Randomized Composable Core-sets for Distributed Optimization
Vahab Mirrokni

Mainly based on joint work with:
Hossein Bateni, Aditya Bhaskara,
Hossein Esfandiari, Silvio Lattanzi,
Morteza Zadimoghaddam
Our team: Google NYC Algorithms Research Teams

Market Algorithms/Ads Optimization (search & display)

Infrastructure and Large-Scale Optimization

common expertise: online allocation problems

Large-Scale Graph Mining/Distributed Optimization

Tools: e.g. Clustering

Tools: e.g. Balanced Partitioning
Three most popular techniques applied in our tools

1. Local Algorithms: Message Passing/Label Propagation/Local Random Walks
   ○ e.g., similarity ranking via PPR etc, Connected Components
   ○ Connected components code that’s 10-50 times faster the state-of-the-art

2. Embedding/Hashing/Sketching Techniques
   ○ e.g., linear embedding for balanced graph partitioning to minimize cut
   ○ Improves the state-of-the-art by 26%. Improved flash bandwidth for search backend by 25%. Paper appeared in WSDM’16.

3. Randomized Composable Core-sets for Distributed Computation: This Talk
Agenda

- **Composable core-sets: Definitions & Applications**
  - Applications in Distributed & Streaming settings
  - Applications: Feature Selection, Diversity in Search & Recom.

- **Composable Core-sets for Four Problems: Survey**
  - Diversity Maximization (PODS’14, AAAI’17),
    Clustering (NIPS’14), **Submodular Maximization** (STOC’15),
    and Column Subset Selection (ICML’16)

- **Sketching for Coverage Problems (on arXiv)**
  - Sketching Technique
Composable Core-Sets for Distributed Optimization

Run ALG in each machine

Input Set

Machine 1
T₁

T₁

Machine 2
T₂

S₁

Machine m
Tₘ

Sₘ

Selected Items

Run ALG’ on selected items to find the final output set

Output Set

Selected Items

Selected Items

Selected Items

Selected Items

Selected Items

Selected Items

Selected Items

Selected Items

Selected Items

Selected Items

Selected Items

Selected Items
Composable Core-sets

Setup: Consider partitioning data set $T$ of elements into $m$ sets $(T_1, T_2, \ldots, T_m)$.

$$T = T_1 \cup T_2 \cup \cdots \cup T_m$$

Goal: Given a set function $f$, find a subset $S^*$ with $|S^*| \leq k$, optimizing $f(S^*)$.

$$\text{opt}(T') = f(S^*)$$

Find: small core-set $S_1 \subseteq T_1$, $S_2 \subseteq T_2$, \ldots, $S_m \subseteq T_m$ such that

$$\frac{1}{c} \text{opt}(S_1 \cup S_2 \ldots \cup S_m) \leq \text{opt}(T_1 \cup T_2 \ldots \cup T_m) \leq c \times \text{opt}(S_1 \cup S_2 \ldots \cup S_m)$$
Application in MapReduce/Distributed Computation

Run ALG in each machine

Run ALG' on selected items to find the final output set

E.g., two rounds of MapReduce
Application in Streaming Computation

- **Streaming Computation:**
  - Processing sequence of $n$ data points “on the fly”
  - Limited storage

- **Use C-composable core-set of size $k$, for example:**
  - Chunks of size $\sqrt{nk}$, thus number of chunks is $\sqrt{n/k}$
  - Compute core-set of size $k$ for each chunk
  - Total space: $k\sqrt{n/k} + \sqrt{nk} = O(\sqrt{nk})$
Overview of recent theoretical results

Need to solve (combinatorial) optimization problems on large data

1. **Diversity Maximization**, 
   - *PODS’14* by IndykMahdianMahabadiMirrokni
   - for Feature Selection in *AAAI’17* by AbbasiGhadiriMirrokniZadimoghaddam

2. **Capacitated $\ell_p$ Clustering**, *NIPS’14* by BateniBhaskaraLattanziMirrokni

3. **Submodular Maximization**, *STOC’15* by MirrokniZadimoghaddam

4. **Column Subset Selection** (Feature Selection), *ICML’16* by Alschulter et al.

5. Coverage Problems: **Submitted** by BateniEsfandiariMirrokni
Applications: Diversity & Submodular Maximization

Diverse suggestions
- Play apps
- Campaign keywords
- Search results
- News articles
- YouTube videos

Data summarization
- Feature selection

Exemplar sampling
Feature selection

We have
- Data points (docs, web pages, etc.)
- Features (topics, etc.)

**Goal**: pick a small set of “representative” features
Five Problems Considered

**General:** Find a set $S$ of $k$ items & maximize/minimize $f(S)$.

- **Diversity Maximization:** Find a set $S$ of $k$ points, and maximize the sum of pairwise distances i.e. $\max diversity(S) = \sum_{i,j \in S} dist(i, j)$.

- **Capacitated/Balanced Clustering:** Find a set $S$ of $k$ centers and cluster nodes around them while minimizing the sum of distances to $S$.

- **Coverage/Submodular Maximization:** Find a set $S$ of $k$ items. Maximize submodular function $f(S)$. Generalizing set cover.

- **Column subset selection:** Given a matrix $A$, find a set $S$ of $k$ columns.
  - Minimize $\| A - \Pi_{A[S]} A \|_F^2$
Diversity Maximization Problem

• Given: A set of \( n \) points in a metric space \((X, \text{dist})\)
• Find a set \( S \) of \( k \) points
• Goal: maximize \( \text{diversity}(S) \) i.e.

\[
\text{diversity}(S) = \text{sum of pairwise distances of points in } S.
\]

\[
\text{diversity}(S) = \sum_{i,j \in S} \text{dist}(i, j)
\]

• Background: Max Dispersion (Halldorson et al, Abbassi et al)

• Useful for feature selection, diverse candidate selection in Search, representative centers...
Core-sets for Diversity Maximization

Two rounds of MapReduce

Run LocalSearch on each machine

Run LocalSearch on selected items to find the final output set

- Arbitrary Partitioning works. Random partitioning is better.
Composable Core-set Results for Diversity Maximization

• Theorem(IndykMahabadiMahdianM.’14): The local search algorithm computes a constant-factor composable core-set for maximizing sum of pairwise distances in 2 rounds:

• Theorem(EpastoM.ZadiMoghaddam’16): A sampling+greedy algorithm computes a randomized 2-approximate composable small-size core-set for diversity maximization in one round.
  • randomized: works under random partitioning
  • small-size: size of core-set is less than k.
Distributed Clustering Problems

**Clustering:** Divide data into groups containing “nearby” points

**Minimize:**
- $k$-center: $\max_i \max_{u \in S_i} d(u, c_i)$
- $k$-means: $\sum_i \sum_{u \in S_i} d(u, c_i)^2$
- $k$-median: $\sum_i \sum_{u \in S_i} d(u, c_i)$

**Metric space $(d, X)$**

$\alpha$-approximation algorithm: cost less than $\alpha \cdot \text{OPT}$
Mapping Core-sets for Capacitated Clustering
Capacitated $\ell_p$ clustering

**Problem**: Given $n$ points in a metric space, find $k$ centers and assign points to centers, respecting capacities, to minimize $\ell_p$ norm of the distance vector.

$\rightarrow$ Generalizes balanced $k$-median, $k$-means & $k$-center.

$\rightarrow$ Objective is not minimizing cut size (cf. “balanced partitioning” in the library)

**Theorem**: For any $p$ and $k<\sqrt{n}$, distributed balanced clustering with

- approx ratio: ‘small constant’ * ‘best single machine guarantee’
- # rounds: 2
- memory: $(n/m)^2$ with $m$ machines

$\rightarrow$ Improves [BMVKV‘12] and [BEL‘13]

(Bateni,Bhaskara,Lattanzi,Mirrokni, NIPS‘14)
Empirical study for distributed clustering

Test in terms of **scalability** and **quality of solution**

Two “base” instances & subsamples
- US graph ~30M nodes
- World graph ~500M nodes

<table>
<thead>
<tr>
<th></th>
<th>Size of seq. inst</th>
<th>Increase in OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>1/300</td>
<td>1.52</td>
</tr>
<tr>
<td>World</td>
<td>1/1000</td>
<td>1.58</td>
</tr>
</tbody>
</table>

**Quality**: pessimistic analysis

**Sublinear** running time **scaling**
Submodular maximization

**Problem**: Given $k$ & submodular function $f$, find set $S$ of size $k$ that maximizes $f(S)$.

Some applications
- Data summarization
- Feature selection
- Exemplar clustering

**Special case**: “coverage maximization”: Given a family of subsets, choose a subfamily of $k$ sets, and maximize cardinality of union.
- cover various topics/meanings
- target all kinds of users
Submodular maximization

**Problem**: Given \( k \) & submodular function \( f \), find set \( S \) of size \( k \) that maximizes \( f(S) \).

Some applications

- Data summarization
- Feature selection
- Exemplar clustering

**Special case**: “coverage maximization”: Given a family of subsets, choose a subfamily of \( k \) sets, and maximize cardinality of union.

- cover various topics/meanings
- target all kinds of users

[IMMM’14] Bad News: **No deterministic** composable core-set with approx \( \leq \frac{\sqrt{k}}{\log k} \).
Submodular maximization

**Problem**: Given $k$ & submodular function $f$, find set $S$ of size $k$ that maximizes $f(S)$.

Some applications
- Data summarization
- Feature selection
- Exemplar clustering

**Special case**: “coverage maximization”: Given a family of subsets, choose a subfamily of $k$ sets, and maximize cardinality of union.
- cover various topics/meanings
- target all kinds of users

[IMMM’14] Bad News: No deterministic composable core-set with approx $\leq \frac{\sqrt{k}}{\log k}$

**Randomization** is necessary and useful:
- Send each set randomly to some machine
- Build a coreset on each machine by greedy algorithm
Randomization to the Rescue: Randomized Core-sets

Run GREEDY on each machine

Run GREEDY on selected items to find the final output set

Two rounds of MapReduce
Results for Submodular Maximization: MZ (STOC’15)

- A class of 0.33-approximate randomized composable core-sets of size $k$ for non-monotone submodular maximization. For example, Greedy Algorithm.

- Hard to go beyond $\frac{1}{2}$ approximation with size $k$. Impossible to get better than $1-\frac{1}{e}$.

- 0.58-approximate randomized composable core-set of size $4k$ for monotone $f$. Results in 0.54-approximate distributed algorithm in two rounds with linear communication complexity.

- For small-size composable core-sets of $k'$ less than $k$: $\sqrt{\frac{k'}{k}}$-approximate randomized composable core-set.
Low-Rank Approximation

Given (large) matrix $A$ in $\mathbb{R}^{m \times n}$ and target rank $k << m,n$:

$$\arg\min_{X, \text{rank}(X)=k} \| A - X \|_F^2$$

- Optimal solution: k-rank SVD
- Applications:
  - Dimensionality reduction
  - Signal denoising
  - Compression
  - ...
Column Subset Selection (CSS)

- Columns often have important meaning
- **CSS**: Low-rank matrix approximation in column space of $A$

$$\arg \min_{S \subset [n], \ |S| = k} \| A - \Pi_{A[S]} A \|_F^2$$

$A$ and $A[S]$ are matrices with $m \times n$ and $m \times k$ dimensions, respectively. The approximation $\Pi_{A[S]} A$ is a matrix with the same dimensions as $A$ but with columns selected from $A[S]$. The diagram illustrates the process of selecting a subset of columns from $A$ to form $A[S]$. The goal is to approximate $A$ with a lower rank matrix $\Pi_{A[S]} A$.
DISTGREEDY: GCSS(A, B, k) with L machines

Machine 1  Machine 2  ...  Machine L
DISTGREEDY: GCSS(A,B,k) with L machines
DISTGREEDY: GCSS(A,B,k) with L machines

Machine 1

Machine 2

Machine L

\[ S_1 = \text{GREEDY}(A, T_1, \frac{32k}{\sigma_{\text{min}}(\text{OPT}_k)}) \]

\[ S_2 = \text{GREEDY}(A, T_2, \frac{32k}{\sigma_{\text{min}}(\text{OPT}_k)}) \]

\[ S_L = \text{GREEDY}(A, T_L, \frac{32k}{\sigma_{\text{min}}(\text{OPT}_k)}) \]
DISTGREENY: GCSS(A,B,k) with L machines

Machine 1

\[ T_1 \]

Machine 2

\[ T_2 \]

\[ S_1 = \text{GREEDY} \left( A, T_1, \frac{32k}{\sigma_{\text{min}}(\text{OPT}_k)} \right) \]

Machine L

\[ T_L \]

\[ S_2 = \text{GREEDY} \left( A, T_2, \frac{32k}{\sigma_{\text{min}}(\text{OPT}_k)} \right) \]

Designated machine

\[ S_L = \text{GREEDY} \left( A, T_L, \frac{32k}{\sigma_{\text{min}}(\text{OPT}_k)} \right) \]

\[ S = \text{GREEDY} \left( A, \bigcup_{i=1}^{L} S_i, \frac{12k}{\sigma_{\text{min}}(\text{OPT}_k)} \right) \]
DISTGREEDY for column subset selection

1 round result: DISTGREEDY with \( r = O \left( \frac{k}{\sigma_{\min}(OPT)} \right) \) gives objective value \( \Omega \left( \frac{f(OPT_k)}{\kappa(OPT_k)} \right) \)

Multi-round result: \( O \left( \frac{\kappa(OPT)}{\varepsilon} \right) \) rounds gives objective value \( \Omega \left( (1 - \varepsilon) f(OPT_k) \right) \)
Empirical result for column subset selection

- Training accuracy on massive data set (news 20.binary, 15k x 100k matrix)
- Speedup over 2-phase algorithm in parentheses

<table>
<thead>
<tr>
<th>n</th>
<th>Rand</th>
<th>2-Phase</th>
<th>DISTGREEDY</th>
<th>PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>54.9</td>
<td>81.8 (1.0)</td>
<td>80.2 (72.3)</td>
<td>85.8 (1.3)</td>
</tr>
<tr>
<td>1000</td>
<td>59.2</td>
<td>84.4 (1.0)</td>
<td>82.9 (16.4)</td>
<td>88.6 (1.4)</td>
</tr>
<tr>
<td>2500</td>
<td>67.6</td>
<td>87.9 (1.0)</td>
<td>85.5 (2.4)</td>
<td>90.6 (1.7)</td>
</tr>
</tbody>
</table>

**Interesting experiment:** What if we partition more carefully and not randomly?

- **Recent observation:** If we treat each machine separately, it does not help much! Random partitioning is good even compared with more careful partitioning.
Coverage Problems

**Problems**: Given a set system \((n \text{ sets and } m \text{ elements})\),
1. “**K-coverage**”: pick \(k\) sets to max. size of union
2. “set cover”: cover *all* elements with least number of sets
3. “set cover with outliers”: cover \((1-\lambda)m\) elements with least number of sets
Coverage Problems

**Problems**: Given a set system \((n \text{ sets and } m \text{ elements})\),

1. "\(K\)-coverage": pick \(k\) sets to max. size of union
2. “set cover”: cover all elements with least number of sets
3. “set cover with outliers”: cover \((1-\lambda)m\) elements with least number of sets

Greedy Algorithm: Pick a subset with the maximum marginal coverage,
Coverage Problems

Problems: Given a set system \((n \text{ sets and } m \text{ elements})\),
1. "K-coverage": pick \(k\) sets to max. size of union
2. "set cover": cover all elements with least number of sets
3. "set cover with outliers": cover \((1-\lambda)m\) elements with least number of sets

Greedy Algorithm: Pick a subset with the maximum marginal coverage,

- 1-1/e-approx. To \(k\)-coverage, \(\log n\)-approximation for set cover...
- Goal: Achieve good fast approximation with minimum memory footprint
  - Streaming: elements arrive one by one, not sets
  - Distributed: linear communication and memory independent of the size of ground set
Submodular Maximization vs. Maximum Coverage

Coverage function is a special case of submodular function:

\[ f(R) = \text{cardinality of union of family } R \text{ of subsets} \]

\[ f(R) = \left| \bigcup_{S \in R} S \right| \]
Submodular Maximization vs. Maximum Coverage

Coverage function is a special case of submodular maximization:

\[ f(R) = \text{cardinality of union of family } R \text{ of subsets} \]

\[ f(R) = | \bigcup_{S \in R} S | \]

So problem solved?

[MirrokniZadimoghaddam STOC’15]: Randomized composable core-sets work

[Mirzasoleiman et al NIPS’14]: This method works well in Practice!
Submodular Maximization vs. Maximum Coverage

Coverage function is a special case of submodular maximization:
\[ f(R) = \text{cardinality of union of family } R \text{ of subsets} \]
\[
 f(R) = \left| \bigcup_{S \in R} S \right|
\]

So problem solved?

[MirrokniZadimoghaddam STOC’15]: Randomized composable core-sets work

[Mirzasoleiman et al NIPS’14]: This method works well in Practice!

No. This solution has several issues for coverage problems:

- **It requires** expensive oracle access **to computing cardinality of union!**
- Distributed Computation: Send whole “sets” around?
- **Streaming:** Handles set arrival model, does not handle “element” arrival model!
Why can’t we apply core-sets for submodular functions?

What if the subsets are large? Can we send a sketch of them?
Idea: Send a sketch for each set (e.g., sample of elements)

Run ALG in each machine

Sketch of subsets $T_1$

Machine 1 $T_1$

Machine 2 $T_2$

Machine m $T_m$

Family of subsets

Sketch of subsets $T_m$

Selected Items

Run ALG’ on selected items to find the final output set

Output Set

Question: Does any approximation-preserving sketch work?
Approximation-preserving sketching is not sufficient.

Idea: Use sketching to define a \((1\pm\varepsilon)\)-approx oracle to cardinality of union function?

[BateniEsfandiariMirrokni’16]:

- **Thm 1**: A \((1\pm\varepsilon)\)-approx sketch of coverage function May NOT Help
  - Given an \((1\pm\varepsilon)\)-approx oracle to coverage function, we get \(n^{0.49}\) approximation
Approximation-preserving sketching is not sufficient.

Idea: Use sketching to define a \( (1 \pm \varepsilon) \)-approx oracle to cardinality of union function?

[BateniEsfandiariMirrokni’16]:

- **Thm 1**: A \( (1 \pm \varepsilon) \)-approx sketch of coverage function May NOT Help
  - Given an \( (1 \pm \varepsilon) \)-approx oracle to coverage function, we get \( n^{0.49} \) approximation

- **Thm 2**: With some tricks, MinHash-based sketch + proper sampling WORKS
  - Sample elements not sets (different from previous coreset idea)
  - Correlation between samples (MinHash)
  - Cap degrees of elements in the sketch (reduces memory footprint)
Bipartite Graph Formulation for Coverage Problems

Bipartite graph $G(U, V, E)$
- $U$: sets
- $V$: elements
- $E$: membership

**Set cover problem**: Pick minimum number of sets that cover all elements.

**Set cover with outliers problem**: Pick minimum number of sets that cover a $1 - \lambda$ fraction of elements.

**Maximum coverage problem**: Pick $k$ sets that cover maximum number of elements.
Sketching Technique

Construction

- Dependent sampling: Assign hash values from [0,1) to elements.
- Remove any element with hash value exceeding $p$.
- Arbitrarily remove edges to have max-degree $\Delta$ for elements.

Sample parameters

1) $\Delta$ is easy to compute.
2) $\Delta$ can be found via a round of MapReduce.
Sketch: sparse subgraph with sufficient information

For instance with many sets, parallelize using core sets.

Any single-machine greedy algorithm
**Proof ingredients:**

1. Parameters are chosen to produce small sketch (indep. of size of ground set): $O(\#\text{sets})$
   - Challenge: how to choose parameters in distributed or streaming models
2. Any $\alpha$-approximation on the sketch is an $\alpha + \varepsilon$ approximation for original instance
Summary of Results for Coverage Functions

- Special case of submodular maximization
- Problems are **NP-hard** and **APX-hard**
- Greedy algorithm gives best guarantees

Good implementations (linear-time)
- Lazy greedy algorithm
- Lazier-than-lazy algorithm

**Problem:** Graph should be stored in RAM

**Our algorithm:**
- Memory $O(#\text{sets})$
- Linear-time
- Optimal approximation guarantees
- MapReduce, streaming, etc.

**GREEDY**
1) Start with empty solution
2) Until “done,”
   (a) find set with best marginal coverage, and
   (b) add it to tentative solution.
Bounds for distributed coverage problems

From [BEM’16]: 1) Space indep. of size of sets or ground set, 2) Optimal Approximation Factor, 3) Communication linear in #sets (indep. of their size), 4) small #rounds

Previous work: [39]=[CKT’11], [42]=[MZ’15], [19]=[BENW’16], [43]=[MBKK’16]

<table>
<thead>
<tr>
<th>Problem</th>
<th>Credit</th>
<th># rounds</th>
<th>Approximation</th>
<th>Load per machine</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-cover</td>
<td>[39]</td>
<td>$O(\frac{1}{\epsilon \delta} \log m)$</td>
<td>$1 - \frac{1}{\epsilon} - \epsilon$</td>
<td>$O(mkn^\delta)$</td>
<td>submodular functions</td>
</tr>
<tr>
<td>$k$-cover</td>
<td>[42]</td>
<td>2</td>
<td>0.54</td>
<td>max$(mk^2, mn/k)$</td>
<td>submodular functions</td>
</tr>
<tr>
<td>$k$-cover</td>
<td>[19]</td>
<td>$\frac{1}{\epsilon}$</td>
<td>$1 - \frac{1}{\epsilon} - \epsilon$</td>
<td>max$(mk^2, mn/k)$</td>
<td>submodular functions</td>
</tr>
<tr>
<td>$k$-cover</td>
<td>Here</td>
<td>3</td>
<td>$1 - \frac{1}{\epsilon} - \epsilon$</td>
<td>$\tilde{O}(n + m)$</td>
<td>-</td>
</tr>
<tr>
<td>Set cover w outliers</td>
<td>Here</td>
<td>3</td>
<td>$(1 + \epsilon) \log \frac{1}{\lambda}$</td>
<td>$\tilde{O}(n + m)$</td>
<td>-</td>
</tr>
<tr>
<td>Set cover</td>
<td>[43]</td>
<td>$\log(nm)$</td>
<td>$(1 + \epsilon) \log n$</td>
<td>$\Omega(mn^{1-\epsilon})$</td>
<td>Submodular cover</td>
</tr>
<tr>
<td>Set cover</td>
<td>Here</td>
<td>$r$</td>
<td>$(1 + \epsilon) \log n$</td>
<td>$\tilde{O}(nm^{O(\frac{1}{r})} + m)$</td>
<td>-</td>
</tr>
</tbody>
</table>
Bounds for streaming coverage problems

From [BEM’16]: 1) Space indep. of size of ground set, 2) Optimal Approximation Factor, 3) “Edge” vs “set” arrival

Previous work: [14]=[CW’15], [22]=[DIMV’14], [24]=[ER’14], [31]=[IMV’15], [49]=[SG’09]

<table>
<thead>
<tr>
<th>Problem</th>
<th>Credit</th>
<th># passes</th>
<th>Approximation</th>
<th>Space</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-cover</td>
<td>[49]</td>
<td>1</td>
<td>1/4</td>
<td>$\tilde{O}(m)$</td>
<td>set</td>
</tr>
<tr>
<td>k-cover</td>
<td>Here</td>
<td>1</td>
<td>$1 - 1/e - \varepsilon$</td>
<td>$\tilde{O}(n)$</td>
<td>edge</td>
</tr>
<tr>
<td>Set cover w outliers</td>
<td>[24, 14]</td>
<td>p</td>
<td>$O(\min(n^{p+1}, e^{-1/p}))$</td>
<td>$\tilde{O}(m)$</td>
<td>set</td>
</tr>
<tr>
<td>Set cover w outliers</td>
<td>Here</td>
<td>1</td>
<td>$(1 + \varepsilon)\log\frac{1}{\lambda}$</td>
<td>$\tilde{O}_\lambda(n)$</td>
<td>edge</td>
</tr>
<tr>
<td>Set cover</td>
<td>[14, 49]</td>
<td>p</td>
<td>$(p+1)n^{\frac{1}{p+1}}$</td>
<td>$\tilde{O}(m)$</td>
<td>set</td>
</tr>
<tr>
<td>Set cover</td>
<td>[22]</td>
<td>$4^k$</td>
<td>$4^k\log n$</td>
<td>$\tilde{O}(nm^{\frac{1}{k}})$</td>
<td>set</td>
</tr>
<tr>
<td>Set cover$^1$</td>
<td>[31]</td>
<td>p</td>
<td>$O(p\log n)$</td>
<td>$\tilde{O}(nm^{O(\frac{1}{p})})$</td>
<td>set</td>
</tr>
<tr>
<td>Set cover</td>
<td>Here</td>
<td>p</td>
<td>$(1 + \varepsilon)\log n$</td>
<td>$\tilde{O}(nm^{O(\frac{1}{p})} + m)$</td>
<td>edge</td>
</tr>
</tbody>
</table>
Empirical Study

Public datasets

- Social networks
- Bags of words
- Contribution graphs
- Planted instances

- Very small sketches (0.01–5%) suffice for obtaining good approximations (95+%).

- Without core sets, can handle in <1h XXXB edges or elements.
**Feature Selection (ongoing)**

**Goal:** Pick $k$ “representative” features

Based on composable core sets

<table>
<thead>
<tr>
<th>$k$</th>
<th>Random clusters</th>
<th>Best cluster method</th>
<th>Set cover (pairs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.8538</td>
<td>0.851</td>
<td>0.862</td>
</tr>
<tr>
<td>1000</td>
<td>0.8864</td>
<td>0.8912</td>
<td>0.8936</td>
</tr>
<tr>
<td>2500</td>
<td><strong>0.9236</strong></td>
<td>0.9234</td>
<td>0.9118</td>
</tr>
</tbody>
</table>

1) Pick features that cover all entities
2) Pick features that cover many pairs (or triples, etc.) of entities
Summary: Distributed Algorithms for Five Problems

Define on a metric space & composable core-sets apply.

1. Diversity Maximization,
   - PODS’14 by IndykMahdianMahabadiM.
   - for Feature Selection in AAAI’17 by AbbasiGhadiriMirrokniZadimoghaddam

2. Capacitated $\ell_p$ Clustering, NIPS’14 by BateniBhaskaraLattanziM.

Beyond Metric Spaces. Only *Randomized* partitioning apply.

3. Submodular Maximization, STOC’15 by M. Zadimoghaddam
4. Feature Selection (Column Subset Selection), ICML’16 by Alschulter et al.

Needs adaptive sampling/sketching techniques

5. Coverage Problems: by BateniEsfandiarM
Our team: Google NYC Algorithms Research Team

Recently released external team website: research.google.com/teams/nycalg/

Market Algorithms/Ads Optimization (search & display)

Common expertise: online allocation problems

Infrastructure and Large-Scale Optimization

Large-Scale Graph Mining/Distributed Optimization

Tools: e.g. Clustering

Tools: e.g. Balanced Partitioning
THANK YOU

mirrokni@google.com
Local Search for Diversity Maximization [KDD’13]

- Used for sum of pairwise distances
- Algorithm [Abbasi, Mirrokni, Thakur]
  - Initialize $S$ with an arbitrary set of $k$ points which contains the two farthest points
  - While there exists a swap that improves diversity by a factor of $\left(1 + \frac{\epsilon}{n}\right)$
    » Perform the swap
- For Remote-Clique
  - Number of rounds: $\log_{\left(1+\frac{\epsilon}{n}\right)} k^2 = O\left(\frac{n}{\epsilon} \log k\right)$
  - Approximation factor is constant.