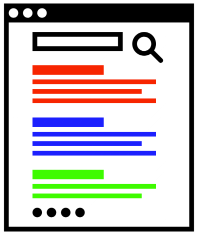
RUMs from Head-to-Head Contests

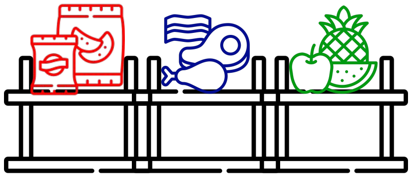
M. Almanza, F. Chierichetti, R. Kumar, A. Panconesi, A. Tomkins

SAPIENZA UNIVERSITÀ DI ROMA Google

Discrete choice

- Important model in Economics and Informatics
- Widely used in studying consumer demand
- Especially important in online/interactive settings (search results, product alternatives, recommendations)

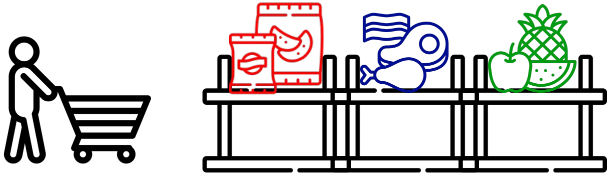




Discrete choice

- In discrete choice models, agents need to pick a choice from a finite set (*slate*) of possible choices

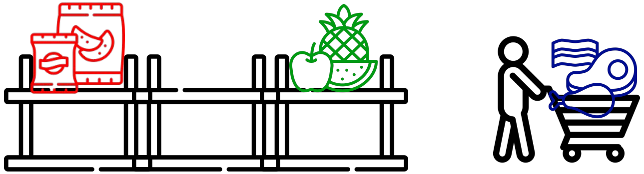
- Learning problem: given a set of observed interactions, can we predict future ones?



Random Utility Models (RUMs)

- A model for discrete choice (McFadden, Nobel prize)
- Each user has a preference for the items (i.e., a permutation)

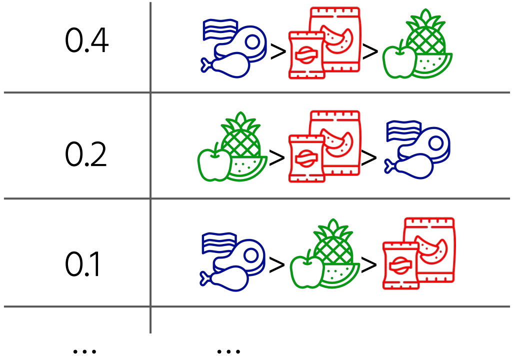
- Given a set of choices, a user selects the highest ranked item available



Random Utility Models (RUMs)

- A *RUM* on **[n]** is a probability distribution **D** over the permutations of **[n]**
  - Each permutation can be seen as a user *type*

- Given a slate, the probability that one of its items is picked is equal to the probability of that item appearing before all the others (of the slate) in a permutation sampled from **D**



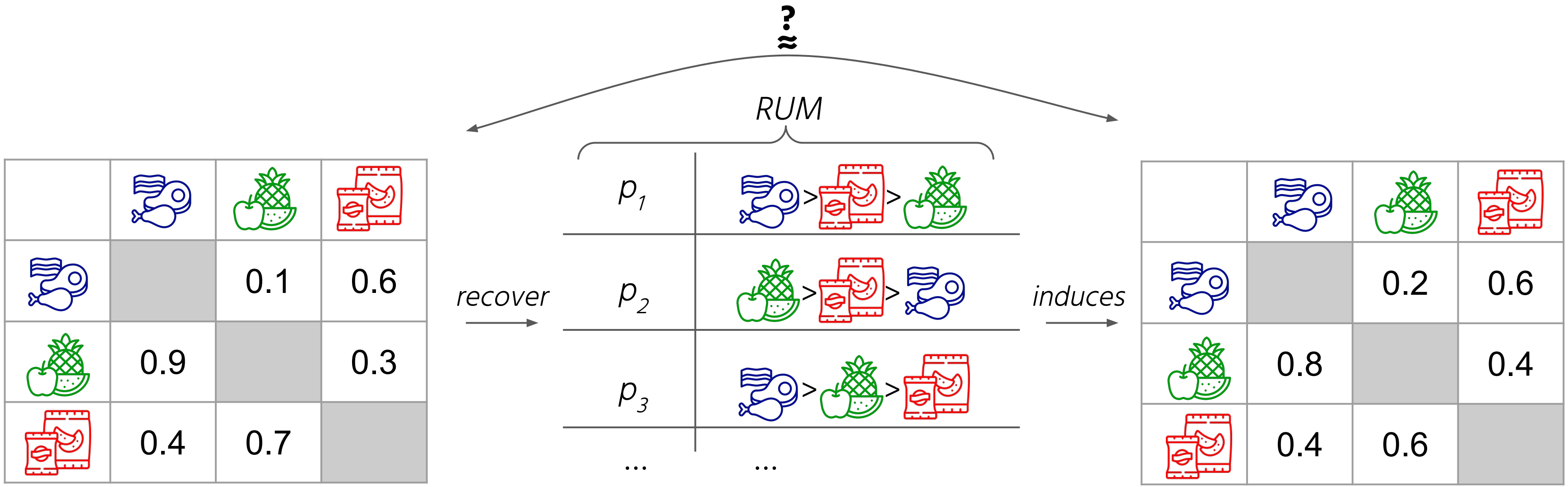
Pairwise Choice

- Each slate is a pair
  - head-to-head competitions, online experiences comparing one item and an alternative
- Represented as tournament matrix
  - entry $M_{a,b}$ represents the (empirical) probability of *a* beating *b*

			
		0.1	0.6
	0.9		0.3
	0.4	0.7	

Pairwise Choice

- Given a tournament matrix, can we **recover** a good RUM for it?
  - A RUM is good if the tournament matrix it induces is close to the original one



Previous works

- Different choice models have been proposed for representing a tournament matrix:
  - *Blade-Chest* — Chen & Joachims, WSDM '16
  - *Majority Vote* — Makhijani & Ugander, WWW '19
  - *Two-level model* — Veerathu & Rajkumar, NeurIPS '21
  - ...

Contributions

- An algorithm recovering a RUM that **approximately minimizes** the **average error** over the pairs (in the induced tournament matrix) in **polynomial time**

- A practical **implementation** of the previous algorithm finding a **near-optimal** RUM without the polytime guarantee, but that **performs well** in practice

LP - Ellipsoid method

- Grötschel, Lovász, Schrijver (1988): the ellipsoid update step can be replaced by a Separation Oracle that, given a point, returns either:
  - The point **is a valid** solution
  - The point **is not a valid** solution, and here's a hyperplane **separating** it from the set of valid solutions

- Converges in polynomially many steps

Recovering a RUM

- Consider the LP to find a RUM approximating the input matrix
  - A variable for each permutation (its probability) $\longrightarrow$ exponentially many

Recovering a RUM

- Consider the LP to find a RUM approximating the input matrix
  - A variable for each permutation (its probability) $\longrightarrow$ exponentially many
- Consider the dual: is it better?
  - polynomially many variables
  - exponentially many constraints

Recovering a RUM

- Consider the LP to find a RUM approximating the input matrix
    - A variable for each permutation (its probability) ⟶ exponentially many
- Consider the dual: is it better? **Yes: we have an ε-apprx Separation Oracle!**
    - polynomially many variables
    - exponentially many constraints
- The validity of the *permutation constraints* of the dual can be determined by solving an instance of **Minimum Feedback Arc Set** (MinFAS)
    - NP-hard but can be approximated in polynomial time ⟶ approximated oracle

Recovering a RUM

- Consider the LP to find a RUM approximating the input matrix
- Solve the dual with the ellipsoid method and the ε-apprx. Separation Oracle
    - Converges in poly steps ⟶ poly many constraints
- Put them back in the primal and solve it to get a RUM that is within ε of the optimal solution

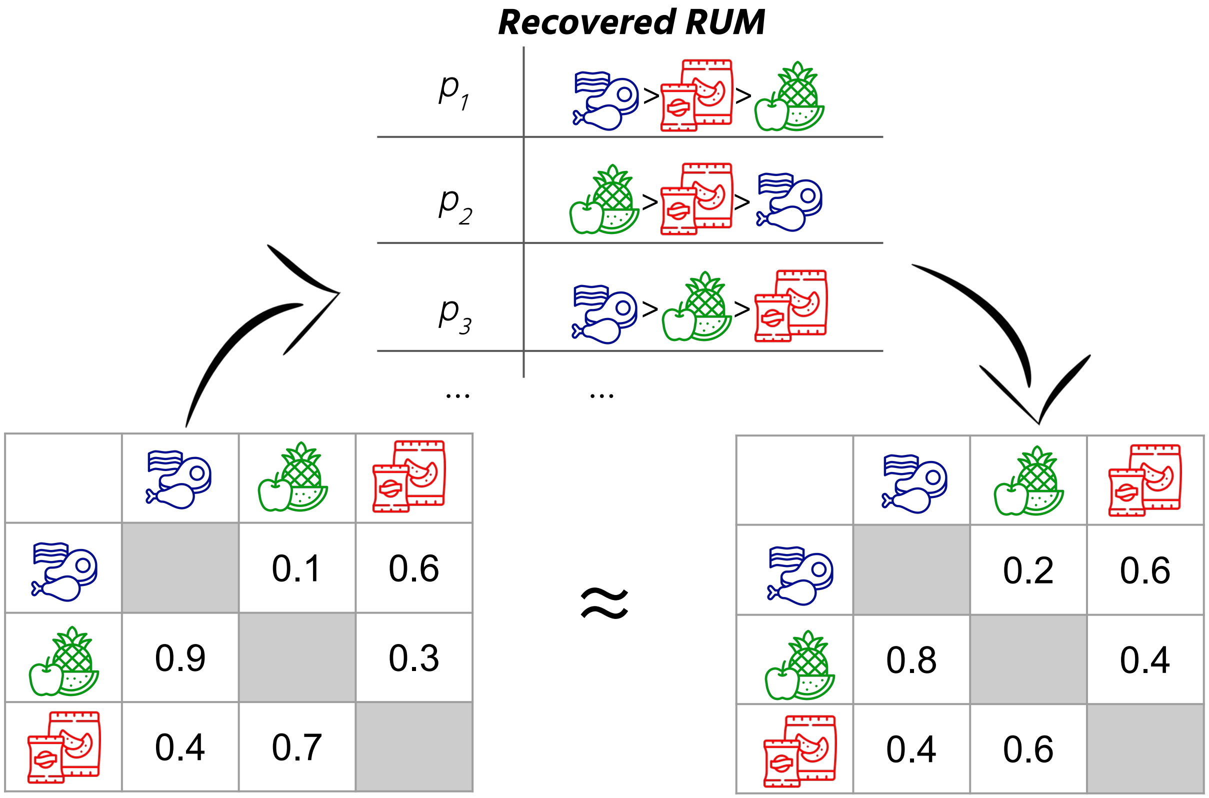
- *Issue: ellipsoid is not really feasible in practice*

Recovering a RUM (quickly)

- Heuristic for the dual:
  - While we can find a violated constraint:
    - Add it to the (dual) LP and solve it

- How to find violated constraints (quickly)?
  - Start from a random permutation
  - Perturb it to find a minimal FAS
  - Check whether the constraint of the permutation given by the Minimal FAS is violated: if so, add that constraint to the dual

Experimental results: RUM recovery



Input tournament matrix

RUM-induced matrix

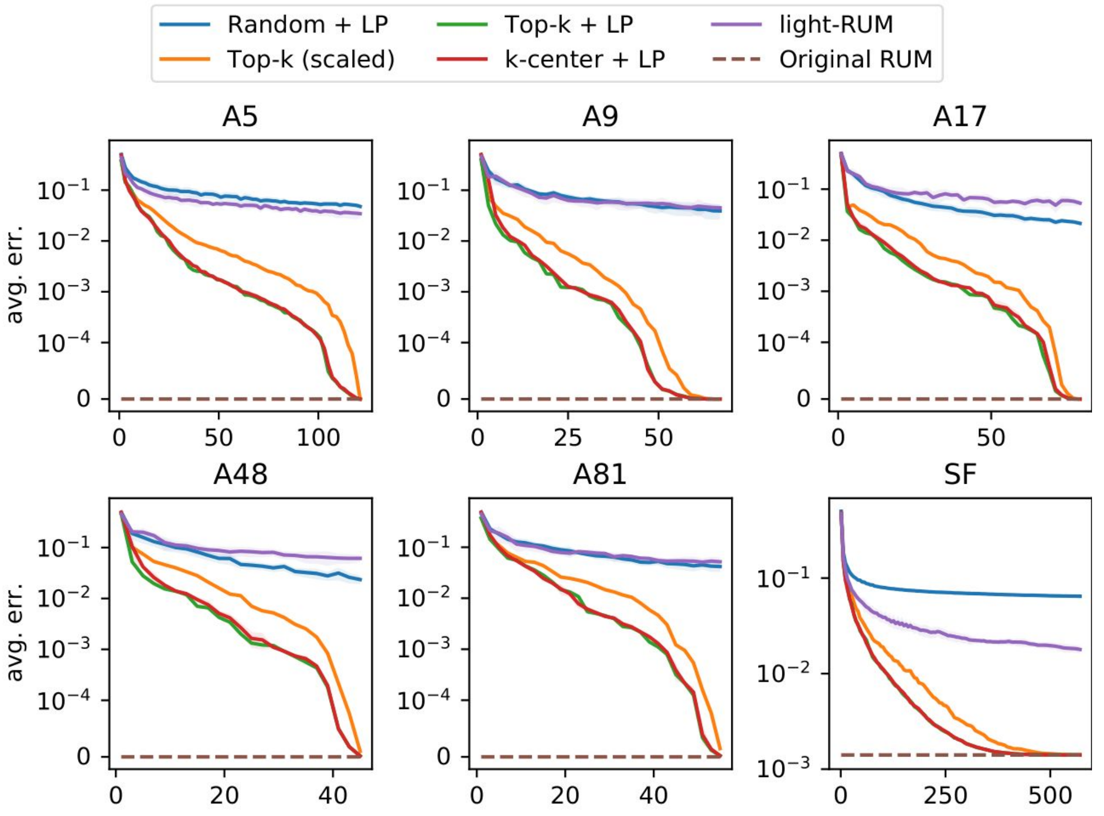
Experimental results

- Our algorithm manages to recover **exactly** most of the (real-world) tournament matrices we tested

Dataset	n	avg. err.	lower bound on avg. err.
A5	16	0	0
A9	12	0	0
A17	13	0	0
A48	10	0	0
A81	11	0	0
SF	35	0.001408	0.001408
Jester	100	0.000461	0

Experimental results

- We propose different heuristics to find compact RUMs
- Even with a small number of permutations (x-axis), the best ones achieve a low avg error



Thanks!

Questions/Comments?

Experimental results

- We can use the LP approach to "fit" any set of permutations to a tournament matrix
- We propose different heuristics to find a compact representation of the tournament matrix using only a fixed number of permutations (e.g. only 10 permutations)

Dataset	Original RUM	Random + LP	Top-k (scaled)	Top-k + LP	k-center + LP	light-RUM
A5	0	0.156 ±0.020	0.059	**0.041**	0.045 ±0.004	0.111 ±0.016
A9	0	0.133 ±0.030	0.031	**0.010**	0.011 ±0.002	0.119 ±0.023
A17	0	0.112 ±0.014	0.034	**0.014**	**0.014 ±0.001**	0.119 ±0.033
A48	0	0.107 ±0.022	0.043	**0.015**	**0.015 ±0.001**	0.119 ±0.029
A81	0	0.129 ±0.024	0.056	0.047	**0.046 ±0.005**	0.113 ±0.021
SF	0.00141	0.149 ±0.008	0.113	**0.104**	0.105 ±0.008	0.127 ±0.011
Jester	0.00046	0.168 ±0.008	0.119	**0.108**	**0.108 ±0.003**	0.121 ±0.006