# Distributed Submodular Maximization in Massive Datasets

Alina Ene

Joint work with

Rafael Barbosa, Huy L. Nguyen, Justin Ward

# Combinatorial Optimization

- Given
  - A set of objects V
  - A function f on subsets of V
  - A collection of feasible subsets I
- Find
  - A feasible subset of I that maximizes f
- Goal
  - Abstract/general f and I
  - Capture many interesting problems
  - Allow for efficient algorithms

# Submodularity

We say that a function $f : 2^V \to \mathbb{R}_+$ is submodular if:

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$$

We say that $f$ is monotone if:

$$f(A) \leq f(B), \quad \forall A \subseteq B$$

Alternatively, f is submodular if:

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$$

for all $A \subseteq B$ and $x \notin B$

Submodularity captures diminishing returns.

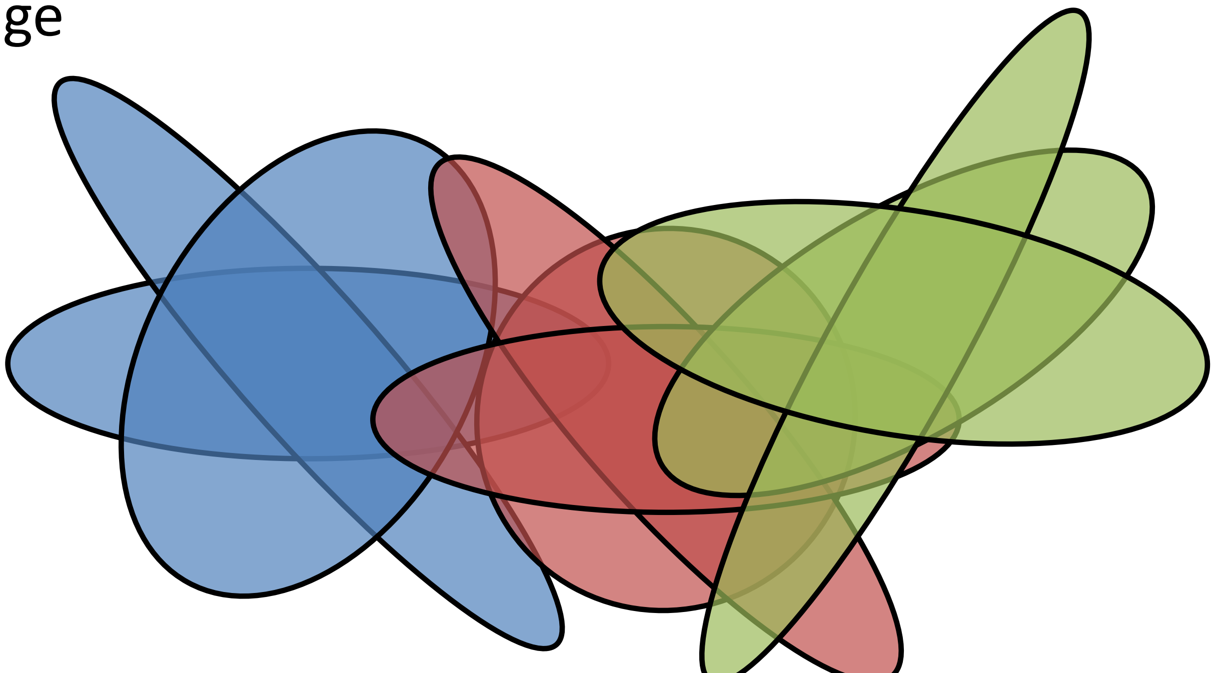
# Submodularity

Examples of submodular functions:

- The number of elements covered by a collection of sets
- Entropy of a set of random variables
- The capacity of a cut in a directed or undirected graph
- Rank of a set of columns of a matrix
- Matroid rank functions
- Log determinant of a submatrix

# Example: Multimode Sensor Coverage

- We have distinct locations where we can place sensors
- Each sensor can operate in different modes, each with a distinct coverage profile
- Find sensor locations, each with a single mode to maximize coverage



# Example: Identifying Representatives In Massive Data

# Example: Identifying Representative Images

- We are given a huge set X of images.
- Each image is stored multidimensional vector.
- We have a function d giving the difference between two images.
- We want to pick a set S of at most k images to minimize the loss function:

$$L(S) = \frac{1}{|X|} \sum_{e \in X} \min_{r \in S} d(e, r)$$

- Suppose we choose a distinguished vector $e_0$ (e.g. 0 vector), and set:

$$f(S) = L(\{e_0\}) - L(S \cup \{e_0\})$$

- The function f is submodular. Our problem is then equivalent to maximizing f under a single cardinality constraint.

# Need for Parallelization

- Datasets grow very large
  - TinyImages has 80M images
  - Kosarak has 990K sets
- Need multiple machines to fit the dataset
- Use parallel frameworks such as MapReduce

# Problem Definition

- Given set V and submodular function f
- Hereditary constraint I (cardinality at most k, matroid constraint of rank k, … )
- Find a subset that satisfies I and maximizes f
- Parameters
  - $n = |V|$
  - k : max size of feasible solutions
  - m : number of machines

# Greedy Algorithm

Initialize S = {}

While there is some element x that can be added to S:

Add to S the element x that maximizes the marginal gain $f(S \cup \{x\}) - f(S)$

Return S

# Greedy Algorithm

- Approximation Guarantee:
  - 1 - 1/e for a cardinality constraint
  - 1/2 for a matroid constraint
- Runtime: O(nk)
  - Need to recompute marginals each time an element is added
  - Not good for large data sets

Mirzasoleiman, Karbasi, Sarkar, Krause '13

# Distributed Greedy

Mirzasoleiman, Karbasi, Sarkar, Krause '13

# Performance of Distributed Greedy

- Only requires 2 rounds of communication
- Approximation ratio is:

$$\frac{\left(1-\frac{1}{e}\right)^2}{\min(m,k)}$$

(where m is number of machines)

- If we use the optimal algorithm on each machine in both phases, we can still only get:

$$\frac{1}{\min(m,k)}$$

# Performance of Distributed Greedy

- If we use the optimal algorithm on each machine in both phases, we can still only get:

$$\frac{1}{\min(m,k)}$$

- In fact, we can show that using greedy gives:

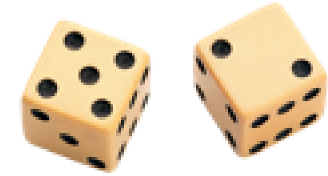
$$O\left(\frac{1}{\sqrt{\min(m,k)}}\right)$$

- Why?
  - The problem doesn't have optimal substructure.
  - Better to run greedy in round 1 instead of the optimal algorithm.

# Revisiting the Analysis

- Can construct bad examples for Greedy/optimal
- Lower bound for any poly(k) coresets (Indyk et al. ’14)
- Yet the distributed greedy algorithm works very well on real instances
- Why?

# Power of Randomness



- Randomized distributed Greedy
  - Distribute the elements of V randomly in round 1
  - Select the best solution found in rounds 1 & 2
- Theorem: If Greedy achieves a C approximation, randomized distributed Greedy achieves a C/2 approximation in expectation.

# Intuition

- If elements in OPT are selected in round 1 with high probability
  - Most of OPT is present in round 2 so solution in round 2 is good
- If elements in OPT are selected in round 1 with low probability
  - OPT is not very different from typical solution so solution in round 1 is good

# Analysis (Preliminaries)

- Greedy Property:
  - Suppose:
    - x is not selected by greedy on S∪{x}
    - y is not selected by greedy on S∪{y}
  - Then:
    - x and y are not selected by greedy on S∪{x,y}
- Lovasz extension $\hat{f}$ : convex function on $[0,1]^V$ that agrees with $f$ on integral vectors.

# Analysis (Sketch)

- Let X be a random 1/m sample of V
- For e in OPT, let $p_e$ be the probability (over choice of X) that e is selected by Greedy on $X \cup \{e\}$
- Then, expected value of elements of OPT on the final machine is $\hat{f}(\mathbf{p})$
- On the other hand, expected value of rejected elements is $\hat{f}(1_{OPT} - \mathbf{p})$

# Analysis (Sketch)

The final greedy solution T satisfies:

$$\mathbb{E}[f(T)] \geq \alpha \cdot \hat{f}(\mathbf{p})$$

The best single machine solution S satisfies:

$$\mathbb{E}[f(S)] \geq \alpha \cdot \hat{f}(1_{OPT} - \mathbf{p})$$

Altogether, we get an approximation in expectation of:

$$\frac{\alpha}{2}$$

# Generality

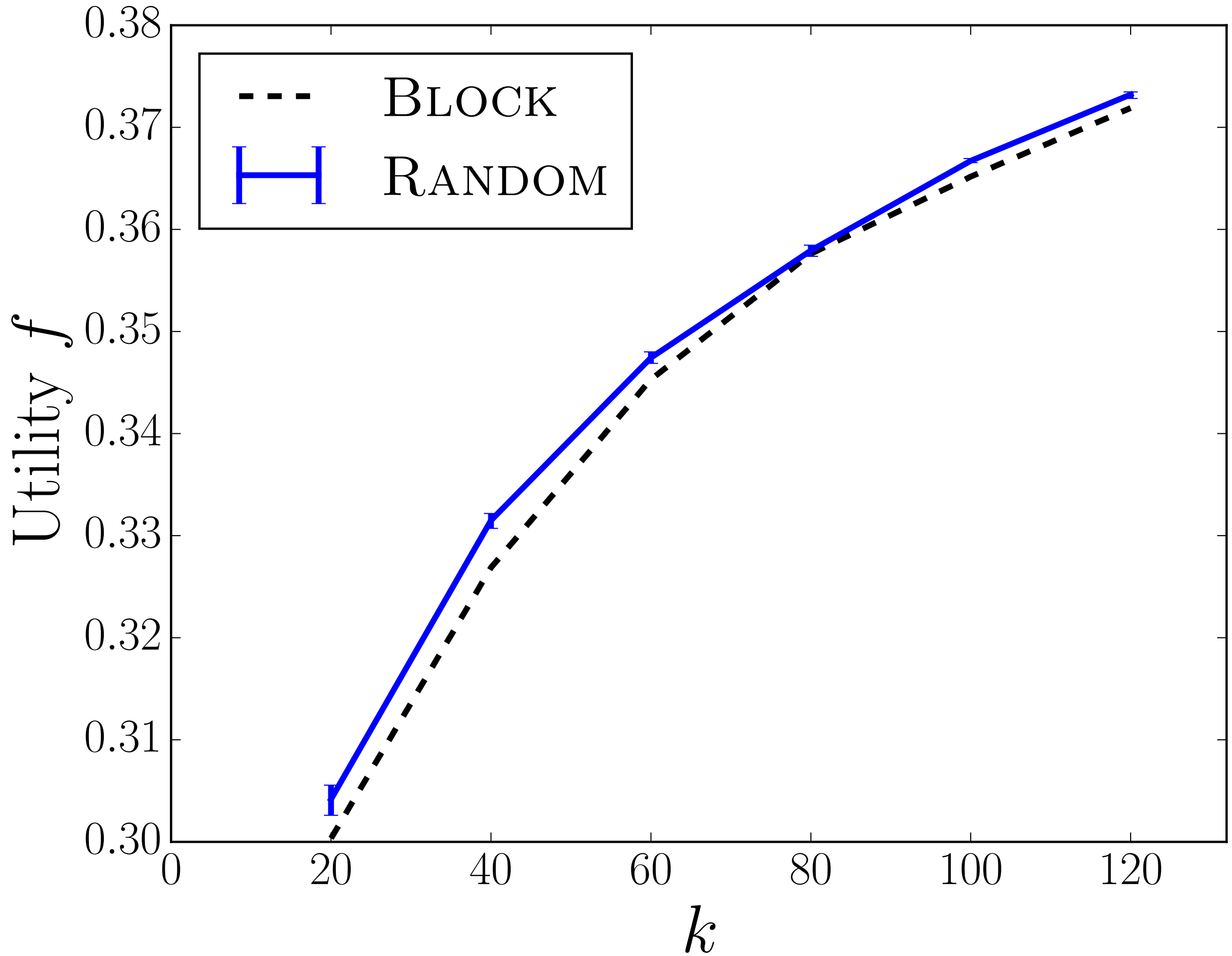
- What do we need for the proof?
  - Monotonicity and submodularity of f
  - Heredity of constraint
  - Greedy property
- The result holds in general any time greedy is an $\alpha$-approximation for a hereditary, constrained submodular maximization problem.

# Non-monotone Functions

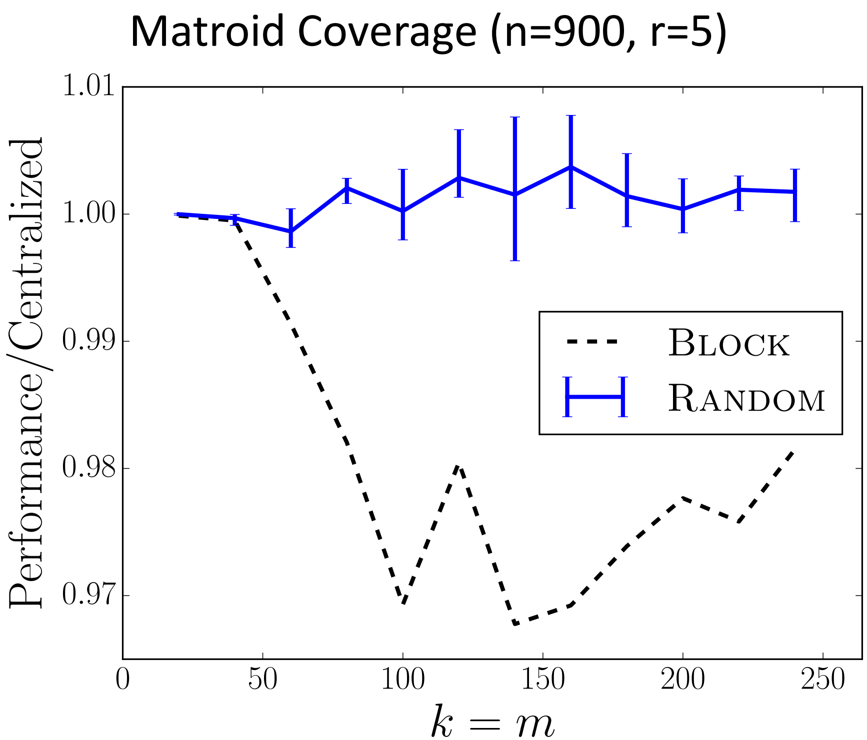
- In the first round, use Greedy on each machine
- In the second round, use any algorithm on the last machine
- We still obtain a constant factor approximation for most problems

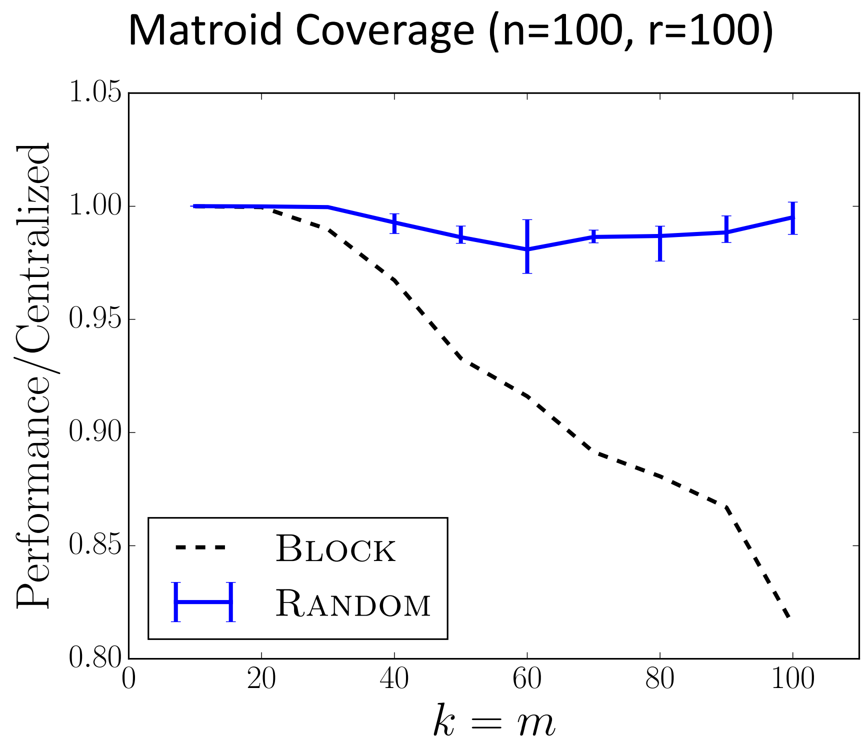
# Tiny Image Experiments

(n = 1M, m = 100)



# Matroid Coverage Experiments




It's better to distribute ellipses from each location across several machines!

# Future Directions

- Can we relax the greedy property further?
- What about non-greedy algorithms?
- Can we speed up the final round, or reduce the number machines required?
- Better approximation guarantees?