knowledge; we initiate their study in this paper. We model this interaction as a Stackelberg game between a Designer and one or more Agents. We model an Agent as a Markov chain whose states are activities; we assume that the Agent's utility is a linear function of the steady-state distribution of this chain. The Designer may design a platform for each of these activities/states; if a platform is adopted by the Agent, the transition probabilities of the Markov chain are affected, and so is the objective of the Agent. The Designer's utility is a linear function of the steady state probabilities of the accessible states (that is, the ones for which the platform has been adopted), minus the development cost of the platforms. The underlying optimization problem of the Agent — that is, how to choose the states for which to adopt the platform — is an MDP. If this MDP has a simple yet plausible structure (the transition probabilities from one state to another only depend on the target state and the recurrent probability of the current state) the Agent's problem can be solved by a greedy algorithm. The Designer's optimization problem (designing a custom suite for the Agent so as to optimize, through the Agent's optimum reaction, the Designer's revenue), while NP-hard, has an FPTAS. These results generalize, under mild additional assumptions, from a single Agent to a distribution of Agents with finite support. The Designer's optimization problem has abysmal "price of robustness" (worst-case ratio of the optimum over the robust optimum), suggesting that learning accurately the parameters of the problem is crucial for the Designer. We discuss other implications of our results and directions of future research.

Contents

1 Introduction

In mainstream economics, the production of wealth happens through markets: environments in which firms employ land, labor, capital, raw materials, and technology to produce new goods for sale, at equilibrium prices, to consumers and other firms. Since all agents in this scenario participate voluntarily, wealth must be created. Accordingly, markets have been the focus of a tremendous intellectual effort by economists, mathematicians, and, more recently, computer scientists.

Over the past three decades the global information environment has spawned novel business models seemingly beyond the reach of the extant theory of markets, and which, arguably, account for a large part of present-time wealth production, chief among them a new kind of software company that can be called *platform designer*. On-line platforms are created with which consumers interact during certain activities: search engines facilitate browsing, social networks host social interactions, movie, music, and game sites provide entertainment, chatting and email apps mediate communication. Shopping platforms, navigation maps, tax preparation sites, and many more platforms bring convenience and therefore value to consumers' lives. Increasingly during these past two decades, on-line firms have created comprehensive *suites* of platforms, covering many such life activities. Platform designers draw much of their their revenue through the data that they collect about the users interacting with their platforms, which data they either sell to other firms or use to further fine tune and enhance their own business. In this paper we point out that, in the case of platform designers, the most elementary aspects of markets, for example the theory of production and consumption, are quite nontrivial, and in fact in very unfamiliar directions.

Our Model and Results We model the platform design problem as a Stackelberg game (that is, a game where one player goes first and the others react optimally) with two players, a Designer and an Agent (the extension to many Agents is also studied). Here, the Designer plays first, and the Agent(s) respond. The Agent is modeled as a controlled Markov chain, which is ergodic under every Agent policy, on a set of states $\mathcal{A}$, representing the Agent's life activities. We assume that the Agent receives a fixed payoff per unit of time spent at each state. The Designer has the opportunity to design a platform for each state in $\mathcal{A}$, which the Agent may or may not choose to adopt. There is a one-time cost for the Designer to build a platform for a given state. If the Agent adopts the platform, the transitions of the Agent's life change at that state, and the Agent's utility at that state may increase or decrease as a result of adoption[1]. In return, the Designer gets to observe the Agent at that state and derives a fixed utility payoff for the fraction of time the Agent spends in that state. We note that the Agent's optimization problem, as posed, is a Markov decision process (MDP), and it follows from MDP theory that the Agent will adopt some of the platforms offered and reject the rest — and there is no loss of generality in assuming that they fully adopt

[1]One possible reason for diminished utility is the aversion of the Agent to the Designer's access to personal information pertaining to that state.

the platforms that they do adopt even partially — and the optimum set of adopted platforms can be computed by linear programming.

In this paper we focus on a particular kind of Markov chain that we call *the flower* (see Figure 1), with a number of transition parameters that is linear in $|\mathcal{A}|$. We assume that, at each state i, the transition leads back to the state with some probability q_i, while the rest of the probability $(1-q_i)$ is split among the other states *in proportions that are fixed.* Evidently, this is equivalent with a chain that has an extra "rest state" 0 with $q_0 = 0$, that is, a purely transitional state (see the figure). Adopting a platform now increases or decreases the transition probability of the state to itself, decreasing or increasing, respectively, the transition probability to the other states. We show that, in this case, the MDP optimizing the Agent's objective, given the available platforms, becomes a quasiconcave combinatorial optimization problem with special structure (Lemma 1), which can be solved by a greedy algorithm (Theorem 1)[2].

Now the *platform design problem* (PDP) is the following: Given the Markov chain, all utility coefficients for both the Agent and the Designer, and the development costs of the platforms, choose a set of states S for which to create platforms, so as to maximize the Designer's utility: The utility to the Designer of the Markov chain that results by the optimum response (through the greedy algorithm) by the Agent, minus the development costs of the platforms in S. PDP is NP-hard (Theorem 3), but has a dynamic programming FPTAS if one parameter — the expected time spent at each state — is quantized (Theorem 2). Finally, the dynamic programming algorithm can be extended, through some further quantization, to the case of many agents — except that the number of agents is now in the exponent (Theorem 4). Given that the number of agents is likely to be very large, the best way to think of this algorithm is as an algorithm for the case in which one is given a *distribution* of agents of finite support — that is, with a small number of *agent types.*

Related Work We are not aware of past research on the production and consumption of online platforms. Computational aspects of Stackelberg games between consumers and firms designing or packaging on-line products have been explored to a small degree, see e.g. (Kleinberg et al., 1998a,b). There has been work on online decision making, where at each round the Designer gets to select from some set of options (e.g., which is the best ad to display to the user of a website) and receives a reward after deployment for that round, as well as additional information about the performance of the other options (Cesa-Bianchi and Lugosi, 2006); see also (Frazier et al., 2014; Mansour et al., 2015; Roughgarden and Wang, 2016; Liu and Ho, 2018; Lykouris et al., 2019). This line of research is of obvious relevance to the present one, even though our Agent model is far more complex. More recently, trade-offs in on-line activity by consumers, for example between effectiveness of browsing and privacy, have been discussed (Tsitsiklis and Xu, 2018; Tsitsiklis et al., 2018). The ways in which on-line firms profit from data has been somewhat explored, see e.g. (Agarwal et al., 2019) but not in any manner that can be used in our model; here we consider it a given parameter.

Our Contributions Our main contributions are: the articulation of the Platform Design Problem, its solution through the Agent's greedy algorithm and dynamic programming, the robustness analysis, and the many directions for further research opened (see the discussion in Section 6).

[2]This result can be extended in a straightforward way to the case where the designer has the option to design several versions of each platform, each version with its own cost and modifications of the transitions and utilities, and must choose to deploy one or none of these options for each state in $\mathcal{A}$.

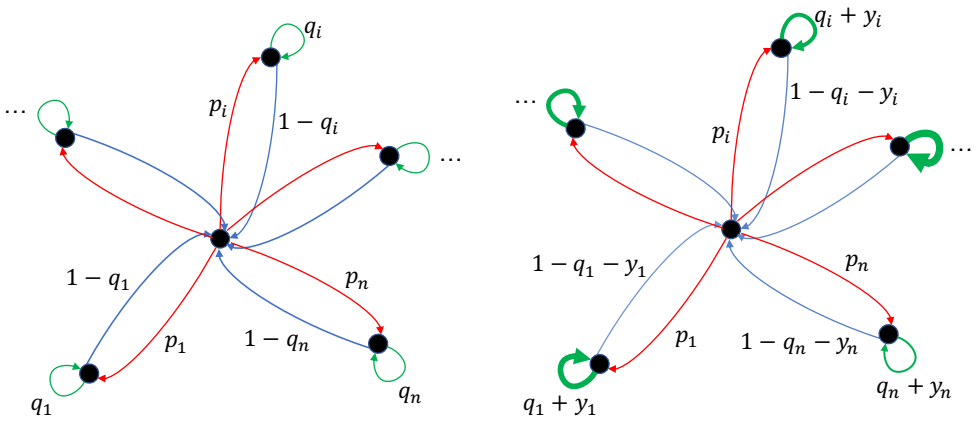

Figure 1: T_{life}(left) and T_{platform}(right).

2 The Agent's Problem

We model the Agent's behavior via an optimal policy produced by solving an average reward Markov Decision Process (MDP). In MDPs, one can consider general non-stationary randomized policies. However, it is well known that one can restrict, without loss of optimality, to deterministic, Markovian, stationary policies, where there is a particular action to be applied each time a particular state is visited. From now on, we use the term "policy" to mean "policy within this restricted class." For an introduction to MDPs, see Puterman (Puterman, 1994).

Definition 1 (Agent MDP)**.** [3] We have state space $\mathcal{A}$ with $n+1$ states and with two actions for the Agent, "adopt platform" and "do not adopt platform." Let a^0 denote "do not adopt," and a^1 denote "adopt." Define

$$R(i,a^0)=c_i^{\text{life}};\quad R(i,a^1)=c_i^{\text{platform}}\quad \forall i\in[n]=\{1,\dots,n\};\quad R(0,\cdot)=0$$

for fixed constants $c_i^{\text{life}}, c_i^{\text{platform}}$. In Figure 1, we define the transitions of the Markov chain T_{life} to represent the Agent's life, and T_{platform} to represent the Agent's life when the platform is adopted at all states. Here, p_i, q_i, y_i satisfy $\sum_i p_i = 1$, $0 < p_i$, $0 < q_i < 1$, and $0 < y_i < 1 - q_i$ for all $i \in [n]$.

At each state $i \in [n]$, the Agent can choose the transition probabilities out of that state (i.e., the i^{th} row of the transition probability matrix) to be those of either T_{life} or of T_{platform}; this choice corresponds to action a^0 and a^1 respectively at each state $i \in [n]$. At state 0, the action chosen by the Agent does not affect the transitions, since the Designer never builds a platform there. Finally, note that we assume that T_{life} is irreducible, and thus has a unique stationary distribution. The restrictions on y_i ensure this property is preserved for any choice of Agent policy.

Then, the agent attempts to maximize average reward:

Definition 2 (Average Reward)**.** For some fixed starting point $i_0 = 0$ and arbitrary a_0, average reward due to policy ρ is

$$R(\rho) := \lim_{T\to\infty} \mathbb{E}_{i_1,a_1,\cdots,i_T,a_T\sim\rho} \frac{1}{T}\sum_{t=1}^{T} R(i_t,a_t)$$

[3] Note that the rest state 0 is for convenience and is not necessary in our model. We could equivalently have a graph where each node i transitions to node j with probability $(1-q_i-y_i)\cdot p_j$, and self-transitions with probability $q_i + y_i + p_i(1-q_i-y_i)$.

2.1 The Greedy Algorithm

Irreducible average-reward MDPs are efficiently solvable via linear programming, value and policy iteration, etc. (Bertsekas, 2017). Here, we reformulate the Agent's problem as a combinatorial optimization problem with special structure, and solve it through a greedy algorithm. The following is straightforward:

Lemma 1. *The agent's objective for an optimal policy defined in Definitions 1 and 2 can be re-written as the following optimization:*

$$\underset{S\subseteq[n]}{\operatorname{argmax}} \frac{A+\sum_{j\in S} z_j \phi(j)}{B+\sum_{j\in S} z_j} \tag{1}$$

where

$$A := \sum_{i=1}^{n} \lambda_i c_i^{\text{life}}; \quad B := 1+\sum_{i=1}^{n} \lambda_i; \quad \lambda_i = \frac{p_i}{1-q_i}; \quad z_i = \frac{p_i}{1-q_i-y_i} - \frac{p_i}{1-q_i} \geq 0;$$

$$\phi(i) := \begin{cases} c_i^{\text{platform}} + \frac{\lambda_i}{z_i}\left(c_i^{\text{platform}} - c_i^{\text{life}}\right) & \text{if } z_i > 0 \\ 0 & \text{if } z_i = 0 \end{cases};$$

We therefore define

$$\text{utility}^{\text{Agent}}(S) := \frac{A+\sum_{j\in S} z_j \phi(j)}{B+\sum_{j\in S} z_j}$$

Proof. See Appendix A. □

We note that here we assume that $y_i > 0$ at each state — that is to say, adopting the platform increases a state's recurrence probability. This assumption is not necessary, and the general case can be handled in a similar way by modifying the greedy algorithm to pay attention to signs (see Appendix D). We also reiterate that the solution to the original average case MDP problem need not be unique. Therefore, the argmax solution to Equation 1 has many potential solutions.

The optimization problem formulated in Lemma 1 can be solved in polynomial time. The intuitive reason is this: Looking at the fractional objective function, we note that it is the ratio of two linear functions of the combinatorial (integer) variables implicit in S, and such functions are known to be quasiconvex. It is therefore no huge surprise that a greedy algorithm solves it — however, the details are rather involved. Incidentally, one could arrive at the same algorithm by tracing the simplex algorithm on the MDP linear program[4].

Theorem 1. *Algorithm 1 returns*

$$S^* \in \underset{S\subseteq[n]}{\operatorname{argmax}}\ \text{utility}^{\text{Agent}}(S)$$

That is, the policy

$$\pi(s) = \begin{cases} a^1 & \text{if } s \in S^* \\ a^0 & \text{o.w.} \end{cases}$$

is an optimal policy.

[4]Many thanks to John Tsitsiklis for pointing this out to us.

ALGORITHM 1: GREEDY ALGORITHM

Input: Parameters of the Agent's problem: transition probabilities and utility coefficients in and out of the platform.

Output: An optimal subset $S \subseteq [n]$ of states where the Agent accepts the platform.

Initialize $S := \{\}$

for $k \in [n]$ *sorted from largest to smallest* $\phi(i)$ **do**

  if $\text{utility}^{\text{Agent}}(S) < \phi(k)$ **then**

    Update $S := S \cup \{k\}$

  else

    return S

  end

end

return S

Before we prove the theorem, we give a useful definition and a lemma.

Definition 3 (Contiguous policy)**.** We say a policy S is contiguous if the states in the policy are the first m states in order sorted by ϕ, for some value of m.

Lemma 2.

$$\frac{x}{y} < \frac{r}{s} \iff \frac{x}{y} < \frac{x+r}{y+s} < \frac{r}{s} \quad \text{where } y, s > 0.$$

Proof. Since $y, s > 0$ and thus $y + s > 0$, cross-multiply and simplify to get the desired inequalities. □

With this lemma in hand, we prove Theorem 1.

Proof of Theorem 1. We can prove optimality in two steps.

1. First we will show that any non-contiguous policy is dominated by a contiguous policy. Thus an optimal policy must be contiguous.
2. Then, we show that the greedy algorithm necessarily finds a best contiguous policy (e.g., an optimal stopping point).

We begin with the first step. Suppose we have a non-contiguous policy S. Let state $\ell \in [n]$ be a "missing piece" (e.g., if we index by sorted order and S contained $1, 2, 4, 5, 7$, missing pieces would be 3 and 6). This $\ell \in [n]$ necessarily exists since S is non-contiguous. Now there are two cases.

Case 1:

$$\text{utility}^{\text{Agent}}(S) < \phi(\ell)$$

We apply Lemma 2 to show that adding state ℓ results in improvement in the objective.

Case 2:

$$\text{utility}^{\text{Agent}}(S) \geq \phi(\ell)$$

Here we show that removing all states $k \in S$ where $\phi(k) < \phi(\ell)$ improves the objective. If equality holds, then it does not matter whether we add the state to the objective, so for simplicity, we terminate at equality. From the assumption and the definition of ϕ, we have

$$\text{utility}^{\text{Agent}}(S) \geq \phi(\ell) > \phi(k)$$

for all such k. By Lemma 2, removing state k increases the ratio, e.g.

$$\text{utility}^{\text{Agent}}(S \setminus \{k\}) > \text{utility}^{\text{Agent}}(S) \geq \phi(\ell) > \phi(k)$$

The same argument applies to all $k' \in S \setminus \{k\}$ such that $\phi(k') \leq \phi(k)$ as well. Therefore, we can remove all the k' with score less than the score of ℓ and improve the objective.

After a single round of considering a missing state ℓ (where either case 1 or case 2 applies), we produce a new S', which can again be non-contiguous. However, the maximum index present in the new S' has either decreased (if the second case happened and we deleted everything worse than ℓ) or we filled in the missing state ℓ. Either way, the number of missing pieces has strictly decreased and we have added no new missing states. Using induction on the number of missing pieces proves that iterating over all original missing pieces will "fill in all the gaps" and produce a contiguous policy S^* which is strictly better than the original non-contiguous policy S.

Finally, we show the greedy algorithm selects an optimal contiguous policy. Let the output of the greedy algorithm be $\hat{S}$. The desired result directly follows since if the next state ℓ satisfies

$$\text{utility}^{\text{Agent}}(\hat{S}) \geq \phi(\ell)$$

and is not selected, since all smaller states (sorted by ϕ) are less than or equal to $\phi(\ell)$, any contiguous subset of the smaller states is an effective average which is $\leq \phi(\ell)$, and any contiguous subset of future states is worse off. Thus, the greedy algorithm produces a maximal solution. □

3 The Designer's Problem

The Designer's problem is a bi-level optimization over the output of the Agent's greedy algorithm. The Designer must choose a set S of states for which to design platforms, pays a fixed cost cost_j for the platform at each state j in S, and then gets a reward rate d_j for every state in S, provided the Agent opts in to the platform. For each state j in S, we assume there is only one possible platform the Designer may choose to create, associated with increasing the probability the Agent stays at state j by a fixed y_j. The Designer's reward for each state j that the Agent adopts is equal to the rate d_j for the state times the fraction of the time that the agent spends in state j. Let $\text{Agent}(S)$ denote the subset of states that the Agent adopts when the Designer offers platforms for the subset S of states. The fraction of the time that the Agent spends in state $j \in \text{Agent}(S)$ is $\frac{\frac{p_j}{1-q_j-y_j}}{B+\sum_{i\in\text{Agent}(S)} z_i}$, using the notation of Section 2 given in Definition 1 and Lemma 1 (see Appendix A for the stationary distribution of the Markov chain). Thus, the Designer's *profit* can be written as

$$\text{profit}(S) := \frac{\sum_{j\in\text{Agent}(S)} d_j \cdot \frac{p_j}{1-q_j-y_j}}{B + \sum_{i\in\text{Agent}(S)} z_i} - \sum_{j\in S} \text{cost}_j$$

Since the response of the Agent through the greedy algorithm of the previous section can be fully anticipated by the Designer — recall that the parameters of this response are known to the Designer — it is clear that, at optimality, all states in S will be accepted by the agent. Call a set S of states *feasible* if $\text{Agent}(S) = S$. Therefore, only feasible sets S need be considered. It is easy to see from the definition and the greedy algorithm of Section 2 that if a set S is feasible then so are all its subsets.

If for some state i, $\text{profit}(\{i\}) \leq 0$, then it follows that for all sets S that contain i we have $\text{profit}(S) \leq \text{profit}(S - \{j\})$. Hence, there is no reason to build a platform at i, and we can ignore i. Thus, we may restrict our attention to the states i such that $\text{profit}(\{i\}) > 0$. We may assume also

without loss of generality that every state i by itself is feasible: If $\{i\}$ is not feasible, then neither is any set that contains i, therefore we can ignore i.

Let $K = \max_i \text{profit}(\{i\})$. It is easy to see that for any set S, $\text{profit}(S) \leq \sum_{i \in S} \text{profit}(\{i\})$. Therefore, the optimal profit OPT is at most nK and at least K.

We will assume that the cost cost_i of building a platform at any site i is not astronomically larger than the anticipated optimal profit, specifically we assume $\text{cost}_i \leq rK$ for some polynomially bounded factor r.

Furthermore, and importantly, for our dynamic programming FPTAS to work in polynomial time, a discretization assumption is necessary. For each state i, the platform available will change (increase or decrease) the term $\frac{p_i}{1-q_i}$ appearing in the numerator and the denominator of the Agent's objective by an additive z_i. *We assume that all these z_i's are multiples of the same small constant* δ (think of δ as 1%). This assumption means that there are $O\left(\frac{1}{\delta}\right)$ possible values of the denominator, and ensures the dynamic programming is polynomial-time. One should think of this maneuver as one of the compromises (in addition to accepting a slightly suboptimal solution) for the approximation of the whole problem. We suspect that the problem has no FPTAS without this assumption, although there is a pseudo-polytime algorithm.

3.1 FPTAS for the PDP

The Platform Design Problem is approximately solvable in polynomial time; in this section we present a FPTAS which returns a $(1-\epsilon)$-approximate solution. Our approach is inspired by the FPTAS for knapsack presented in Ibarra and Kim (1975). We also note that our algorithm relies on the structure of the greedy algorithm presented in Algorithm 1.

The algorithm uses dynamic programming. It employs a 3-dimensional hash table, called SET, into which the sets under consideration are being hashed. The hash function has three components that correspond to the following components of the profit of the set, scaled and rounded appropriately to integers: (1) the whole profit $\text{profit}(S)$, (2) the first term in the profit, denoted $P_1(S)$, and (3) the denominator of the first term, denoted $\mathbf{D}(S)$ (which note, is also the denominator of the agent's objective function). We use $\mathbf{N}(S)$ to denote the numerator of the agent's objective function.

Lemma 3. *Let $S, S' \subseteq [k]$ be two sets that hash in the same bin and suppose that $\boldsymbol{N}(S) \leq \boldsymbol{N}(S')$. Then for every set $T \subseteq \{k+1, \ldots, n\}$, if $S' \cup T$ is feasible then $S \cup T$ is also feasible, and* $\text{profit}(S \cup T) \geq \text{profit}(S' \cup T) - \epsilon K/n$.

Proof. Since S, S' hash in the same bin, they have the same denominator $\mathbf{D}(S) = \mathbf{D}(S')$. Hence also $\mathbf{D}(S \cup T) = \mathbf{D}(S' \cup T)$. Furthermore, $\mathbf{N}(S) \leq \mathbf{N}(S')$ implies that $\mathbf{N}(S \cup T) \leq \mathbf{N}(S' \cup T)$. Since $S' \cup T$ is feasible it follows that $S \cup T$ is also feasible.

Consider the difference $\text{profit}(S' \cup T) - \text{profit}(S \cup T)$. From the formulas it follows that this difference is equal to $\text{profit}(S') - \text{profit}(S) + (P_1(S) - P_1(S'))\frac{\sum_{i \in T} z_i}{\mathbf{D}(S) + \sum_{i \in T} z_i}$. Since S, S' hash in the same bin, $|\text{profit}(S') - \text{profit}(S)| \leq \epsilon K/2n$ and $|P_1(S) - P_1(S')| \leq \epsilon K/2n$. Therefore, $|\text{profit}(S' \cup T) - \text{profit}(S \cup T)| \leq \epsilon K/n$. □

Lemma 4. *For every $k = 0, 1, \ldots, n$, after the k^{th} iteration of the loop, there is a set S in the hash table that can be extended with elements from $\{k+1, \ldots, n\}$ to a feasible set that has profit $\geq \text{OPT} - \epsilon k \cdot K/n$.*

Proof. By induction on k. The basis, $k = 0$ of the claim is trivial: The hash table contains initially $\emptyset$, which can be extended to an optimal solution.

ALGORITHM 2: DESIGNER'S FPTAS FOR THE PDP

Input: The parameters of the PDP: transition probabilities, utility and cost coefficients for the Agent and the Designer, and small positive reals ϵ, δ

Output: A $(1-\epsilon)$-approximately optimal subset of states S^* for which to deploy platforms.

$\mathbf{N}(S)$ and $\mathbf{D}(S)$ denote the numerator and the denominator of the agent's objective function, with the constant terms omitted

$P_1(S)$ denotes the first term in the designer's profit function

SET is a hash table of subsets of $[n]$ indexed by triples of integers

The hash function is $\text{hash}(S) := \left(\lceil \frac{\text{profit}(S)}{\epsilon K/2n} \rceil, \lceil \frac{P_1(S)}{\epsilon K/2n} \rceil, \mathbf{D}(S)/\delta\right)$

Initialize the hash table SET to contain only the empty set in the bin $(0,0,0)$

for $k \in [n]$ **do**
  for $S \in$ SET in lexicographic order **do**
    $S' := S \cup \{k\}$
    if *Agent will adopt all platforms in* S' *and* $\text{profit}(S') > 0$ **then**
      if $\text{hash}(S') \in$ SET **then**
        $\hat{S} := \text{SET}[\text{hash}(S')]$
        if $\boldsymbol{N}(\hat{S}) > \boldsymbol{N}(S')$ **then**
          $\text{SET}[\text{hash}(S')] := S'$
        end
      else
        $\text{SET}[\text{hash}(S')] := S'$
    end
  end
end
return *the set* S *in the hash table with largest first hash value*

For the induction step, assume that the property holds after the $(k-1)^{st}$ iteration for a set $S \subseteq [k-1]$ in the table, and let $T \subseteq \{k, \dots, n\}$ be an extension that yields a feasible set $S \cup T$ with profit within $\epsilon(k-1) \cdot K/n$ of OPT. Suppose first that $k \notin T$. At the end of the k^{th} iteration, the hash table contains either S or another set S' that hashes in the same bin and has replaced S, thus S' has the same denominator but smaller numerator. In the latter case, by Lemma 3, $S' \cup T$ is also feasible, and its profit is within $\epsilon K/n$ of the profit of $S \cup T$, hence it is within $\epsilon k \cdot K/n$ of OPT.

The argument in the case $k \in T$ is similar. Since $S \cup T$ is feasible, $S \cup \{k\}$ is also feasible, thus the algorithm will hash it and either insert it in the table or not depending on whether there is another "better" set already in the same bin. At the end of the k^{th} iteration, the hash table will contain either the set $S \cup \{k\}$ or a set $\hat{S}$ that is at least as good (has at least as low numerator) in the corresponding bin. Whichever of these sets is in that bin at the end of the iteration satisfies the property. This is obvious for $S \cup \{k\}$, and it follows from Lemma 3 for $\hat{S}$: Since $S \cup T = (S \cup \{k\}) \cup (T - \{k\})$ is feasible, and $\hat{S}$ hashes in the same bin as $S \cup \{k\}$ and is at least as good, $\hat{S} \cup (T - \{k\})$ is also feasible, it has profit within $\epsilon K/n$ of $S \cup T$, hence within $\epsilon k \cdot K/n$ of OPT. □

Theorem 2. *Algorithm 2 is a FPTAS for the Platform Design Problem.*

Proof. Lemma 4 for $k = n$ tells us that at the end, the table contains a set S whose profit is within ϵK of OPT. Since $OPT \geq K$, the profit of S is at least $(1-\epsilon)OPT$.

Regarding the complexity of the algorithm, note that the three dimensions of the hash table have respectively size $O(n^2/\epsilon)$ (since the maximum profit is at most nK), $O(rn^2/\epsilon)$, and n/δ. In every iteration the algorithm spends time proportional to the number of sets stored the table. In particular, the algorithm only needs linear time to check the feasibility of each S' as well as calculate $\mathbf{N}(S')$ and profit(S'). Thus, the total time is polynomial in n and $\frac{1}{\epsilon}$. □

It turns out that an FPTAS is the best we could hope for, even if all $z_i = 1$:

Theorem 3. *The PDP is NP-complete.*

We prove Theorem 3 in Appendix B.

4 An Algorithm for Many Agents

We have k Agents, each with their own flower Markov chain. We will use the notation of Section 2, with an additional subscript i for each Agent i. Thus, for example $p_{ij}, q_{ij}, y_{ij}; j \in [n]$ denote the parameters of the Markov chain of Agent i, $\phi_i(j)$ denotes the potential of state j for Agent i. The utility of Agent i if he adopts the platforms in a set S of states is $u_i(S) = \frac{A_i + \sum_{j \in S} z_{ij}\phi_i(j)}{B_i + \sum_{j \in S} z_{ij}}$.

The Designer will offer platforms for a set S of states. If the platform at state j is adopted by Agent i, then the Designer gets reward at a rate d_{ij}, i.e. gets reward equal to d_{ij} times the fraction of the time that Agent i spends at state j. The cost of building a platform for state j is cost_j. Thus, the Designer's profit function is:

$$\text{profit}(S) := \sum_i \frac{\sum_{j \in \text{Agent}_i(S)} d_{ij} \cdot \frac{p_{ij}}{1-q_{ij}-y_{ij}}}{B_i + \sum_{l \in \text{Agent}_i(S)} z_{il}} - \sum_{j \in S} \text{cost}_j$$

where $\text{Agent}_i(S)$ is the set of states that Agent i chooses when offered platforms in the set S, i.e. the set chosen by the greedy algorithm of Section 2.

We will assume in this section that, besides the parameters z_{ij}, also the potentials $\phi_i(j)$ are quantized. That is, we assume that both the z_{ij}'s and the $\phi_i(j)$'s are polynomially bounded integer multiples of some small amounts; i.e., each z_{ij} is of the form $l_{ij}\delta$ for some integer $l_{ij} \leq M$ and some δ, with M polynomially bounded, and similarly each $\phi_i(j)$ is of the form $l'_{ij}\delta'$ for some integer $l'_{ij} \leq M$ and some δ'. This implies then that the numerator and denominator of the utility function $u_i(S)$ of an Agent for the various sets S can take a polynomial number of possible values. We will show that under this assumption, the optimal solution can be computed in polynomial time for a fixed number of Agents.

For each Agent i, let $\Phi_i = \{\phi_i(j)|j \in [n]\} \cup \{\infty\}$. Let $\mathcal{D}_i = \{B_i + l\delta | l \in [nM]\}$, $\mathcal{N}_i = \{A_i + l\delta\delta' | l \in [nM^2]\}$. Note that $|\Phi_i|, |\mathcal{D}_i|, |\mathcal{N}_i|$ are polynomially bounded by our assumption. By the definitions, for every subset S of states, the numerator of the utility $u_i(S)$ is in $\mathcal{N}_i$ and the denominator is in $\mathcal{D}_i$. Let $\Phi = \Pi_{i=1}^k \Phi_i$, $\mathcal{D} = \Pi_{i=1}^k \mathcal{D}_i$, and $\mathcal{N} = \Pi_{i=1}^k \mathcal{N}_i$.

For any $\theta_i \in \Phi_i$, define $Q_i(\theta_i) = \{j | \phi_i(j) \geq \theta_i\}$. For any tuple $\theta \in \Phi$ and tuple $D \in \mathcal{D}$, define a corresponding value coefficient c_j for state j to be

$$c_j(\theta, D) = \sum_{i: j \in Q_i(\theta_i)} \frac{d_{ij} p_{ij}}{(1 - q_{ij} - y_{ij}) D_i} - \text{cost}_j$$

That is, the summation in the above formula includes only those $i \in [k]$ such that $j \in Q_i(\theta_i)$.

The algorithm is given below. It uses dynamic programming. For every tuple $\theta \in \Phi$ and $D \in \mathcal{D}$, it computes an optimal set S such that, if the Designer offers platforms in the subset S of states, then Agent i selects all states of S that have $\phi_i(j) \geq \theta_i$ (i.e. $\text{Agent}_i(S) = S \cap Q_i(\theta)$), and the denominator of $u_i(S)$ is D_i. The algorithm then returns the best S that it finds over all tuples $\theta \in \Phi$ and $D \in \mathcal{D}$.

For every tuple $\theta \in \Phi$ and $D \in \mathcal{D}$, the algorithm processes the states in (arbitrary) order $1, \ldots, n$. It employs a hash table H indexed by two k-tuples a, b of integers, where $a \in ([M^2])^k$, $b \in M^k$, represent respectively the integer parts of the numerators and denominators of the utility functions of the Agents for a set. Each entry $H(a, b)$ of the hash table is either empty or contains a subset of the states processed so far that hashes into this slot. A set S of states hashes into the slot (a, b) where $a_i = \sum_{j \in S \cap Q_i(\theta_i)} \frac{z_{ij}\phi_i(j)}{\delta\delta'}$ and $b_i = \sum_{j \in S \cap Q_i(\theta_i)} \frac{z_{ij}}{\delta}$ for all Agents $i \in [k]$. We can implement H by a search data structure that contains only the slots (a, b) that are nonempty and for each one of them records the corresponding set S and its value with respect to θ, D. We define the *value* of a set S to be $\text{value}_{(\theta,D)}(S) = \sum_{j \in S} c_j(\theta, D)$.

Initially the hash table H contains only the empty set in the slot $(0, 0)$. After the algorithm processes all the states, it examines the slots (a, b) that are consistent with the pair (θ, D) in the following sense. For each $i \in [k]$, let $\theta'_i = \max_j\{\phi_i(j) | \phi_i(j) < \theta_i\}$, i.e. the next smaller value of a potential $\phi_i(j)$ after θ_i; if $\theta_i = \infty$, then θ'_i is the maximum $\phi_i(j)$, and if θ_i is the smallest $\phi_i(j)$, then set $\theta'_i = -1$. We say that the pair (a, b) is *consistent* with (θ, D) if $D_i = B_i + b_i\delta$, and $\theta_i > \frac{A_i + a_i\delta\delta'}{D_i} \geq \theta'_i$ for all $i \in [k]$. The algorithm sets $S(\theta, D)$ to be the set $H[a, b]$ with the largest $\text{value}_{(\theta,D)}$ among the consistent slots (a, b). At the end, after the algorithm has processed all the pairs (θ, D), it returns the set $S(\theta, D)$ with the largest value.

Lemma 5. *For every pair $(\theta, D) \in (\Phi, \mathcal{D})$, if a set S hashes into a slot (a, b) that is consistent with (θ, D), then* $\text{value}_{(\theta,D)}(S) = \text{profit}(S)$. *In particular, the set $S(\theta, D)$ selected by the algorithm (if any) satisfies* $\text{value}_{(\theta,D)}(S(\theta, D)) = \text{profit}(S(\theta, D))$.

Proof. Let $(a, b) = \text{hash}(S)$. The slot is consistent with (θ, D), thus, $D_i = B_i + b_i\delta$, and $\theta_i > \frac{A_i + a_i\delta\delta'}{D_i} \geq \theta'_i$ for all $i \in [k]$. Since S hashes into slot (a, b), we have $a_i = \sum_{j \in S \cap Q_i(\theta_i)} \frac{z_{ij}\phi_{ij}}{\delta\delta'}$ and $b_i = \sum_{j \in S \cap Q_i(\theta_i)} \frac{z_{ij}}{\delta}$ for all $i \in [k]$.

ALGORITHM 3: Designer's multiAgent algorithm for the PDP

Input: The parameters of the PDP: transition probabilities, utility and cost coefficients for the Agents and the Designer

Output: An optimal subset of states S^* for which to deploy platforms.

for each $\theta \in \Phi, D \in \mathcal{D}$ **do**
  Initialize the hash table H to contain only the empty set in the slot $(0,0)$
  for $t \in [n]$ **do**
    for each nonempty slot (a,b) of H **do**
      $S = H(a,b)$; $S' := S \cup \{t\}$
      if hash$(S') \in$ H **then**
        $\hat{S} := H[\text{hash}(S')]$
        if $\text{value}_{(\theta,D)}(\hat{S}) < \text{value}_{(\theta,D)}(S')$ **then**
          $H[\text{hash}(S')] := S'$
        end
      else
        $H[\text{hash}(S')] := S'$
    end
  end
  $S(\theta, D) := \underset{H(a,b)}{\text{argmax}} \{\text{value}_{(\theta,D)}(H(a,b)) \mid (a,b) \text{ is consistent with } (\theta, D)\}$
end
return $\underset{S(\theta,D)}{\text{argmax}} \{\text{value}_{(\theta,D)}(S(\theta,D)) \mid (\theta,D) \in (\Phi, \mathcal{D})\}$

For each Agent i, consider the operation of the greedy algorithm when offered the set of platforms S. Since $\theta_i > \frac{A_i + a_i \delta \delta'}{D_i} \geq \theta_i'$, the greedy algorithm will select precisely all states j of S that have potential $\geq \theta_i$, i.e. $\text{Agent}_i(S) = S \cap Q_i(\theta_i)$. Therefore, $\text{profit}(S) = \text{value}_{(\theta,D)}(S)$. □

We remark incidentally that if a slot is not consistent with (θ, D), then the value of its set may not be equal to its profit (the profit may be higher or lower).

Lemma 6. *Let S^* be an optimal solution to the Platform Design Problem, and let $\theta_i = \min_{j \in \text{Agent}_i(S^*)}\{\phi_i(j)\}$, $D_i = B_i + \sum_{j \in \text{Agent}_i(S^*)} z_{ij}$. Then, in the iteration for the pair (θ, D), the algorithm selects a set $S(\theta, D)$, and the set has $\text{profit}(S(\theta, D)) \geq \text{profit}(S^*)$.*

Proof. Consider the iteration of the algorithm for the pair (θ, D). We can show by induction on $t = 0, 1, \ldots, n$, that after stage t, the slot hash$(S^* \cap [t])$ is nonempty, and the value of the set in the slot is at least the value of $S^* \cap [t]$.

The claim is obviously true in the beginning. Consider stage t. If $t \notin S^*$, then the slot $\text{hash}(S^* \cap [t-1]) = \text{hash}(S^* \cap [t])$, and the set in this slot at the end of stage t is either the same as the set after stage $t-1$, or another set with higher value; thus the claim follows from the induction hypothesis.

Suppose $t \in S^*$, let $(a,b) = \text{hash}(S^* \cap [t-1])$, and let $S = H(a,b)$. By the induction hypothesis, $\text{value}_{(\theta,D)}(S) \geq \text{value}_{(\theta,D)}(S^* \cap [t-1])$. Then $S \cup \{t\}$ hashes in the same slot as $S^* \cap [t]$, and $\text{value}_{(\theta,D)}(S \cup \{t\}) = \text{value}_{(\theta,D)}(S) + c_t(\theta, D) \geq \text{value}_{(\theta,D)}(S^* \cap [t-1]) + c_t(\theta, D) = \text{value}_{(\theta,D)}(S^* \cap [t])$. At the end of stage t, this slot has either the set $S \cup \{t\}$ or another set with a higher value.

After stage n, the set S^* hashes into a slot (a,b) that is consistent with (θ, D) from our choice of θ and D. By the claim above, this slot has a set that has equal or larger value. By Lemma 5,

the value is equal to the profit. Therefore, $S(\theta, D)$ exists and $\text{profit}(S(\theta, D) \geq \text{profit}(S^*)$. □

Optimality follows directly from the lemmas.

Theorem 4. *Algorithm 3 computes an optimal solution to the Designer's problem. It runs in polynomial time for fixed number of Agents, under the stated assumptions on the input parameters.*

For one Agent we gave in the previous Section an FPTAS under the weaker assumption that the z parameters are polynomially bounded, but not necessarily the potentials ϕ. We can show that there is no such FPTAS for two Agents, if the ϕ are not restricted.

Theorem 5. *Unless P=NP, there is no FPTAS for the Designer's problem with two Agents if the $\phi_i(j)$ are not restricted to be polynomially bounded.*

The proof is given in Appendix C.

5 Robustness

What if — as it is likely in reality — the Designer does not know precisely the Agent's parameters? For simplicity, we address this important question (a first example of the many open questions explored in the discussion section) for one Agent, and when the platform utility coefficients for the Agent are subject to small perturbations from the known estimates; it will be clear that a very similar approach is possible for more general cases, for example when uncertainty prevails for all parameters.

We shall next treat this question through the well-developed theory of robustness for optimization problems (Ben-Tal et al., 2009; Bertsimas and Sim, 2004). Instead of merely using the parameter estimates directly to define the optimization problem, we encode our uncertainty into a new optimization problem:

Definition 4 (Robust Optimization and Price of Robustness)**.** Given the parameter vector X, we consider the set of all possible parameter vectors $\mathcal{X}_\epsilon = \{X' : \|X' - X\|_\infty \leq \epsilon\}$: all possible independent perturbations of the parameters by at most ϵ. Then, instead of solving the original problem

$$\text{find } \text{opt}(X) = \max_S \text{obj}(S, X),$$

where $\text{obj}(S, X)$ is the objective of S under parameters X, instead we solve

$$\text{find } \text{opt}_{\text{robust}}(X) = \max_S \min_{X' \in \mathcal{X}_\epsilon} \text{obj}(S, X')$$

The *price of robustness* (Bertsimas and Sim, 2004) is then defined to be:

$$\inf_X \frac{\text{opt}(X)}{\text{opt}_{\text{robust}}(X)}$$

We believe that the robust version of both the Agent's and the Designer's problems can be solved through modifications of the corresponding algorithms. However, even if the robust versions of the Agent's and Designer's can be solved, the quality of the robust solution to the Designer's problem can be arbitrarily bad. We formalize this interesting fact with the following negative result:

Theorem 6. *The price of robustness of the Designer's problem is unbounded.*

Proof. **(Sketch:)** Imagine that there are two states, call them lucrative and dreary, and the Designer has a high revenue at the lucrative state and a low revenue at the dreary state. However, when the platform for the lucrative state is deployed, the Agent's utility coefficient is such that the lucrative state barely makes adoption, and will be rejected if its true coefficient is just a tiny bit smaller; while the platform for the dreary state will be robustly adopted if deployed. The optimum is to deploy the lucrative platform. However, the robust optimum is to deploy the dreary platform, which can be arbitrarily bad in comparison. □

It is not hard to see that this result can be easily generalized to robustness over all parameters, and to robustness under other perturbation metrics, such as ℓ_2. Note that this theorem has ominous implications for the realistic deployment of the optimization apparatus we have developed; we discuss this and many other similar matters in the next section.

6 Discussion and Future Work

We believe that we have barely scratched the surface of a very important subject: the economic/mathematical/algorithmic modeling of the interactions between Designers of on-line platforms and the consumers of on-line services/producers of data. Our model captures a few of the important aspects of this complex environment: the way adoption of services affects both the user's activities and the user's enjoyment of these activities, while it enhances the Designer's revenue in ways that depend on the time spent and activities performed on the platform; the nature of the Designer's profit (revenue from the acquisition of data pertaining to the user minus the significant development costs); the fact that multiple platforms, even by the same Designer, compete for the user's attention and use; the nature of some of the user's dilemmas (chief among them: surrender privacy for increased efficiency and/or enjoyment?). A simplified model of these aspects (the flower chain, linearity of utilities) is a tractable bi-level optimization problem — which, however, already has very unfavorable robustness properties, a fact that suggests that it may not scale well. We believe that intractability (both analytical and computational) lurks in many of the possible immediate generalizations of this model — for example, to undiscretized coefficients, to Markov chains more general than the flower, or to more complex objectives than linear (such as the addition of an entropy regularizer to the objectives of both the Agent and the Designer — an especially tempting variant to consider in this particular problem). We believe that more ambitious problem formulations in these directions may need to further simplify the other aspects of the model in this paper to become tractable.

On the other hand, we also believe that any form of intractability of the Designer's problem is arguably *affordable.* Our dynamic programming FPTAS would likely not generalize to more general contexts — such as those involving complex chains, nonlinear objectives, many Designers, learning of the statistics of the Agents' parameters etc, see below — but the alternative exhaustive algorithm, with its rather benign exponential dependency on n, the number of platforms, is extremely realistic in this context. We believe that the true challenges in generalizing our results are challenges of formulation and modeling.

Superficially, platform design resembles Mechanism Design (MD) (Myerson, 1983), but the essence of much of MD is that the Designer knows only *statistics* of the Agent's characteristics and designs the mechanism to optimize revenue over all possible eventualities by incentivizing the Agent to implicitly reveal their type; and this essence is missing in the PDP. In the on-line platform environment, the subject of incentives for type revelation and truthfulness is rather clearly related to the *personalization* of the platform, and we believe that a generalization of our model will have to address this important issue and aspect of platform design. Related to personalization, *Designer*

competition is another interesting research direction, which seems quite tractable with the right simplifications, and may yield informative results.

In the present first brush at platform design, we have abstracted the PDP in terms of a single Agent — a maneuver and methodology familiar from Economics —, and next ventured to the case of a few Agent types. But of course the motivating environment involves myriads of atypical Agents. The traditional approach in Economics to treat a multitude of Agents is through aggregation of parameters; here, our negative robustness results seem forbidding for this approach. The Agent's statistics must be *learned with high accuracy*, and we believe that this can lead to the development of novel aspects of Machine Learning. The learning nature of platform design resembles interactive learning (the learner and the Agent whose parameters are being learned interact closely, and the learner can easily experiment with variants of the platform), and also has certain characteristics of learning from revealed preferences, see e.g. Zadimoghaddam and Roth (2012). We believe that a wealth of novel and intriguing technical problems within Learning Theory and Machine Learning lie in this direction, and can build on recent work in the intersection of these areas with Algorithmic Mechanism Design and Learning in Games (Haghtalab, 2018; Foster et al., 2016). Of course, recent cautionary results on the limitations of optimization by samples (Balkanski et al., 2017; Balkanski and Singer, 2017) come to mind as well.

Finally — and almost needless to say — the subject of platform design, as circumscribed in this paper, is crying out for treatment from the point of view of the exploding literature on *ethics, fairness, and privacy* in algorithm design — see for example Dwork et al. (2012); Kleinberg et al. (2001); Gemici et al. (2018) among many other important works — and exposes new aspects of today's algorithmic environment to these important considerations and emerging methodologies. The PDP defines an environment where privacy and fairness concerns are ubiquitous and paramount. One example where platform design seems particularly relevant to privacy and fairness is online labor platforms (e.g. Lyft, TaskRabbit), which can disproportionately impact the social welfare of underprivileged users who earn livelihoods on these platforms (Abebe and Goldner, 2018). Understanding what kinds of social, economic, regulatory, and technological interventions may result in fairer outcomes of platform design is an important direction of future work.

7 Acknowledgements

We thank John Tsitsklis and Eva Tardos for helpful conversations during the development of this work. K. Vodrahalli acknowledges support from an NSF Graduate Fellowship.

References

Rediet Abebe and Kira Goldner. 2018. Mechanism Design for Social Good. *AI Matters* 4, 3 (2018), 27–34. https://doi.org/10.1145/3284751.3284761

Anish Agarwal, Munther Dahleh, and Tuhin Sarkar. 2019. A Marketplace for Data: An Algorithmic Solution. In *Proceedings of the 2019 ACM Conference on Economics and Computation (EC '19)*. Association for Computing Machinery, New York, NY, USA, 701–726. https://doi.org/10.1145/3328526.3329589

Eric Balkanski, Aviad Rubinstein, and Yaron Singer. 2017. The Limitations of Optimization from Samples. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*. Association for Computing Machinery, New York, NY, USA, 1016–1027. https://doi.org/10.1145/3055399.3055406

Eric Balkanski and Yaron Singer. 2017. The Sample Complexity of Optimizing a Convex Function. In *Proceedings of the 2017 Conference on Learning Theory (Proceedings of Machine Learning Research)*, Satyen Kale and Ohad Shamir (Eds.), Vol. 65. PMLR, Amsterdam, Netherlands, 275–301. http://proceedings.mlr.press/v65/balkanski17a.html

A. Ben-Tal, L. El Ghaoui, and A.S. Nemirovski. 2009. *Robust Optimization*. Princeton University Press.

Dimitri P. Bertsekas. 2017. *Dynamic Programming and Optimal Control, Vol. I* (4th ed.). Athena Scientific.

Dimitris Bertsimas and Melvyn Sim. 2004. The Price of Robustness. *Operations Research* 52, 1 (2004), 35–53.

Nicolò Cesa-Bianchi and Gabor Lugosi. 2006. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA.

Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through Awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. 214–226.

Dylan J Foster, Zhiyuan Li, Thodoris Lykouris, Karthik Sridharan, and Eva Tardos. 2016. Learning in games: Robustness of fast convergence. In *Advances in Neural Information Processing Systems*. 4734–4742.

Peter Frazier, David Kempe, Jon Kleinberg, and Robert Kleinberg. 2014. Incentivizing exploration. In *Proceedings of the fifteenth ACM conference on Economics and computation*. 5–22.

Kurtuluş Gemici, Elias Koutsoupias, Barnabé Monnot, Christos Papadimitriou, and Georgios Piliouras. 2018. Wealth inequality and the price of anarchy. (2018). arXiv:1802.09269

Nika Haghtalab. 2018. *Foundation of Machine Learning, by the People, for the People*. Ph.D. Dissertation. Carnegie Mellon University. http://reports-archive.adm.cs.cmu.edu/anon/anon/usr0/ftp/usr/ftp/2018/CMU-CS-18-114.pdf

Oscar H. Ibarra and Chul E. Kim. 1975. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems. *J. ACM* 22, 4 (Oct. 1975), 463–468. https://doi.org/10.1145/321906.321909

Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. 1998a. A Microeconomic View of Data Mining. *Data Min. Knowl. Discov.* 2, 4 (Dec. 1998), 311–324. https://doi.org/10.1023/A:1009726428407

Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. 1998b. Segmentation Problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC '98)*. Association for Computing Machinery, New York, NY, USA, 473–482. https://doi.org/10.1145/276698.276860

Jon Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan. 2001. On the Value of Private Information. In *Proceedings of the 8th Conference on Theoretical Aspects of Rationality and Knowledge (TARK '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 249–257.

Yang Liu and Chien-Ju Ho. 2018. Incentivizing High Quality User Contributions: New Arm Generation in Bandit Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 1146–1153. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16879

Thodoris Lykouris, Eva Tardos, and Drishti Wali. 2019. Feedback graph regret bounds for Thompson Sampling and UCB. (2019). arXiv:1905.09898

Yishay Mansour, Aleksandrs Slivkins, and Vasilis Syrgkanis. 2015. Bayesian Incentive-Compatible Bandit Exploration. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation (EC '15)*. Association for Computing Machinery, New York, NY, USA, 565–582. https://doi.org/10.1145/2764468.2764508

Roger B Myerson. 1983. Mechanism design by an informed principal. *Econometrica: Journal of the Econometric Society* (1983), 1767–1797.

Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc., USA.

Tim Roughgarden and Joshua R. Wang. 2016. Minimizing Regret with Multiple Reserves. In *Proceedings of the 2016 ACM Conference on Economics and Computation (EC '16)*. Association for Computing Machinery, New York, NY, USA, 601–616. https://doi.org/10.1145/2940716.2940792

John N. Tsitsiklis and Kuang Xu. 2018. Delay-Predictability Trade-offs in Reaching a Secret Goal. *Operations Research* 66, 2 (2018), 587–596. https://doi.org/10.1287/opre.2017.1682

John N. Tsitsiklis, Kuang Xu, and Zhi Xu. 2018. Private Sequential Learning. (2018). arXiv:1805.02136

Morteza Zadimoghaddam and Aaron Roth. 2012. Efficiently Learning from Revealed Preference. In *Internet and Network Economics*, Paul W. Goldberg (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 114–127.

A Proof of Lemma 1

Proof of Lemma 1. First, define the Markov chain transition matrix as the composition of an optimal policy ρ^* and T:

$$M(i,j) = \sum_{a \in \{a^0, a^1\}} \rho^*(i,a) T(i,a,j)$$

By the ergodic theorem for irreducible finite-state Markov chains, it is well known that we can express the total reward of an irreducible average-reward MDP as

$$R(\rho^*) = \langle \pi(\rho^*), r(\rho^*) \rangle$$

where $\pi(\rho^*)$ is the stationary distribution that results from playing policy ρ^* and reward vector $r \in \mathbb{R}^{n+1}$ is fixed for all time, since ρ^* is fixed and the rewards only depend on state and action values (see e.g. Puterman (1994); Bertsekas (2017)). Let us now calculate π and r. By solving the

balance equations for M, elementary algebra shows that the stationary distribution is as follows: Define

$$x_i(\rho) := \begin{cases} \frac{p_i}{1-q_i} & \text{if } \rho(i, a^0) = 1 \\ \frac{p_i}{1-q_i-y_i} & \text{if } \rho(i, a^1) = 1 \\ 1 & \text{if } i = 0 \end{cases}$$

Then, the stationary distribution is given by

$$\pi_i(\rho) := \frac{x_i(\rho)}{\sum_{j=0}^{n} x_j(\rho)}$$

We also note that

$$r_i(\rho) := \begin{cases} c_i^{\text{life}} & \text{if } \rho(i, a^0) = 1 \\ c_i^{\text{platform}} & \text{if } \rho(i, a^1) = 1 \\ 0 & \text{if } i = 0 \end{cases}$$

Translating these values into our objective $R(\rho) = \langle \pi, r \rangle$, we want to solve

$$\max_{\rho \in \{0,1\}^n} \sum_{i=1}^{n} \pi_i(\rho) \cdot r_i(\rho) := \max_{\rho \in \{0,1\}^n} \sum_{i=1}^{n} \frac{x_i(\rho)}{\sum_{j=0}^{n} x_j(\rho)} \cdot r_i(\rho) = \max_{\rho \in \{0,1\}^n} \frac{\sum_{i=1}^{n} x_i(\rho) \cdot r_i(\rho)}{\sum_{j=0}^{n} x_j(\rho)}$$

which we note is exactly the objective stated in the theorem. □

B Proof of Theorem 3

In this section, we give the proof that the PDP is NP-complete.
Theorem 3. The PDP is NP-complete.

Proof. We reduce from the Partition problem: Given a set of positive integers $a_1, \ldots, a_n$, is there a partition of the numbers into two subsets that have equal sums? We first apply the following (standard) transformation which yields an instance of the partition problem where the numbers are comparable in value and any solution must also have equal number of elements in each part. Let $H = n \sum_i a_i$. Construct a new instance of the partition problem with $2n$ elements $b_1, \ldots b_{2n}$; the first n elements are $b_1 = H + a_1, \ldots, b_n = H + a_n$, and the other n elements $b_j, j = n+1, \ldots, 2n$ are all H. It is easy to see that the original instance has a solution iff the new instance does, and furthermore, any solution of the new instance must have n elements in each part.

We create now an instance of the PDP problem. The flower has $2n + 1$ petals. The first $2n$ petals correspond to the $2n$ numbers b_i, and the last petal is the *special* petal. We set the parameters as follows. Set $c_i^{\text{life}} = 0$ for all i. Set $p_i = n^2(1 - q_i)$ and $y_i = \frac{1}{n^2+1}(1 - q_i)$ for all i. Then $\lambda_i = \frac{p_i}{1-q_i} = n^2$, $w_i = \frac{p_i}{1-q_i-y_i} = n^2 + 1$, and $z_i = \frac{p_i}{1-q_i-y_i} - \frac{p_i}{1-q_i} = 1$. Therefore, $A = 0$ and $B = 1 + \sum_i \lambda_i = 1 + n^2(2n + 1)$.

We choose the platform coefficient for the special petal $s = 2n + 1$ so that its potential $\phi(s) = (\sum_i b_i)/2B = (2nH + \sum_i a_i)/2B$. Specifically, set $c_s^{\text{platform}} = (\sum_i b_i)/2B(n^2 + 1)$. For the non-special petals $i \in [2n]$, we choose their platform coefficients so that their potentials satisfy $\phi(i) = \phi(s) + b_i$. For this, set $c_i^{\text{platform}} = ((\sum_i b_i)/2B + b_i)/(n^2 + 1)$.

Set $\text{cost}_i = 0$ for all i. Set $d_i = b_i$ for all $i \in [2n]$, and for the special petal $s = 2n + 1$, we set $d_s = 4nH$. This concludes the specification of the instance of PDP.

We shall show that the given instance of the Partition problem has a solution if and only if the optimal profit is $v^* = (n^2+1)(4nH + \sum_{i=1}^{2n} b_i/2)/(B+n+1)$. In this case, an optimal solution of the PDP instance consists of the special petal and a solution of the partition instance.

First, suppose that the partition instance has a solution $S \subset [2n]$. Consider the solution $S \cup s$ of the PDP instance. We claim that it is feasible. The smallest potential is that of the special petal, $\phi(s)$. The utility of the agent for $S \cup \{s\}$ is $\frac{A+\sum_{i\in S\cup\{s\}} z_i\phi(i)}{B+\sum_{i\in S\cup\{s\}} z_i}$ which is equal to $\phi(s)$, since $\phi(i) = \phi(s) + b_i$ for all $i \in [2n]$ and $\sum_{i\in S} b_i = \sum_{i=1}^{2n} b_i/2$.[5] An easy calculation also shows that the profit of the solution $S \cup \{s\}$ is $(n^2+1)(4nH + (\sum_{i=1}^{2n} b_i/2))/(B+n+1) = v^*$.

Conversely, suppose that the PDP instance has a solution S^* with profit at least $v^* = (n^2+1)(4nH + \sum_{i=1}^{2n} b_i/2)/(B+n+1)$. Then it must contain the special petal s, because even if we take all the other petals, the profit is smaller. Let $S = S^* - \{s\}$. The profit of the solution $S^* = S \cup \{s\}$ is $(n^2+1)(4nH + \sum_{i\in S} b_i)/(B+|S|+1)$. If $|S| < n$, then $\sum_{i\in S} b_i \le (n-1)H + \sum_i a_i$ and the profit is less than v^*. Therefore $|S| \ge n$ and $\sum_{i\in S} b_i \ge \sum_{i=1}^{2n} b_i/2$. Since S^* is feasible, we must have $\phi(s) \ge \frac{A+\sum_{i\in S} z_i\phi(i)}{B+\sum_{i\in S} z_i}$. Substituting the values of the parameters, this inequality yields, $\sum_{i\in S} b_i \le \sum_{i=1}^{2n} b_i/2$. Therefore, for the profit to be v^*, we must have $|S| = n$ and $\sum_{i\in S} b_i = \sum_{i=1}^{2n} b_i/2$. Thus, the partition instance has a solution. □

C Proof of Theorem 5

Theorem 5. Unless P=NP, there is no FPTAS for the Designer's problem with two agents if the $\phi_i(j)$ are not restricted to be polynomially bounded.

Proof. We reduce from the Partition problem: Given a set of positive integers $a_1, \ldots, a_n$, is there a partition of the numbers into two subsets that have equal sums? As in the proof of Theorem 3, we first transform it to an instance of the partition problem with $2n$ numbers $b_1 = H + a_1, \ldots, b_n = H + a_n$, and $b_j = H$ for $j = n+1, \ldots, 2n$, where $H = n\sum_i a_i$. The original instance has a solution iff the new instance does, and furthermore, any solution of the new instance must have n elements in each part.

We have 2 agents. The agents have a flower Markov chain with $2n+1$ petals, where the first $2n$ petals correspond to the $2n$ numbers b_i, and the last petal is the *special* petal. The parameters of the Markov chain for both agents are the same as in the proof of Theorem 3. That is, for $i = 1, 2$ we set $c_{ij}^{\text{life}} = 0$ for all j; $p_{ij} = n^2(1-q_{ij})$ and $y_{ij} = \frac{1}{n^2+1}(1-q_{ij})$ for all j. Then $\lambda_{ij} = \frac{p_{ij}}{1-q_{ij}} = n^2$, $w_{ij} = \frac{p_{ij}}{1-q_{ij}-y_{ij}} = n^2+1$, and $z_{ij} = \frac{p_{ij}}{1-q_{ij}-y_{ij}} - \frac{p_{ij}}{1-q_{ij}} = 1$. Therefore, $A_1 = A_2 = 0$ and $B_1 = B_2 = B = 1 + \sum_j \lambda_{ij} = 1 + n^2(2n+1)$.

The two agents differ in the rewards when they adopt the platform in a state. Agent 1 has the same rewards as in the proof of Theorem 3. Agent 2 has rewards that are defined in a similar way from the numbers $b'_j = 2H - b_j$ for all $j \in [2n]$. Thus, for the special petal $s = 2n+1$, we set $c_{1s}^{\text{platform}} = (\sum_i b_i)/2B(n^2+1)$ and $c_{2s}^{\text{platform}} = (\sum_i b'_i)/2B(n^2+1)$. Therefore, its potential for the two agents is $\phi_1(s) = (\sum_i b_i)/2B$ and $\phi_2(s) = (\sum_i b'_i)/2B$. For the non-special petals $i \in [2n]$, we choose their platform coefficients so that their potentials satisfy $\phi_1(j) = \phi_1(s) + b_i$ and $\phi_2(j) = \phi_2(s) + b'_i$. For this, set $c_{1j}^{\text{platform}} = ((\sum_i b_i)/2B + b_j)/(n^2+1)$, and $c_{2i}^{\text{platform}} = ((\sum_i b'_i)/2B + b'_j)/(n^2+1)$. Set

[5] We assumed here for simplicity that the Agent's greedy algorithm includes a state in case of equality between the potential and the utility; recall that from the Agent's perspective this does not make any difference. If the state is not included in case of equality, then we have to adjust slightly the parameters to increase slightly the potential $\phi(s)$.

$\text{cost}_j = 0$ for all j. Set $d_{1j} = d_{2j} = 1$ for all $j \in [2n]$, and for the special petal $s = 2n+1$, we set $d_{1s} = d_{2s} = 3n$. This concludes the specification of the instance of PDP.

We shall show that if the given instance of the Partition problem has a solution then the optimal profit is $v^* = 8n \cdot \frac{n^2+1}{B+n+1}$, whereas if it does not have a solution then the optimal profit is at most $(8n-2) \cdot \frac{n^2+1}{B+n} < (1 - \frac{1}{8n})v^*$.

First, suppose that the partition instance has a solution $S \subset [2n]$. Consider the solution $S \cup \{s\}$ of the PDP instance. It is easy to check that both agents will adopt all the states in $S \cup \{s\}$, as in the proof of Theorem 3. The profit of the solution $S \cup \{s\}$ is $8n \cdot \frac{n^2+1}{B+n+1}$.

Conversely, suppose that the PDP instance has a solution S^* with profit greater than $(8n-2) \cdot \frac{n^2+1}{B+n}$. Then S^* must contain the special petal s, and s must be selected by both agents, because otherwise, even if they take all the other petals, the profit is no more than $7n \cdot \frac{n^2+1}{B+n+1}$. Since s has lowest potential among all the petals for both agents, it follows that both agents select all the states in S^*.

Let $S = S^* - \{s\}$. If $|S| \geq n+1$ then Agent 1 will not select the special state because $\phi_1(s) < \frac{A_1 + \sum_{j \in S} z_{1j}\phi_1(j)}{B_1 + \sum_{j \in S} z_{1j}} = \frac{\sum_{j \in S}(b_j + \phi_1(s))}{B_1 + |S|}$ since $\sum_{j \in S} b_j \geq |S| \cdot H \geq (n+1)H$, and $B_1\phi_1(s) = (\sum_{i \in [2n]} b_i)/2 < (n+1)H$. Therefore, $|S| \leq n$. On the other hand, if $|S| \leq n-1$ then $\text{profit}(S^*) \leq 2(4n-1) \cdot \frac{n^2+1}{B+n}$. Therefore, $|S| = n$.

Since both agents select the special petal, we have from agent 1: $\phi_1(s) \geq \frac{A_1 + \sum_{j \in S} z_{1j}\phi_1(j)}{B_1 + \sum_{j \in S} z_{1j}} = \frac{\sum_{j \in S} b_j + n\phi_1(s)}{B+n}$, therefore, $\phi_1(s) \geq \frac{\sum_{j \in S} b_j}{B_1}$. Since $\phi_1(s) = \frac{\sum_{i \in [2n]} b_i}{2B_1}$, we get $\sum_{j \in S} b_j \leq \frac{\sum_{i \in [2n]} b_i}{2}$. Similarly, we get from agent 2: $\sum_{j \in S} b'_j \leq \frac{\sum_{i \in [2n]} b'_i}{2}$. Since $b'_j = 2H - b_j$ and $|S| = n$, this implies that $\sum_{j \in S} b_j \geq \frac{\sum_{i \in [2n]} b_i}{2}$. Therefore, $\sum_{j \in S} b_j = \frac{\sum_{i \in [2n]} b_i}{2}$, and S is a solution to the Partition problem. □

D The Greedy Algorithm when the y_i can be Negative

In the case where y_i is allowed to be negative, we need to slightly modify Algorithm 1 as well as its proof. The new algorithm is as shown below.

Lemma 7. *Suppose $s < 0$ and $y + s > 0$. Then*

$$\frac{x}{y} < \frac{r}{s} \iff \frac{r}{s} > \frac{x}{y} > \frac{x+r}{y+s}$$

and

$$\frac{x}{y} > \frac{r}{s} \iff \frac{r}{s} < \frac{x}{y} < \frac{x+r}{y+s}$$

Proof. For the first inequality, we have that since $s < 0, y > 0$ the LHS implies

$$xs > ry \iff xy + xs > xy + ry$$

which is $\iff$ the RHS. The second inequality follows from the same argument reversed. □

Theorem 7. *Algorithm 4 returns an optimal policy when there exist $y_i < 0$.*

Proof. We first sketch the main idea: adding more new states from P is possible when the utility is small, and adding more new states from N is possible when the utility is large, using Lemmas 2 and 7. Thus, to maximize utility, first maximize the utility over P (allowing the utility to be as

ALGORITHM 4: General Greedy Algorithm

Input: Parameters of the Agent's problem: transition probabilities and utility coefficients in and out of the platform.

Output: An optimal subset $S \subseteq [n]$ of states where the Agent accepts the platform.

Initialize $S := \{\}$
Divide $[n]$ into two lists $P = \{i : z_i > 0\}; N = \{j : z_j < 0\}$
Sort P in order of $\phi(k)$ from largest to smallest
Sort N in order of $\phi(k)$ from smallest to largest
for $k \in P$ **do**
  if $\text{utility}^{\text{Agent}}(S) < \phi(k)$ **then**
    Update $S := S \cup \{k\}$
  else
    Break
  end
end
for $k \in N$ **do**
  if $\text{utility}^{\text{Agent}}(S) > \phi(k)$ **then**
    Update $S := S \cup \{k\}$
  else
    Break
  end
end
return S

large as possible for adding states from N). Adding the additional states from N does not mean there are additional states from P to add, because now the utility is larger than when we stopped adding from P, and the contiguity arguments from Theorem 1 imply we are done.

Now we elaborate on the details. By Theorem 1, after the first for loop in Algorithm 4, we have an optimal policy over the states in P. For the states in N, first note that we can apply Lemma 7 as follows: Choose $x = A + \sum_{j \in S} z_j \phi(j)$, $y = B + \sum_{j \in S} z_j$, $r = \phi(k) z_k$, $s = z_k$. We will always have $z_k < 0$ for $k \in N$. We will also always have $y + z_k > 0$ for $k \in N$: We only need to show that $y > |z_k|$. Writing out the expressions from Lemma 1, we have for any set $T \subseteq [n]$

$$1 + \sum_{i \in [n]} \lambda_i - \sum_{j \in T} |z_j| = 1 + \sum_{i \in [n]} \frac{p_i}{1 - q_i} - \sum_{j \in T} \frac{p_j}{1 - q_j} + \sum_{j \in T} \frac{p_i}{1 - q_i - y_i}$$

$$= 1 + \sum_{i \notin T} \frac{p_i}{1 - q_i} + \sum_{j \in T} \frac{p_j}{1 - q_j - y_j} > 0$$

Thus, the case when $y_k < 0$ (and thus $z_k < 0$) satisfies the conditions of Lemma 7.

Now we prove optimality. First note that once a subset from P is fixed, the optimal additional subset of states from N can be determined greedily by adding the states with the smallest viable potential $\phi(k)$ first. By Lemma 7, if $x/y > \phi(k)$, adding state k increases the utility to $x'/y' > x/y$. Since the utility has increased, we are able to add all states with $\phi(k) < x/y$ and increase the utility – not adding any of these states to increase the utility when we can results in sub-optimality (note that Algorithm 4 indeed ensures all of these states will be added, since we start from the state with the smallest potential). This fact establishes a similar contiguity property for adding states from

N (see the proof of Theorem 1). Thus, once the greedy algorithm stops, we have an optimal policy with respect to any given fixed subset of states from P.

To conclude the proof, we show that any policy not containing an optimal policy over P is sub-optimal. This fact directly implies that Algorithm 4 results in an optimal policy, since it first chooses an optimal subset of P, and then adds from N greedily in an optimal manner, as described above. Consider that by Lemma 7, if we choose a sub-optimal subset of P, x/y will be smaller than if we chose an optimal subset of P. Since one can add more states (with larger potential) from N the larger the initial x/y, and since to be optimal, one must add *all* states with potential less than x/y, the overall utility gained by adding states from N is at most the utility of the optimal subset of P plus the optimally added states from N. Thus the policy is sub-optimal unless we choose an optimal subset of P from the start. □

Note now that Theorems 2, 3, 4, 5, 6 all go through identically in the case where there exist states i such that $y_i < 0$. Changing the Agent's algorithm from Algorithm 1 to Algorithm 4 does not affect the proofs of any of the above theorems. For the FPTAS results, the Designer's algorithm only uses the Agent's algorithm to check feasibility in a black-box fashion, and the sign of z_k in the objective does not affect the analysis of the theorems. Similarly, the robustness result also does not depend on the algorithm or the sign of z_k. Finally, the hardness results are unaffected since the constructed hard examples can just choose a case where all $y_i > 0$.