The Platform Design Problem

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

Columbia University

Strategic ML Workshop @ NeurIPS 2021

The Data-Collection Problem

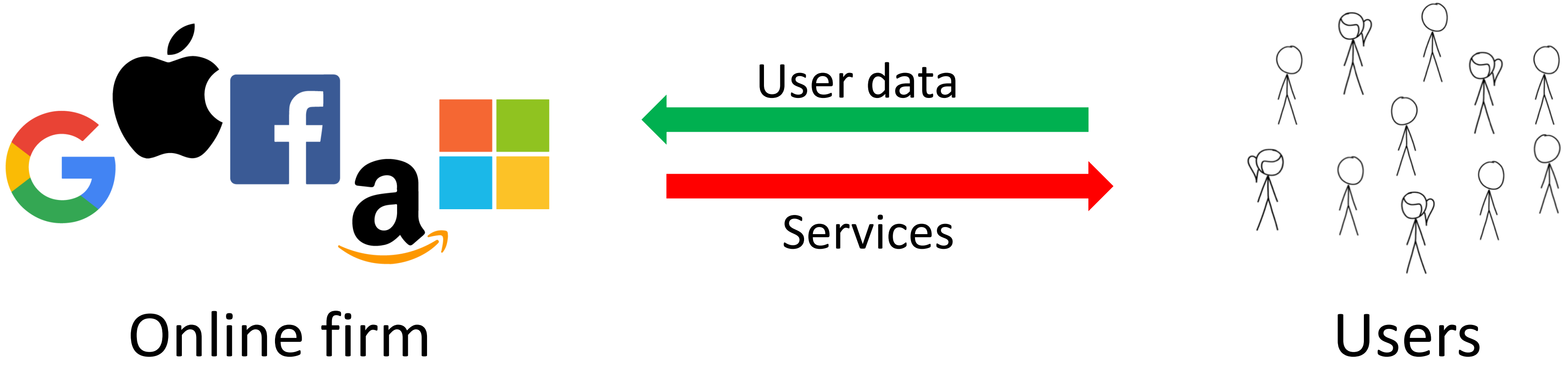
- Modern machine learning requires large amounts of high-quality data
- Collecting supervised labels is expensive
- Unsupervised learning is challenging to use
- Is it possible to create environments which generate useful data?
  - Ex: Reddit users provide sarcasm labels using the “/s” tag

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



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

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)

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.

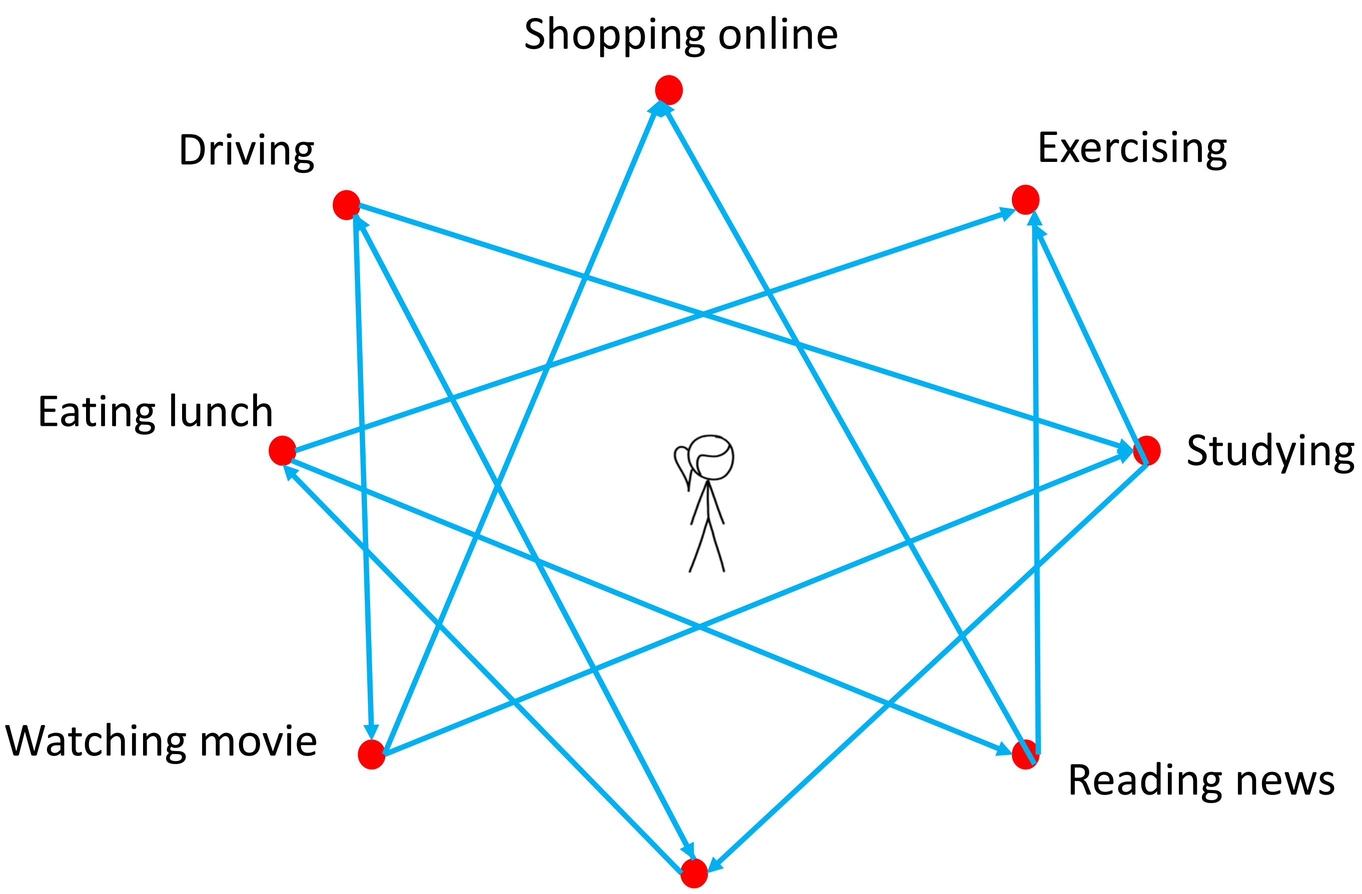
Formal Problem Statement

- 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:

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

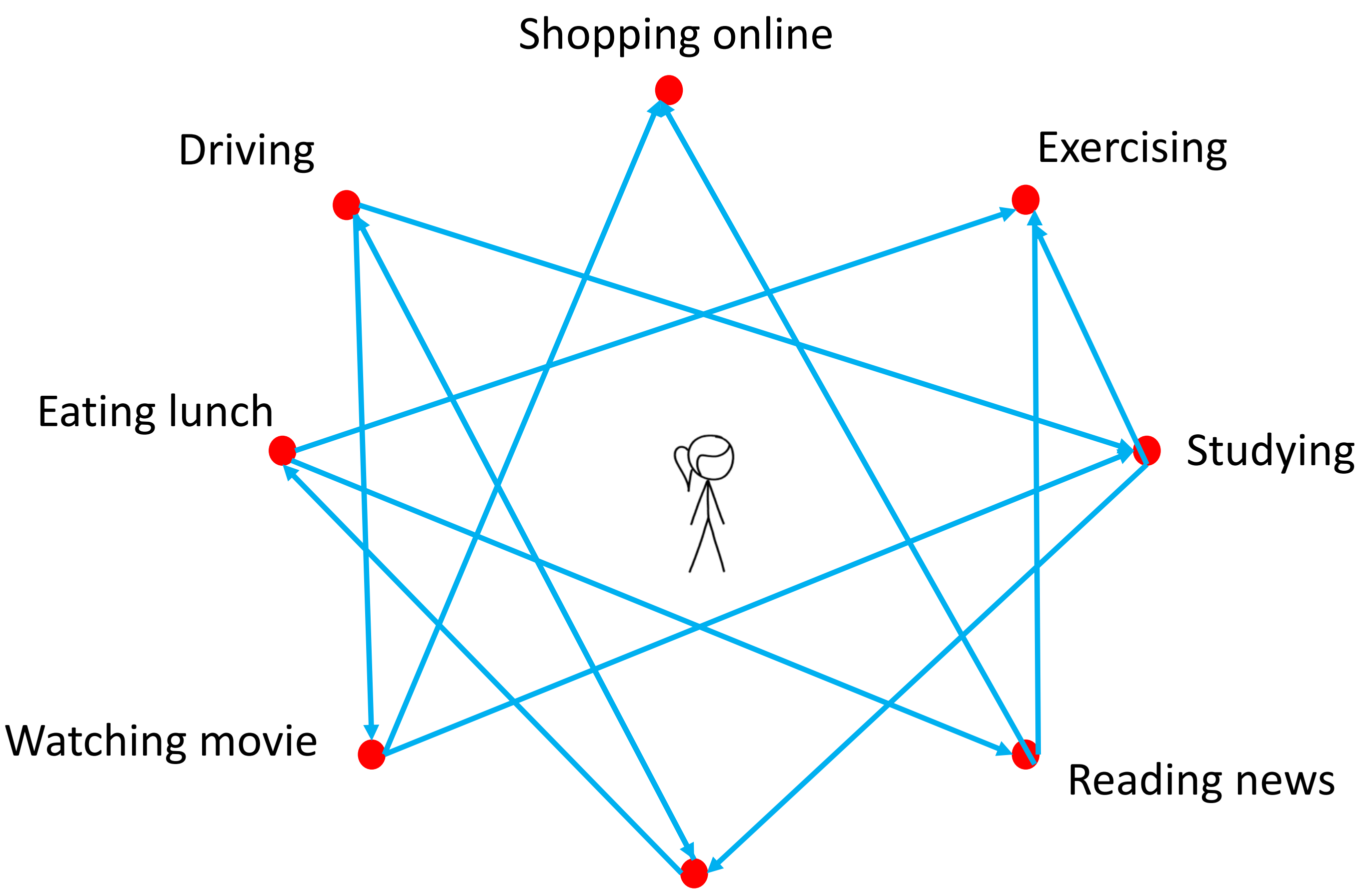
General Case

Picture of the General Case



Agent’s Life

Picture of the General Case



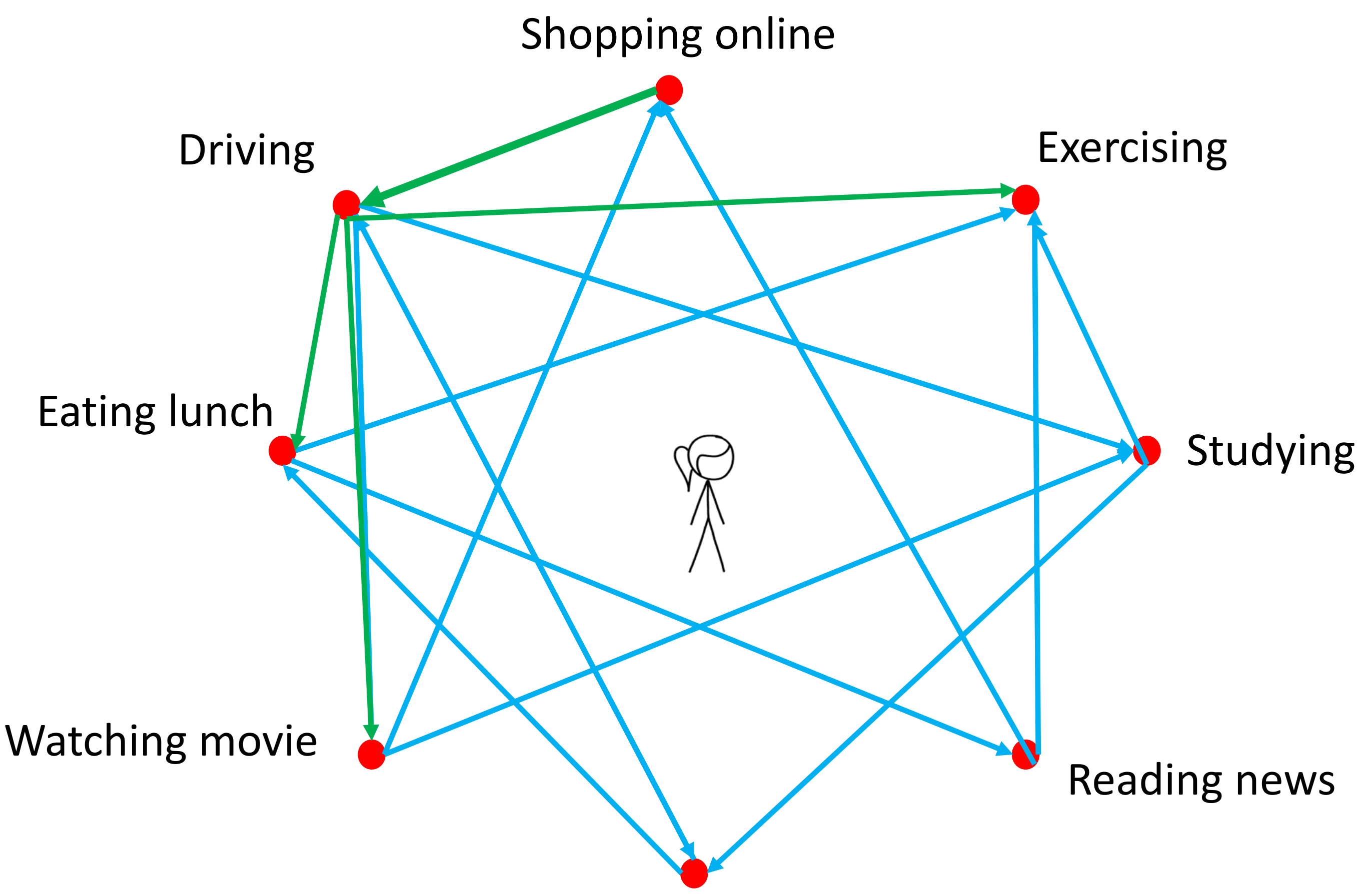
Agent's Life

What platforms should I build?

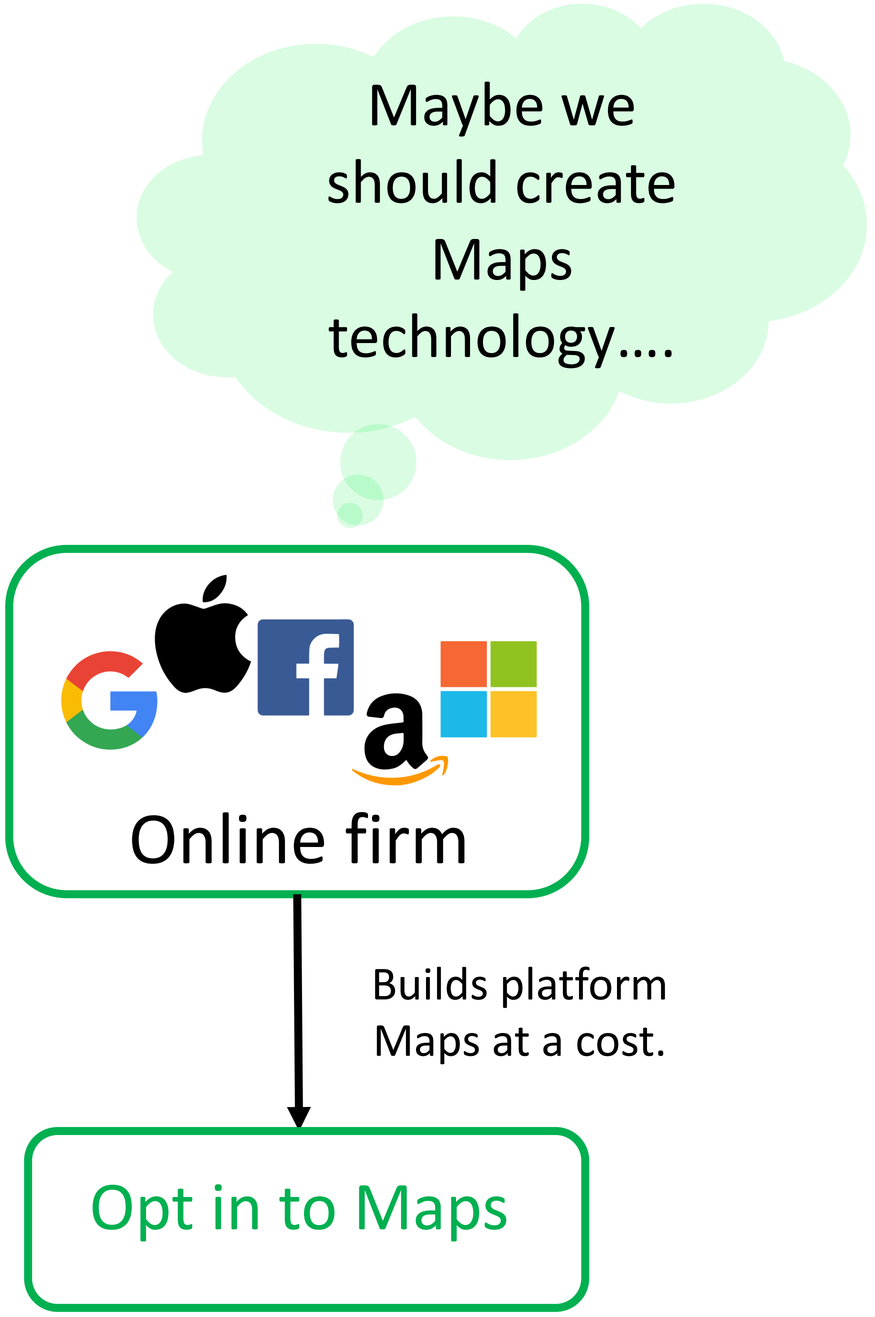
Online firm

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



Computational Tractability I: General Case

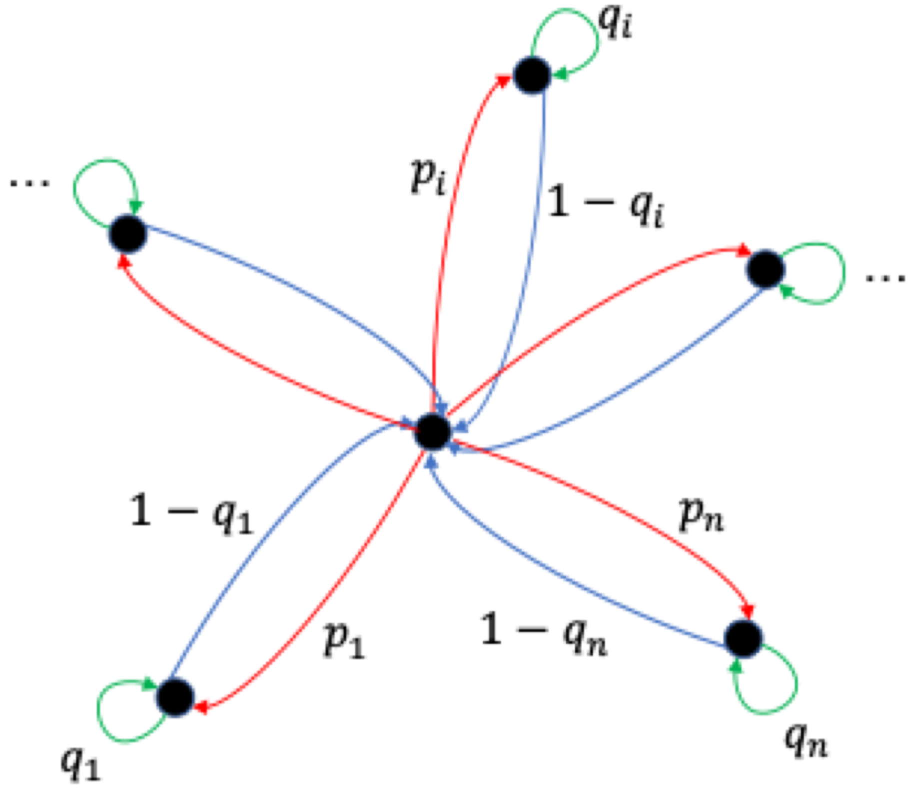
- It is strongly NP-hard to decide whether the Designer can obtain positive profit – and therefore hard to approximate.

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

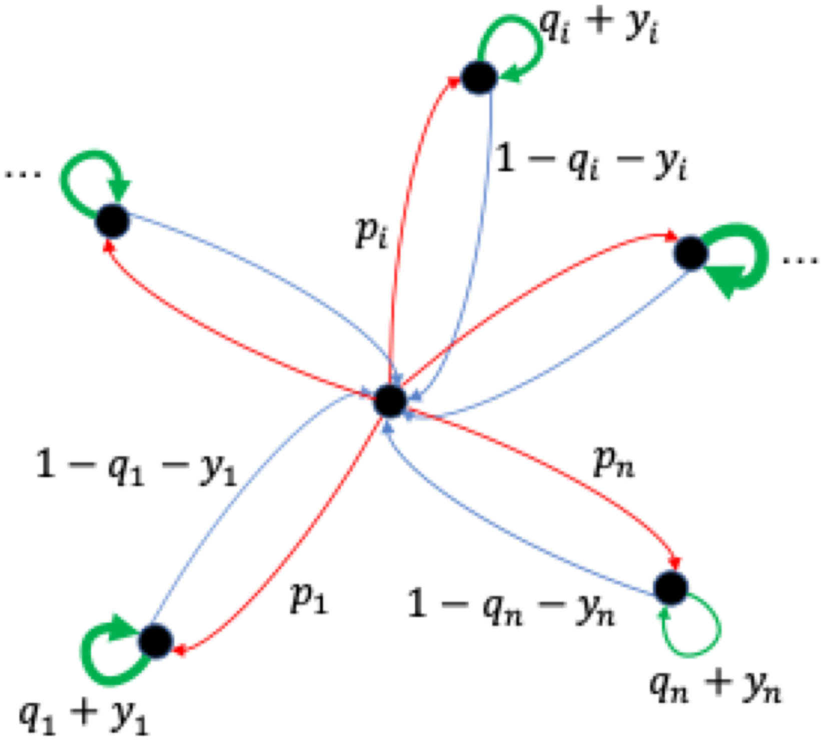
- In economic terms, the reduction exploits the complexity of "complementary goods."
  - Ex: Brick-and-mortar retail ads help the Agent discover the store, Maps helps the Agent get to the store.

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 is optimal)
  - Admits a DP upon discretization (knapsack DP)

The Designer's Dynamic Program

- Designer's profit function for set of platforms S:

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

- Assume z is discretized and costs are polynomially bounded
- Goal: (1 - ϵ) 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:

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

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

Extensions

Multiple Agents

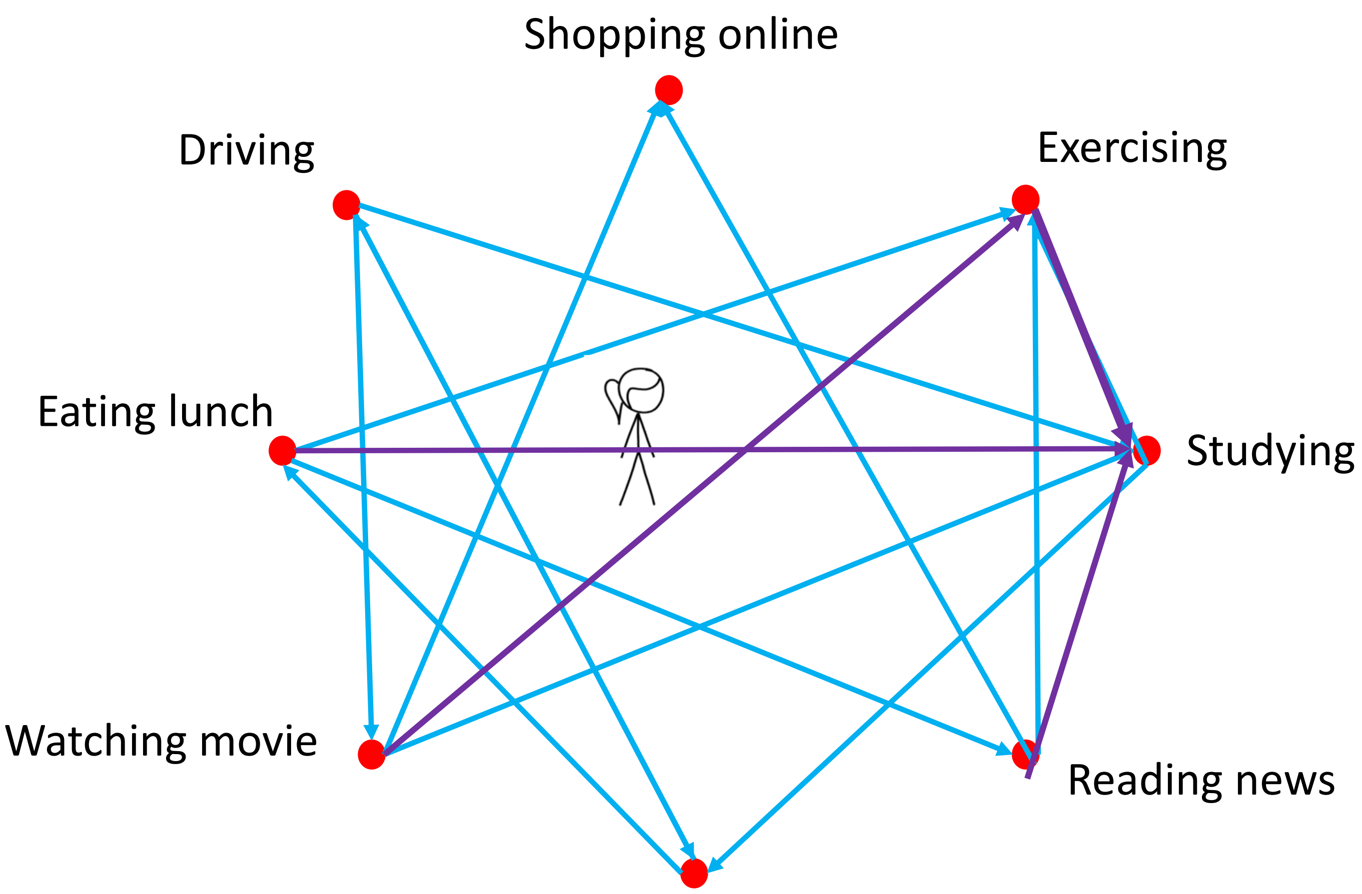
- Replace designer objective with summation over agents:

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

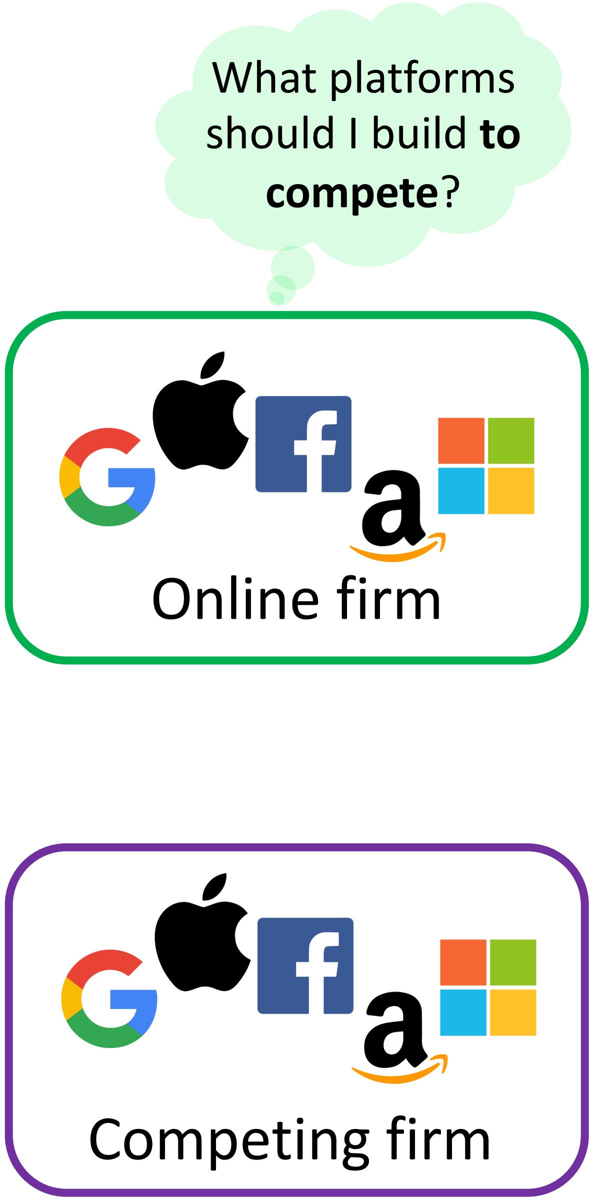
- An exact polytime DP exists if #agents is constant.
  - Exponential in #agents
  - Also require potentials ϕ_i to be discretized by δ' with poly size.

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

Designer Competition



Agent's Life



Future Work

- Designer vs. Designer
  - Complexity of pure Nash
  - Repeated game settings

- Privacy/fairness questions for Agent

- Unknown rewards for Designer and Agent
  - Learning in games
  - Strategic Agents

- And many more… please reach out at kiran.vodrahalli@columbia.edu if you would like to chat!