The Platform Design Problem

Christos Papadimitriou, **Kiran Vodrahalli**, Mihalis Yannakakis

Columbia University

Google Learning Theory Reading Group

October 11, 2021

The Data-Collection Problem

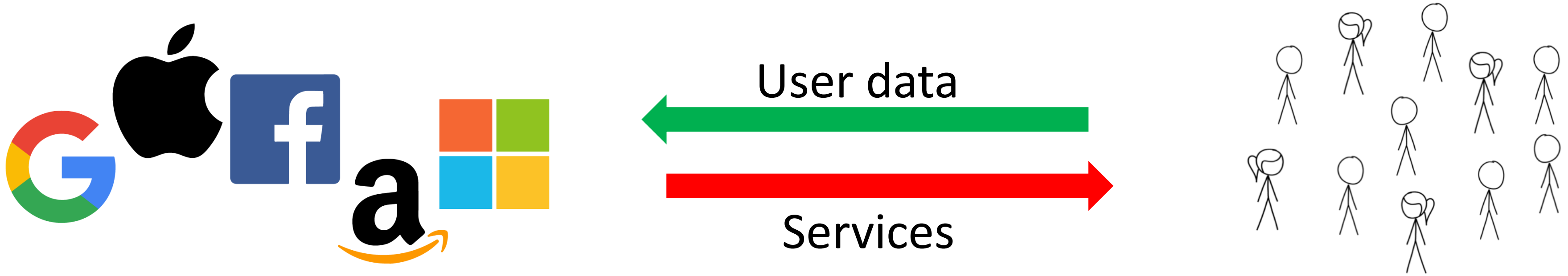
- Modern machine learning requires large amounts of high-quality data
- Collecting supervised labels is expensive
- Unsupervised learning is challenging to use
- Is it possible to create environments which generate useful data?
    - Ex: Reddit users provide sarcasm labels using the "/s" tag

The Data-Collection Problem

- Modern machine learning requires large amounts of high-quality data
- Collecting supervised labels is expensive
- Unsupervised learning is challenging to use
- Is it possible to create environments which generate useful data?
  - Ex: Reddit users provide sarcasm labels using the “/s” tag

Modern tech companies try to solve this problem.

Economics of the Online Firm



Online firm

Users

- User data feeds revenue
  - Better demand segmentation
  - Ad/recommendation revenue
  - Better models => better services
- Online services bring value
  - Convenience
  - Knowledge

Outline

- Problem Definition
- General Case
- Tractable “Flower” Case
  - Agent Behavior
  - Designer’s Algorithm
- Extensions
- Summary
- Future Work

Problem Definition

Platform Design

Problem

Model the revenue-maximization problem of today's online firms (e.g. Google, FB, etc.) and understand computational tractability.

Bi-Level MDP Optimization Model

Agent: participates in Life MDP

Designer: tweaks the Life MDP by building platforms.

Goal: Designer wants to indirectly optimize its reward via Agent's optimal behavior! (Find Stackelberg)

- Key Idea: Google builds various apps (Maps, Search, Social Network, etc.) and profits based on usage of these apps.
- The usage of apps modifies the transitions of the Markov Chain of the user's life
- Assume the Designer has linear rewards over the steady state distribution of the resulting Markov chain (agent policy + Life MDP)

The Stackelberg Game

- Designer moves first:
  - Adds platforms which, if adopted, modify transitions to an existing Markov Chain
- Agent moves second:
  - Receives MDP from Designer, plays optimal behavior
- Example of bi-level MDP optimization
- What is the computational complexity of solving for equilibrium?

Formal Problem Statement

- An **agent** lives in an irreducible Markov chain with $A = [n]$ states.
- The **designer** chooses $S \subseteq A$ states to add platforms to.
- The agent may **adopt or not adopt** the platform at each state:
  - If **adopt**, the transitions change. Otherwise they do not.
  - Assume the chain remains irreducible.

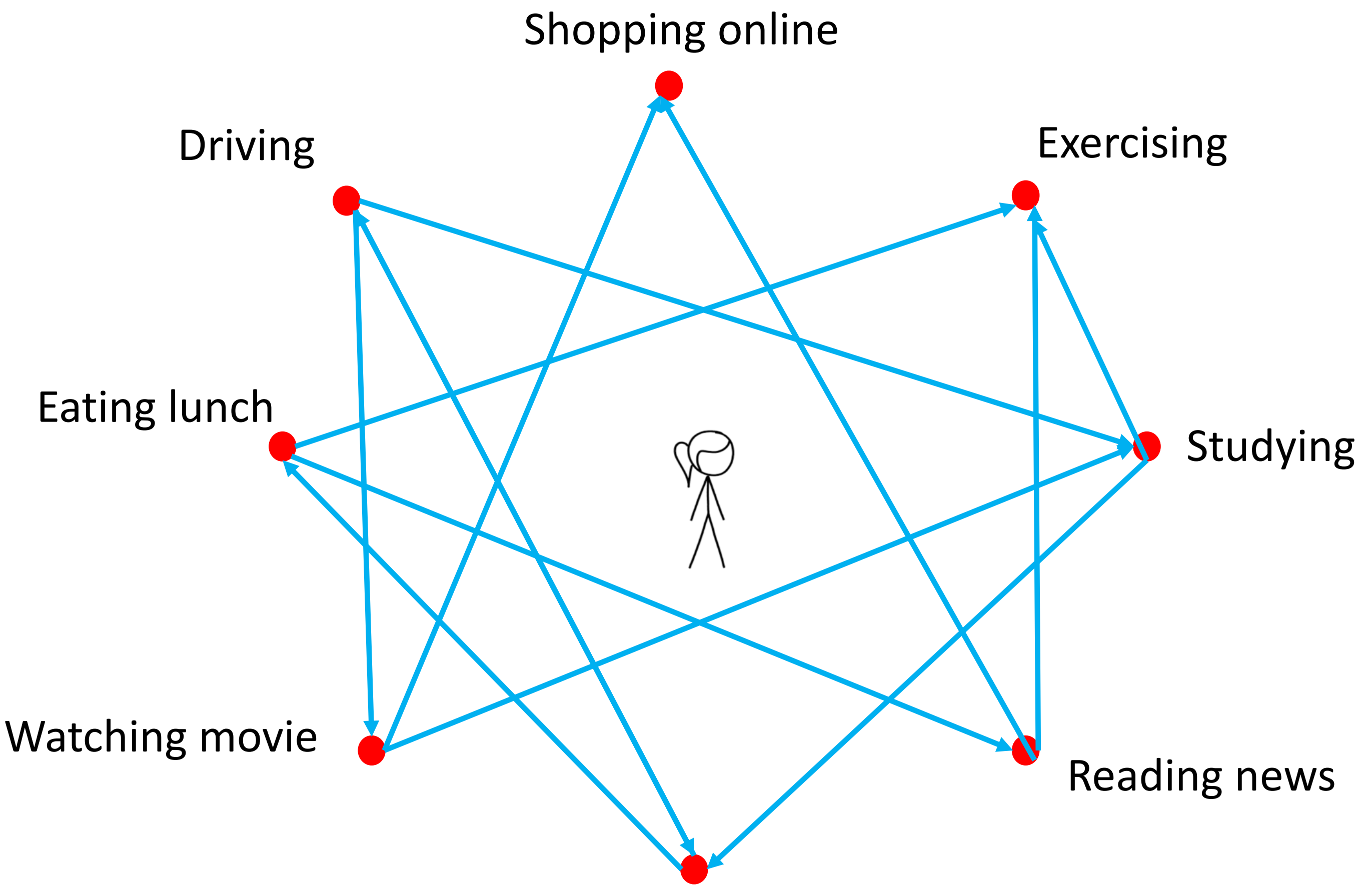
Formal Problem Statement

- Assign a utility rate for the agent (c_i) and the designer (d_i) at $i \in [n]$.
- The agent solves the resulting Markov Decision Process.
  - Resulting steady-state probabilities are given by π.
- The designer optimizes over S:

$$\mathrm{profit}(S) := \sum_{i \in S} d_i \cdot \pi_i(S) - \sum_{i \in S} \mathrm{cost}_i$$

General Case

Picture of the General Case

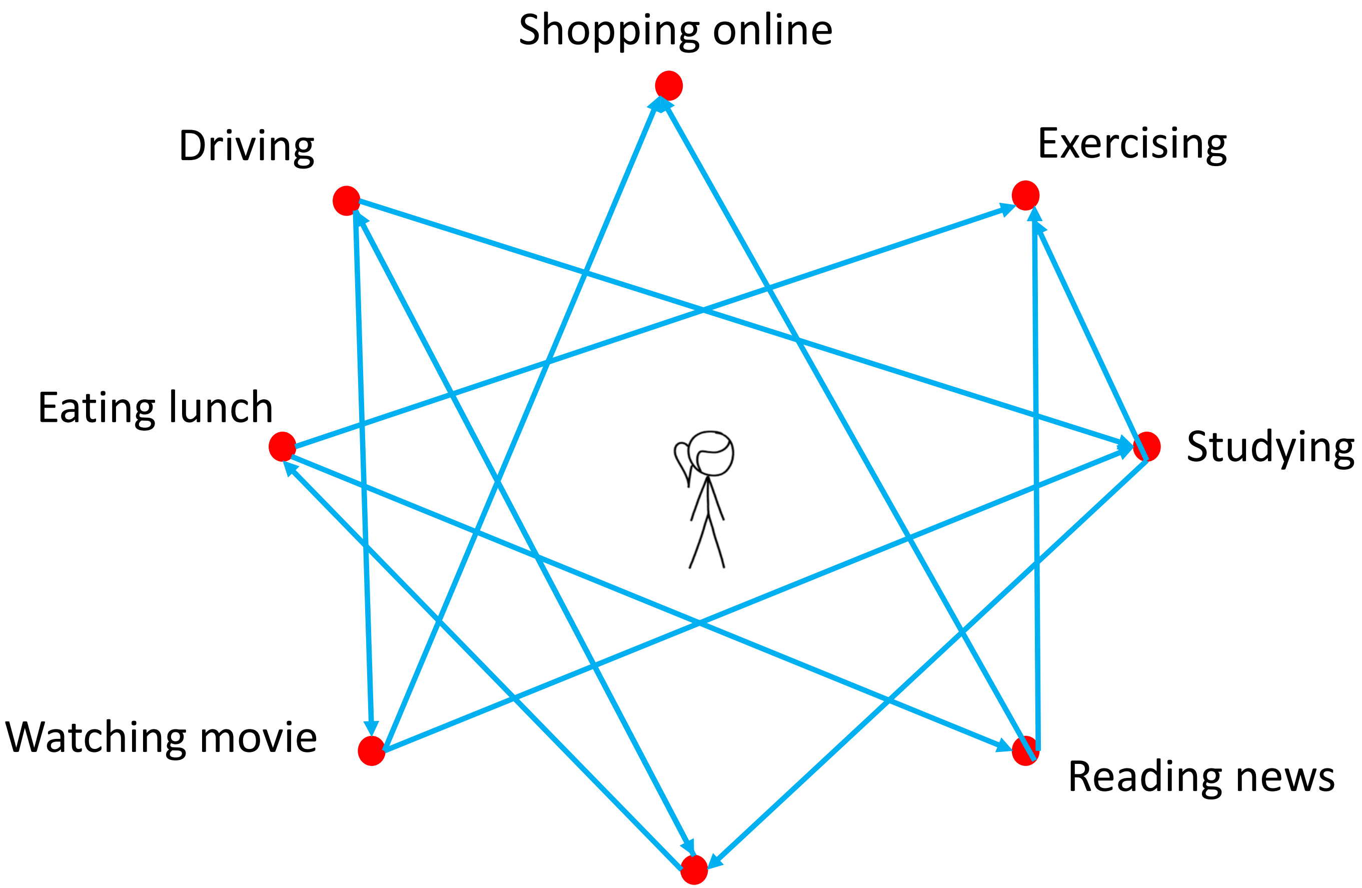


Agent’s Life

What platforms should I build?

Online firm

Picture of the General Case



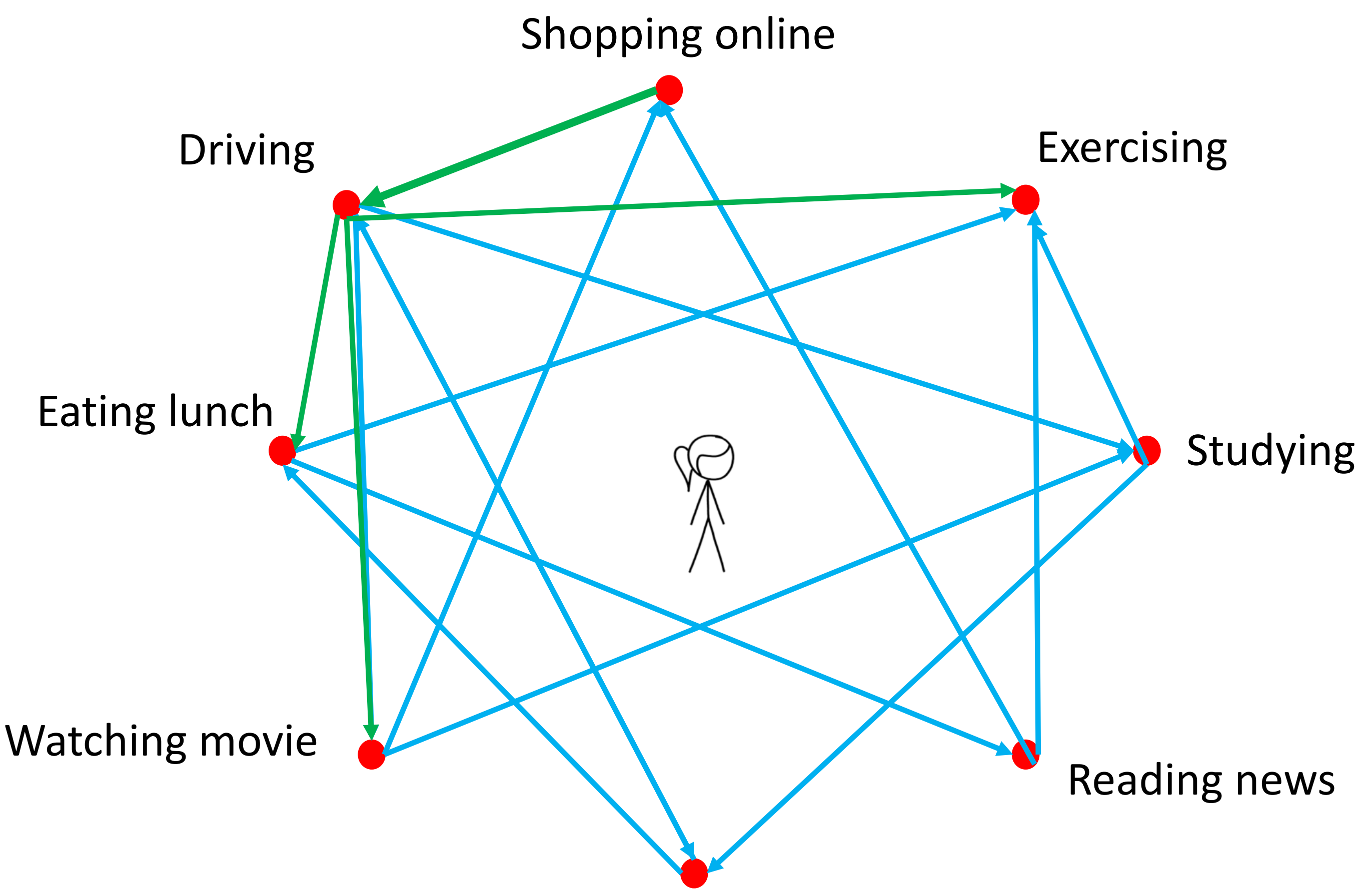
Agent's Life

What platforms should I build?

Online firm

At a cost, the firm can **add an opt-in action** to platforms they create (ex: Google Maps).

Picture of the General Case



Agent's Life
changes

Maybe we should create Maps technology....

Online firm

Builds platform Maps at a cost.

Opt in to Maps

Computational Tractability I: General Case

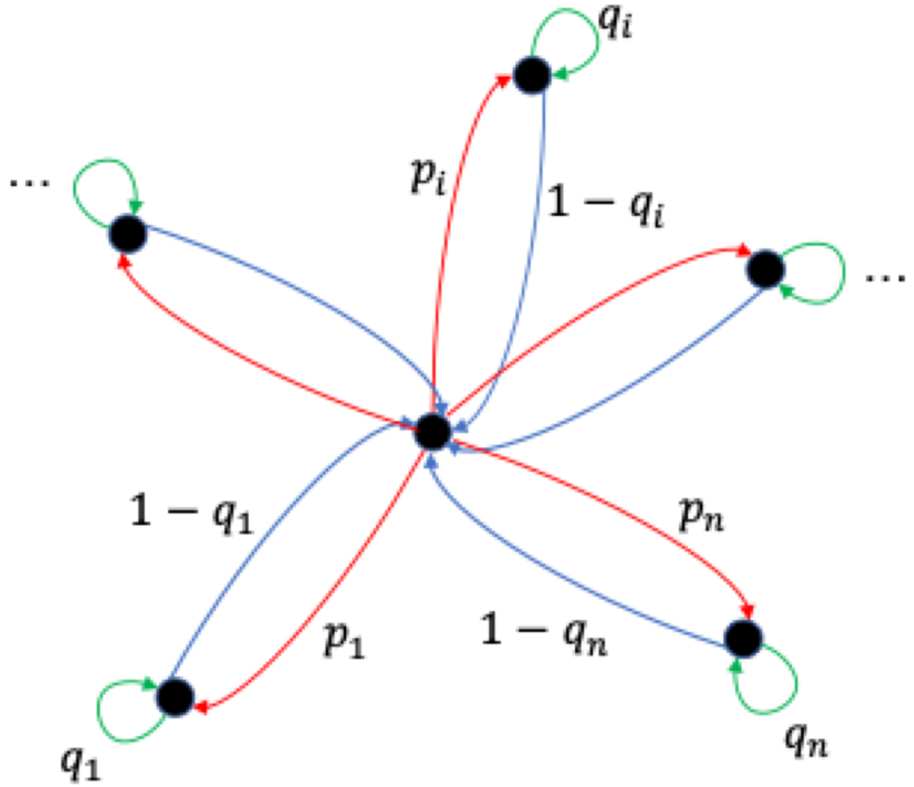
- It is strongly NP-hard to decide whether the Designer can obtain positive profit – and therefore hard to approximate.

- Reduction from SET COVER
  - Designer builds platforms which each solve subset of Agent's problems.
  - Most cost-effective covering set is NP hard.

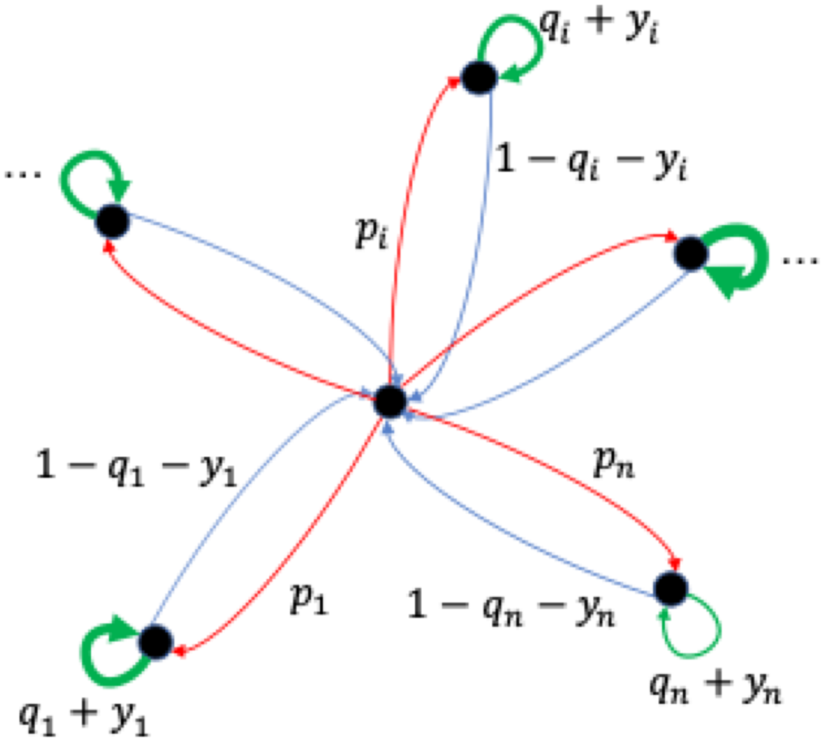
- In economic terms, the reduction exploits the complexity of "complementary goods."
  - Ex: Brick-and-mortar retail ads help the Agent discover the store, Maps helps the Agent get to the store.

Tractable “Flower” Case

A More Tractable Case: The Flower



Life MDP



Tweaked MDP via y_i

A More Tractable Case: The Flower

- Problem can be solved by an FPTAS

- Why tractable?
  - Substitutes rather than complements
    - Allocate time spent in each platform
  - Simpler low-level behavior (greedy agent)
  - Admits a DP upon discretization (knapsack DP)

Agent Behavior

The Agent's Greedy Algorithm

- Solving for the steady state distribution yields a quasi-concave combinatorial optimization problem:

Lemma 1. *The agent's objective for an optimal policy defined in Section 2 can be re-written as the following optimization in the special case of the flower MDP (Definition 2):*

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

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

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

The Agent's Greedy Algorithm

- Solving for the steady state distribution yields quasi-concave combinatorial optimization problem:

Lemma 1. *The agent's objective for an optimal policy defined in Section 2 can be re-written as the following optimization in the special case of the flower MDP (Definition 2):*

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

where

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

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

Potential function

We therefore define

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

The Agent's Greedy Algorithm

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*[9] *from largest to smallest* $\phi(k)$ **do**

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

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

    else

        return S

    end

end

return S

Sort states by potential function and add until utility = potential

Proof Outline

- Application of the mediant inequality: $\frac{x}{y} < \frac{r}{s} \iff \frac{x}{y} < \frac{x+r}{y+s} < \frac{r}{s} \quad \text{where } y, s > 0.$

- Optimal policy must be "prefix": must contain first m states, sorting by potential function ϕ.

- Proof and algorithm slightly more complex when $z_i < 0$.

- ϕ is the platform reward + scaled gain over regular life.

- Agent accepts when
  - Base platform reward is high
  - Amount of time spent on platform is high and platform is better

Designer’s Algorithm

Recall: Designer's Objective

$$\mathrm{profit}(S) := \sum_{i \in S} d_i \cdot \pi_i(S) - \sum_{i \in S} \mathrm{cost}_i$$

Set of states to build platforms

Designer's steady-state reward rates

Agent's steady state probabilities

Designer's one-time costs for building each platform

Restrictions

- Expanding the profit function given agent behavior:

$$\text{profit}(S) := \underbrace{\frac{\sum_{i \in \text{Agent}(S)} d_i \cdot \frac{p_i}{1 - q_i - y_i}}{\underbrace{B + \sum_{i \in \text{Agent}(S)} z_i}_{D(S)}}}_{P_1(S)} - \sum_{i \in S} \text{cost}_i$$

- Define $\max_i d_i =: K$
  - Maximum profit is nK
- Assume z_i are $poly(n)$ and discretized with gap δ and costs are $K * poly(n)$

Target Algorithm

- Deciding whether it is possible to attain a certain profit is NP complete
- Reduction from PARTITION
- Thus, our goal: A $(1 - \epsilon)$ approximate algorithm in polynomial time.

The Designer’s Dynamic Program

- **Key Idea:** Use a (poly-sized) hash table with rounded rewards
- Difficulty comes from profit scale and non-discretized z_i
- Hash function:

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

- Similar to standard Knapsack FPTAS (Ibarra & Kim, 1975)

The Designer's Dynamic Program

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

efficient feasibility check

update if numerator of agent reward is smaller than current hash

new hash

Proof Outline: Key Lemma

- For S, S' that hash to same bin; if $N(S) \leq N(S')$, for any postfix set T:
  - If $S' \cup T$ is feasible, so is $S \cup T$
  - $S \cup T$ is at most ${}^{\epsilon K}/_{n}$ worse than $S' \cup T$

Proof:

- Feasibility: Shared denominator + sub-optimality of $S \cup T$
- Suboptimality: |profit($S \cup T$) - profit($S' \cup T$)| $\leq$ |profit(S) - profit(S')| $+$ $|P_1(S) - P_1(S')|$ and shared hash bin.

Proof Outline: Applying Key Lemma

- After i^{th} iteration:
  - hash table contains S extendable to $S \cup T$ at most ${}^{\epsilon i K}/_{n}$ suboptimal.
- After n iterations, suboptimal by at most $(1 - \epsilon)$ factor.

Proof: By induction.

- Inductive step: apply the Key Lemma at step i to get:
  - If S is replaced this step: ${}^{\epsilon(i-1)K}/_{n} + {}^{\epsilon K}/_{n} = {}^{\epsilon i K}/_{n}$ suboptimal
  - Otherwise: even tighter bound, no degradation.

Proof Outline: Complexity

- Checking feasibility is $O(n \log n)$
- Three dimensions of hash table:
  - Denominator dimension is size n/δ
  - Profit dimensions are each size $\frac{nK}{\epsilon K/_{n}} = \frac{n^2}{\epsilon}$ times a possible polynomial factor for costs

- Total size of hash table: $O(poly(n) \cdot {n^5}/_{\epsilon^2 \delta})$

- Thus: polynomial runtime.

Extensions

Multiple Agents

- Replace designer objective with summation over agents:

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

- An exact polytime DP exists if #agents is constant.
    - Exponential in #agents
    - Also require potentials ϕ_i to be discretized by δ' with poly size.

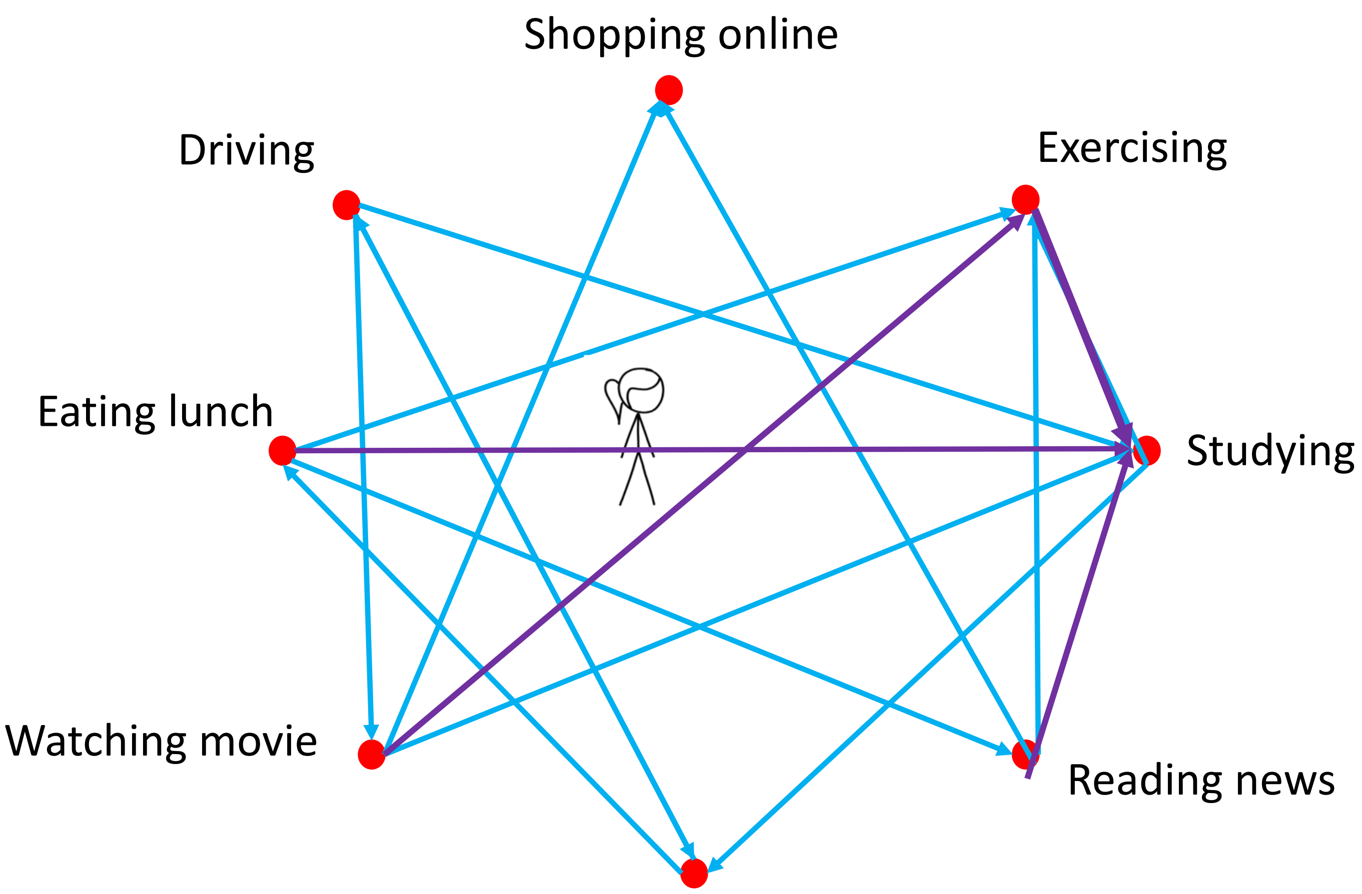
- No FPTAS for 2 agents if ϕ_i not polynomial size.

Multiple Agents

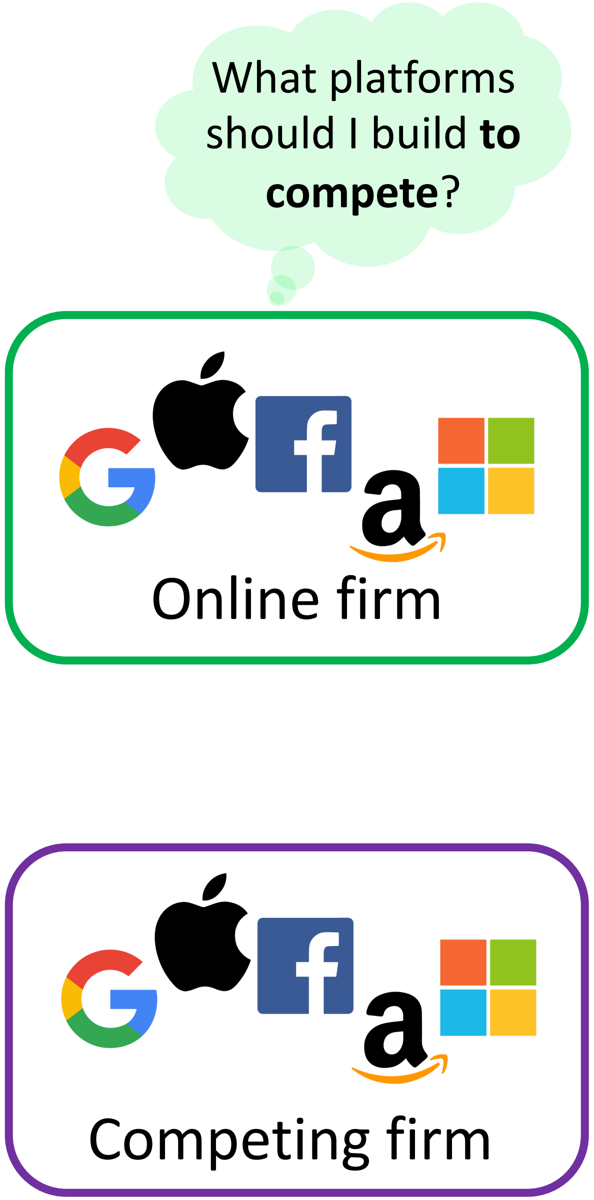
- **Key ideas:**
  - Discretize over potentials and denominators.
  - For each potential-denominator pair (θ, D), compute optimal subset s.t. potentials are at least θ with a DP hash table (size M^{3k}).
  - Enumerate over (θ, D) to get global optimum.

- **Proof sketch:** Key idea is that we can exactly compute values for each entry in the (θ, D) hash table.

Designer Competition



Agent's Life



Multiple Platforms (Flower Setting)

- What if other competing designers have already built platforms?
    - Each platform affects only one state
    - At most one for each designer per state
- How does an agent behave?
- How should a designer optimally place platforms?

Multiple Platforms (Flower Setting)

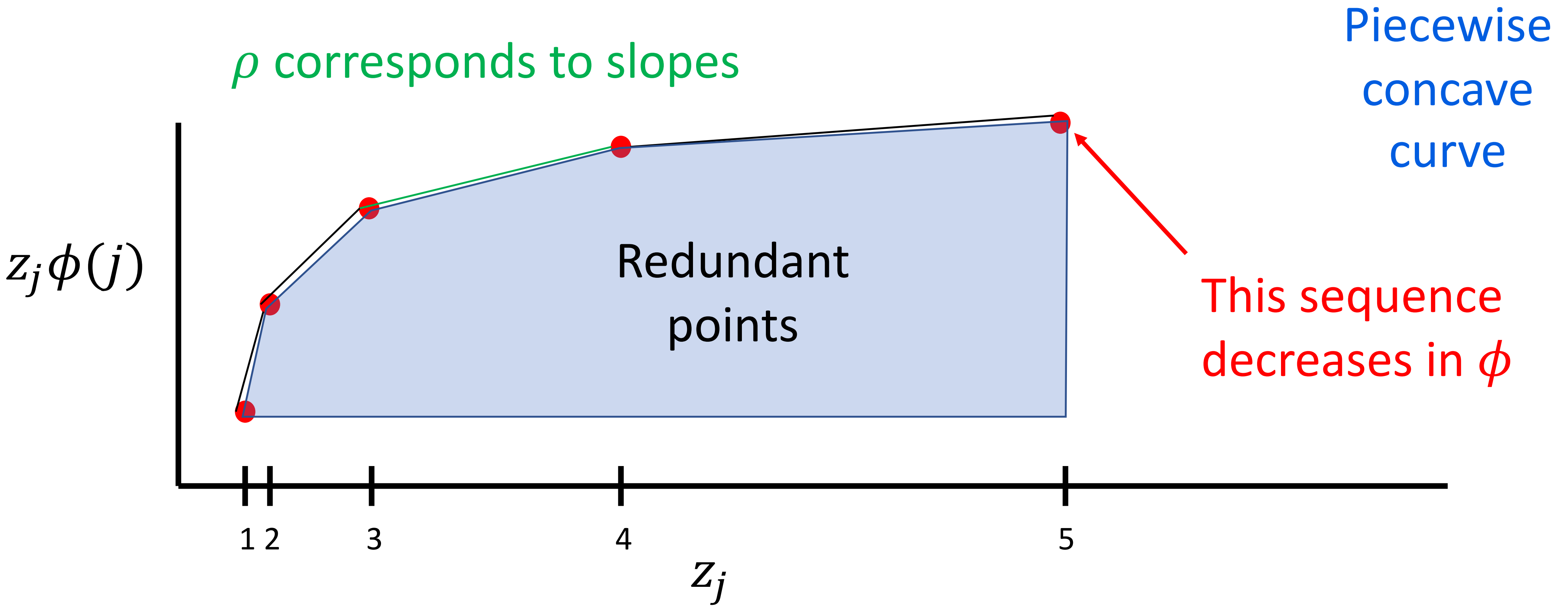
- Agent's algorithm is still greedy – but different potential function

- For platforms j, j' at the same state, define:

$$\rho(j, j') = \frac{z_{j'}\phi(j') - z_j\phi(j)}{z_{j'} - z_j}$$

Multiple Platforms (Flower Setting)

- Each state's non-redundant platforms obey the following structure:



Multiple Platforms (Flower Setting)

- **Proof idea:** Greedy swap argument with new potential, many details.

- The new potential function:

$$\psi(j, \ell) := \begin{cases} \phi(j, 1) \text{ if } \ell = 1 \\ \rho_j(\ell, \ell - 1) \text{ if } \ell > 1 \end{cases}$$

state

Platform index, sorted by ϕ in decreasing order

Multiple Platforms (Flower Setting)

ALGORITHM 4: MULTI-PLATFORM AGENT'S ALGORITHM

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

Output: An optimal feasible subset S of platforms.

```
Remove redundant platforms for each state
Compute the parameters ψ for the (nonredundant) platforms
Sort the platforms in decreasing order ψ(j)
Initialize S := {}
for each platform j in decreasing order of ψ(j) do
    if ψ(j) ≤ u(S) then
        return S
    else
        if j is the first platform for its state then
            Update S := S ∪ {j}
        else
            Update S := S ∪ {j} \ {prev(j)}
        end
    end
end
return S
```

Agent utility

Replace the platform at state *j*

Runs in time

$O(n + m \log m)$

n = # states

m = # total platforms

Multiple Platforms (Flower Setting)

- Is there an efficient designer algorithm?
- The multi-agent algorithm also (essentially) works in the multi-platform setting
  - Same discretization assumptions (potentials, denominator)
  - Exact algorithm
  - Polynomial time when #agents is constant
- Slight difference from old algorithm:
  - Modify the hash function: numerator and denominator of ψ instead

Summary

Recap

- Platform design: model economic activity of online firms
- General case of platform design is strongly NP complete.
- Tractable special case: the flower MDP
- Greedy agent algorithm
- Knapsack-style DP FPTAS for designer w/unbounded potentials
- Under polynomial, discretized potentials, exact DP for k agents (poly(n)$\cdot$ 2^k)
- Similar for multiple platforms
- **Many open directions!**

Future Work

Future Work

- Designer vs. designer
  - Complexity of pure Nash
  - Repeated game settings
- Privacy/fairness questions for agent
- Other classes of tractable MDPs?
- Results for generic classes of agent behavior?
- Many questions are problems of formulation

Learning Theoretic Questions

- What if agent/designer have to learn?

Learning Theoretic Questions

- What if agent/designer have to learn?

- Optimizing over distribution of agent types w/finite support
  - Expected reward: smoothed version of the objective

  - ERM with quasi-linear function class
    - We can solve ERM efficiently if finite support is constant (with discretized, poly-bounded potentials)

Learning Theoretic Questions

- What if agent/designer have to learn?

- Optimizing over distribution of agent types w/finite support
  - **Open:** What if support isn't constant size, or is continuous?
    - ERM via our algorithm no longer computationally efficient

    - Other approaches? Under what conditions is computationally efficient learning possible?

    - Goal: beat $O(2^k poly(n))$ algorithms for k-type supports

Learning Theoretic Questions

- What if agent/designer have to learn?

- Combinatorial bandit setting
  - Suppose the designer is an online bandit
  - Plays combinatorial set S each round
  - **Open: Complicated dependencies between arms + nonlinear rewards**

Learning Theoretic Questions

- What if agent/designer have to learn?
- Repeated game variants of the problem where both agent and designer are learners
- Other equilibria: what is computationally tractable?
- Strategic agents and designer-designer competition

Learning Theoretic Questions

- What if agent/designer have to learn?

- Remove the abstraction of designer rewards:
  - Agents emit data distributions
  - Restricted sampling conditions
  - Goal: Solve some learning problem about agents
  - Connected with data valuation
  - How to design sampling environment?

Grand Vision

- **Design environments** which generate useful, sampleable data
- **Model economics** of companies dependent on information economy
- **Model strategic behavior** of online firms and their users
- **Reinforcement learning** aided by environment design
- **Manipulation and resistance** of learning agents