Segmentation

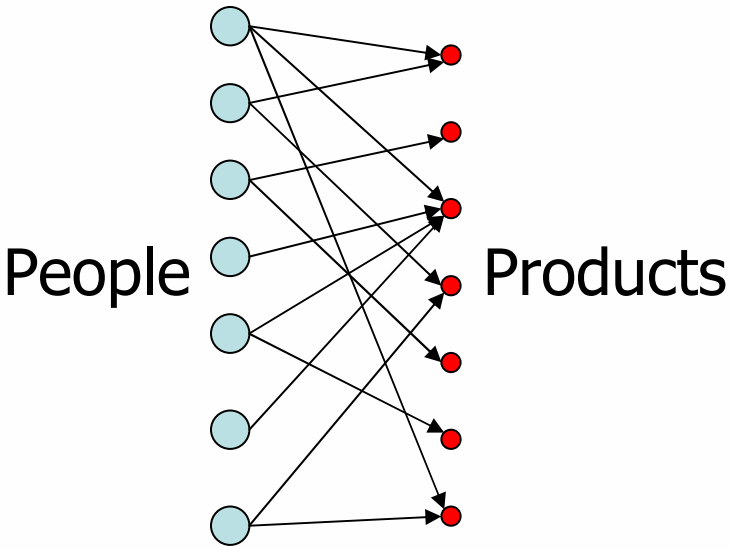
Prabhakar Raghavan

Joint w/Jon Kleinberg and Christos Papadimitriou

Optimization

- Basis for modern decision theory
- Given a feasible decision space D
- Extremize an objective function F
- Linear programming
- Integer programming
- …

Simple example - catalogs

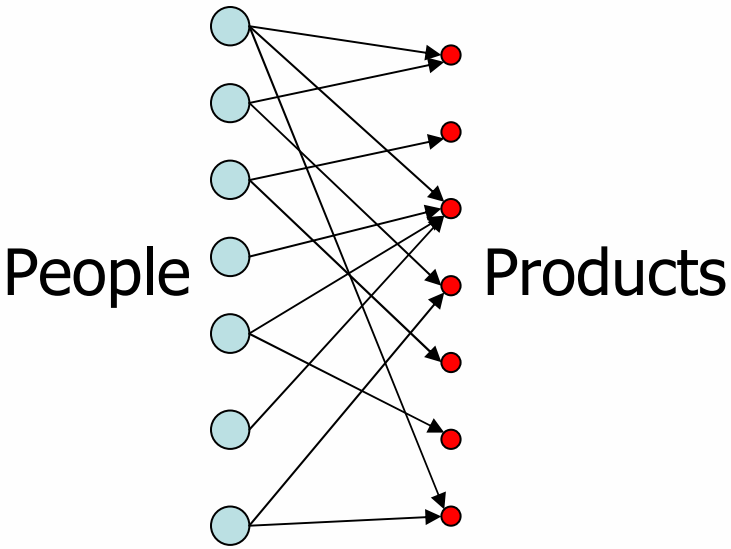


Bipartite graph: *(i,j)* denotes person *i* likes product *j*.

If we can print a catalog that can hold *k* items … take *k* items of highest degree.

Here **D** is the set of catalogs with *k* items; and degree is a proxy for **F**.

2-catalog segmentation



Bipartite graph: *(i,j)* denotes person *i* likes product *j*.

If we can print a catalog that can hold *k* items … take *k* items of highest degree.

What if we could print two catalogs of *k* items each?

Each person receives only one catalog – the one "closer" to them.

Clean objective: maximize revenue, proportional to popularity of items printed in the catalog that you receive.

⇒ Clean measure of value of data. (Can we extract it?!)

Enterprise optimization

- Enterprise objective function

  $F = \sum_{i \in C} f_i(x)$

- C = set of customers
- f_i is the objective function for customer i

  $= g(x, y_i)$

  So want to optimize $F = \sum_{i \in C} g(x, y_i)$.

  Aggregation: Optimize $g(x, \sum_{i \in C} y_i)$.

Market segmentation

- The enterprise could tailor a separate policy for each customer

  Optimize $\sum_{i \in C} g(x_i, y_i)$.
- Too many segments – uneconomical.
- Instead: partition customers into *s* segments.
- For each segment, tailor policy and optimize subject to D.

Mathematically

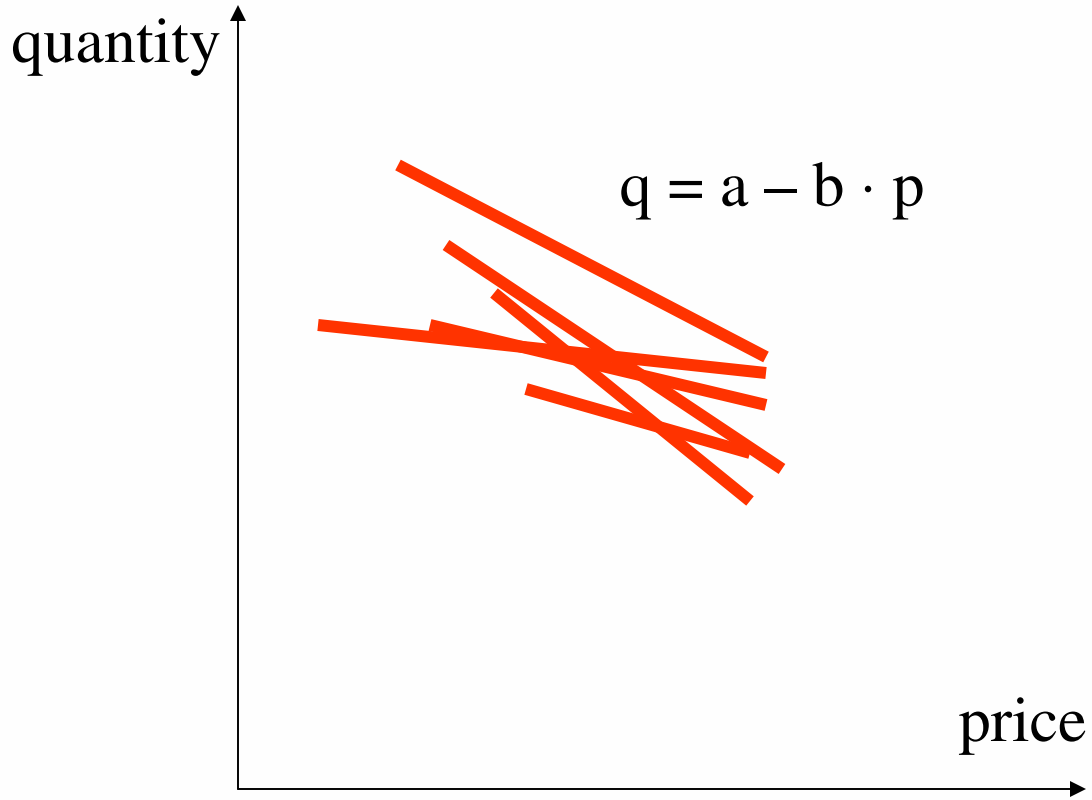
- Find policies x_1 through x_s
- Assign to each customer the best policy for that customer, thus
- Optimize $\sum_{i \in C} \max_{1 \le j \le s} g(x_j, y_i)$.

Clustering formulation

- Partition customer set C into s segments C_1 through C_s
- Find best policy for each segment

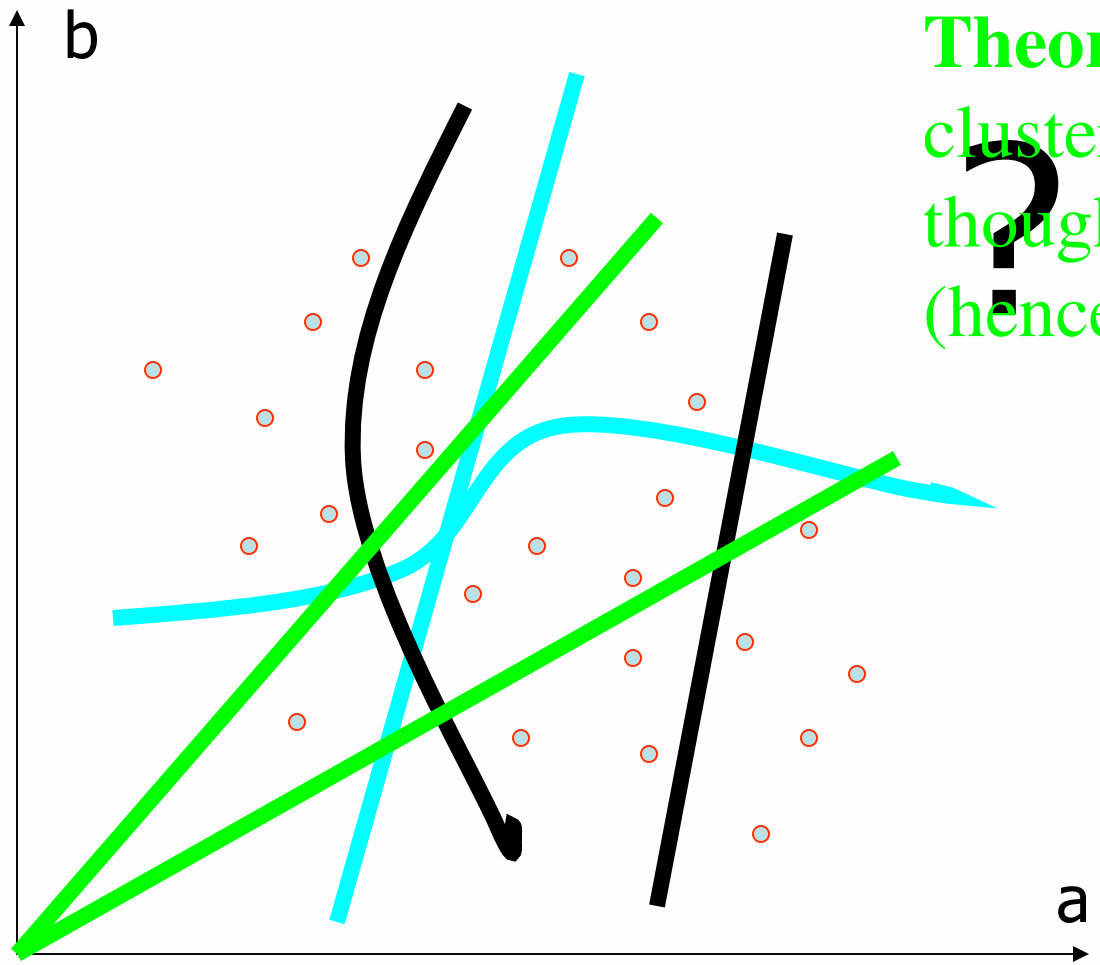
Optimize $\sum_{1 \le j \le s} \max_{x \in D} \sum_{i \in C_j} f_i(x_j)$

Toy example



Segment monopolistic market to maximize revenue

In the a-b plane



Theorem: Optimum clustering is by lines though the origin (hence: $O(n^2)$ DP)

Catalog segmentation, s=2

- Given a set U of items
- And n subsets of U (the customers), U_1 through U_n
- Find subsets X_1, X_2 of U each of size k to maximize

$$\sum_{1 \le i \le n} \max \{|X_1 \cap U_i|, |X_2 \cap U_i|\}$$

Variable *k* segmentation

- Instead of fixing *k* (number of segments) a priori
- Instead: a linear penalty for each new segment

Data mining: fun and profit

- For any optimization problem, we can define a segmented version
- For "the right" optimization problems, segmentation is data mining for profit
  - E.g., catalog segmentation

But for combinatorial optimists

- Could take any problem (say TSP) and derive a segmented version
- Given *n* customers each with a cost matrix on cities
- Find *k* segments of customers, each assigned a TSP tour, to minimize total tour cost

Life is hard …

- Even the most trivial optimization problems become hard in their segmented form
- E.g., maximizing a linear function in the unit ball
- (Most of these can be solved efficiently for fixed *s*)

Dense catalog problems

- Suppose every customer likes a constant fraction of the items
- Then, the following sampling algorithm is a PTAS for fixed *k*:
  - Pick a sample of *c* log *n* customers.
  - Enumerate the segmentations induced by each *s*-partition of this sample
  - Take the best of these

Why does it work? Sketch *s*=2:

- Either one segment of the optimal solution is dominant (produces most of the revenue) OR
- Both segments contribute to optimal revenue
  - Claim 1: our sample hits $\Omega(\log n)$ customers from each optimal segment

Why does it work?

- Claim 2: there is a partition in our sample that gets close to optimal revenue.
- The running time is $n^{O(k)}$

Hypercube segmentation

- Consider *n* customers sited at some vertices of the *d*-dim hypercube
- Think of the *d* dimensions as attributes of a movie – violence, drama, etc.
  - Each customer has some set of attributes they want/don't.

Hypercube segmentation

- Want to produce *s* movies (each a vertex of the cube) s.t.:
- Maximize customer utility, measured by: Hamming overlap of movie with customer prefs

Trivial 0.5-approximation

- Create a single movie: in each dimension, pick the majority value
- Since Hamming overlap (even weighted) is linear, the result follows
- Can we do better?

Better approximations

- Lemma: in any set of customers, there is a *typical* customer: making the perfect movie for that customer yields total utility at least 0.828 of optimal.
- Counterpoint: this is about the best possible; there are customer sets for which no customer is better than 0.833 of optimal total utility.

How do we use this?

- Algorithm 1:
  - Enumerate all *k*-subsets of customers, make them the movies.
  - Assign each other customer to the nearest movie.
  - Runs in time $n^{O(k)}$ and yields utility at least 0.828 of optimal.

Faster+weaker approximations

- Pick random samples of customers
- Use these as movies
- Claim 1: In time $nk(1/\varepsilon)^k$ produce utility with expectation 0.828- ε of optimal.
- Claim 2: In time $O(nL)$, with probability 1-o(1), within 0.828-0.328 $e^{(-L/2k)}$ of optimal.
- Alon/Sudakov: 1- ε approximation.

Competitive segmentation

- So far we've done with a monopolistic enterprise.
- What happens with competition?
- Consider a 2-person game with a payoff matrix.
  - Think of the strategies for two competing enterprises.

Segmented games

- Now there are *n* payoff matrices, one for each customer.
- Each player gets to pick *s* strategies, one for each of *s* customer segments.
- What does this mean?

Segmented games: questions

- Do the players move concurrently or sequentially?
- Are they computationally bounded?
- Must both players use the same *s*?

Concrete segmented game

- Each entry ±1
  - A customer is won/lost by each player.
- Players play sequentially
  - Player 1 segments; so has a row strategy for each segment
- Then Player 2 faces LP segmentation problem … NP-hard

First player's move

- So Player 1 creates instance of NP-hard problem for Player 2!
  - Tries to minimize optimum for Player 2
  - This in turn is NP-hard!
- But does he have to?
  - Could create an instance where finding a good optimum is computationally hard

Geometric segmentation games: Fast food segmentation

- Market consists of points in the plane (customers)
- Players McD and BK alternate in placing outlets in the plane
- Each customer “captured” by the nearest outlet

Open

- Better algorithms for cases of catalog segmentation
- Anything for segmented games
- Geometric segmentation
- (Even a stylized) segmentation problem for which some natural clustering algorithm works well