Randomized Composable Core-sets for Distributed Optimization

Vahab Mirrokni

Algorithms Research Group,
Google Research, New York

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)

common expertise:
online allocation problems

Infrastructure and
Large-Scale
Optimization

Tools:
e.g. Clustering

**Large-Scale Graph
Mining/Distributed
Optimization**

Tools:
e.g. Balanced Partitioning

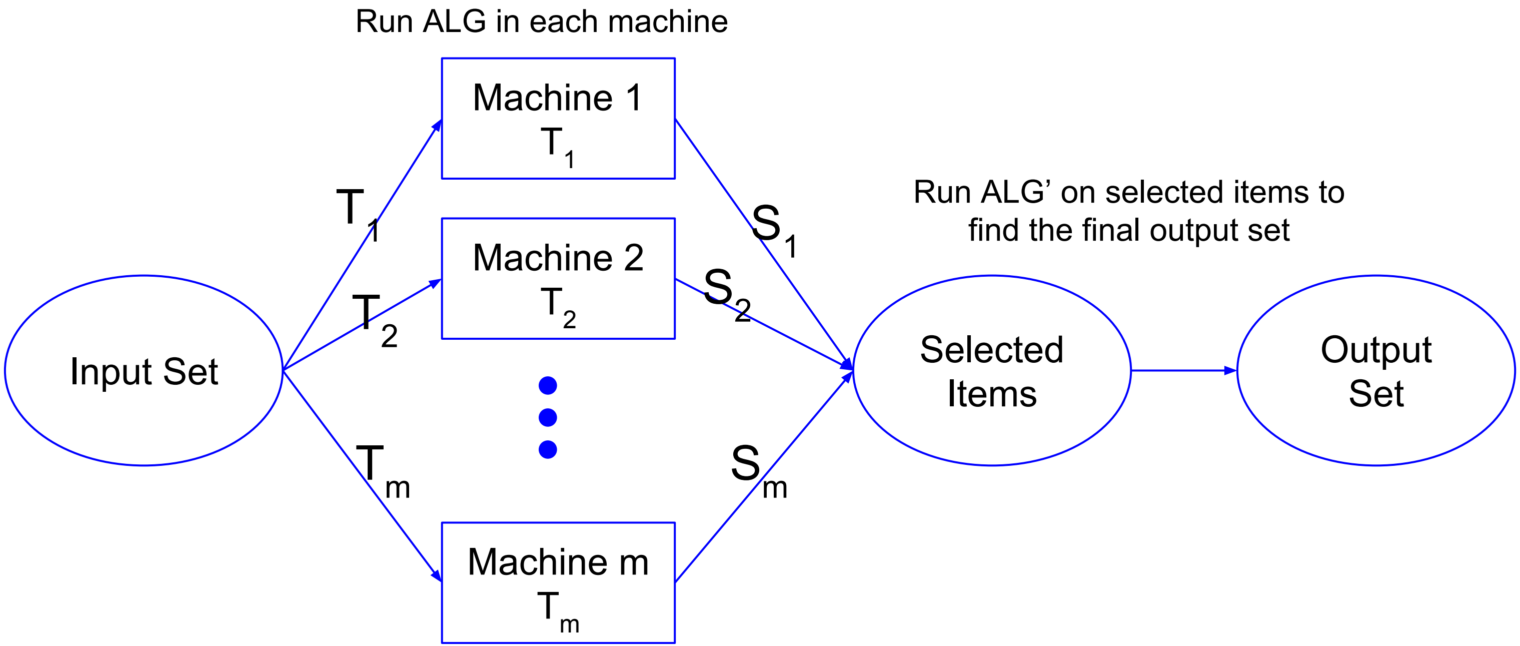
Three most popular techniques applied in our tools

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



Composable Core-sets

Setup: Consider partitioning data set *T* of elements into *m* sets *(T_1,T_2,...,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*)*.

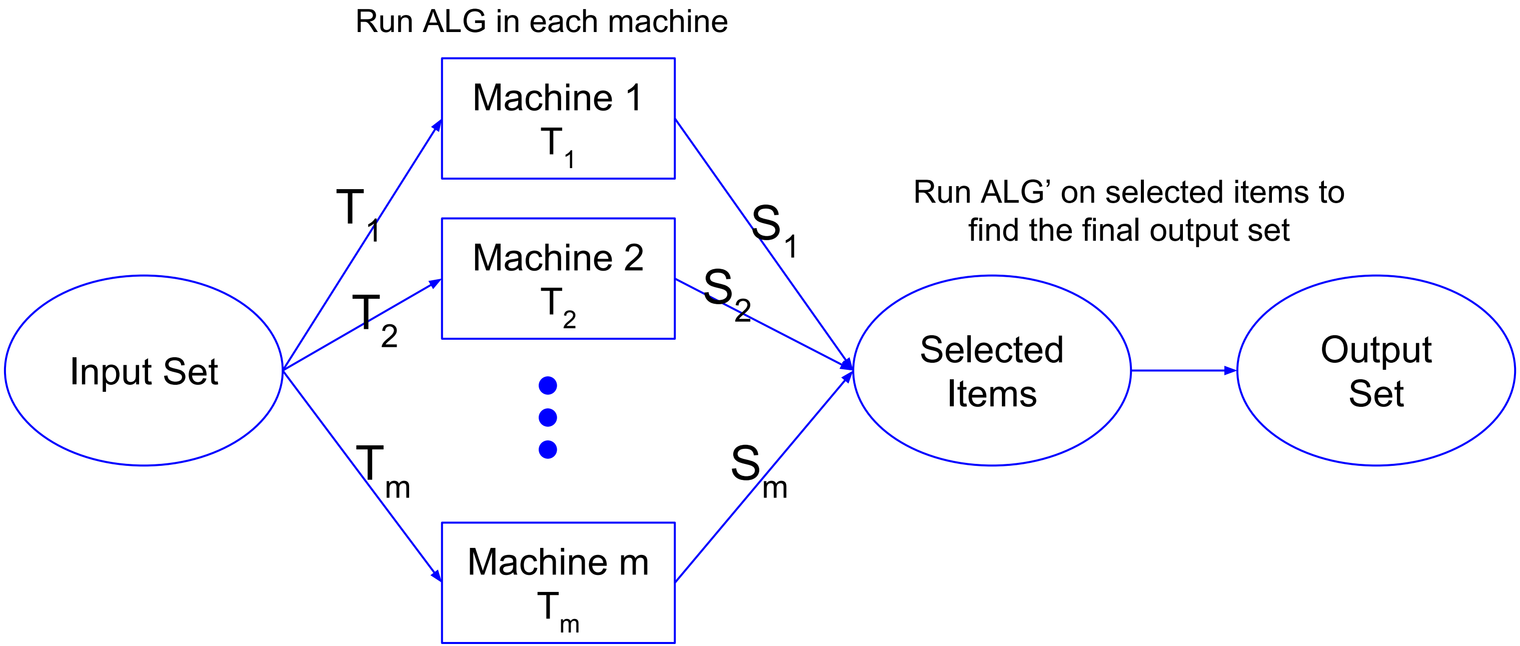
$$opt(T) = f(S^*)$$

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

optimum solution in union of core-sets approximates the optimum solution of T

$$\frac{1}{c}\mathrm{opt}(S_1 \cup S_2 \ldots \cup S_m) \leq \mathrm{opt}(T_1 \cup T_2 \ldots \cup T_m) \leq c \times \mathrm{opt}(S_1 \cup S_2 \ldots \cup S_m)$$

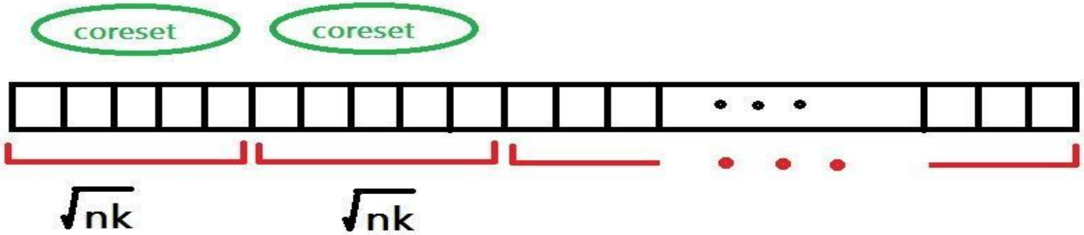
Application in MapReduce/Distributed Computation



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: $k\sqrt{n/k} + \sqrt{nk} = O(\sqrt{nk})$



Google

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 ℓ_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

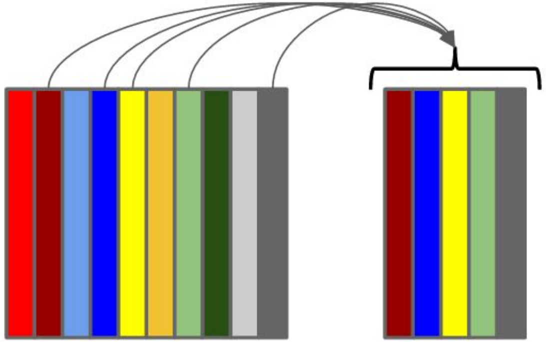
Exemplar sampling

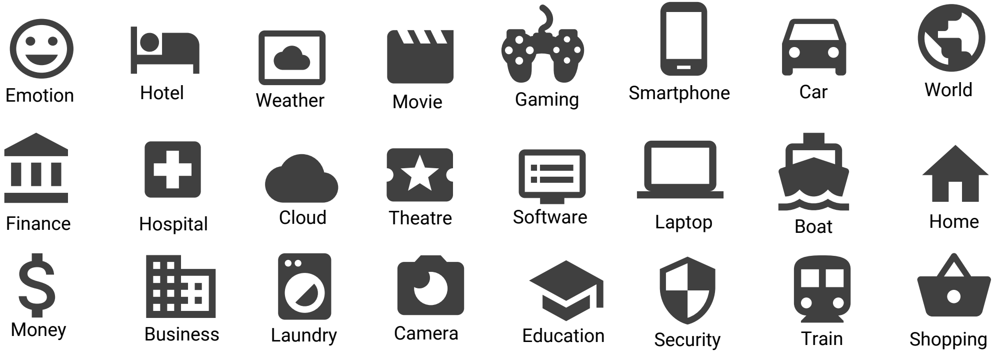
Feature selection

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 $||A - \Pi_{A[S]} A||_{\mathcal{F}}^2$

Diversity Maximization Problem

- Given: A set of *n* points in a metric space *(X, dist)*
- Find a set *S* of *k* points
- Goal: maximize *diversity(S)* i.e.

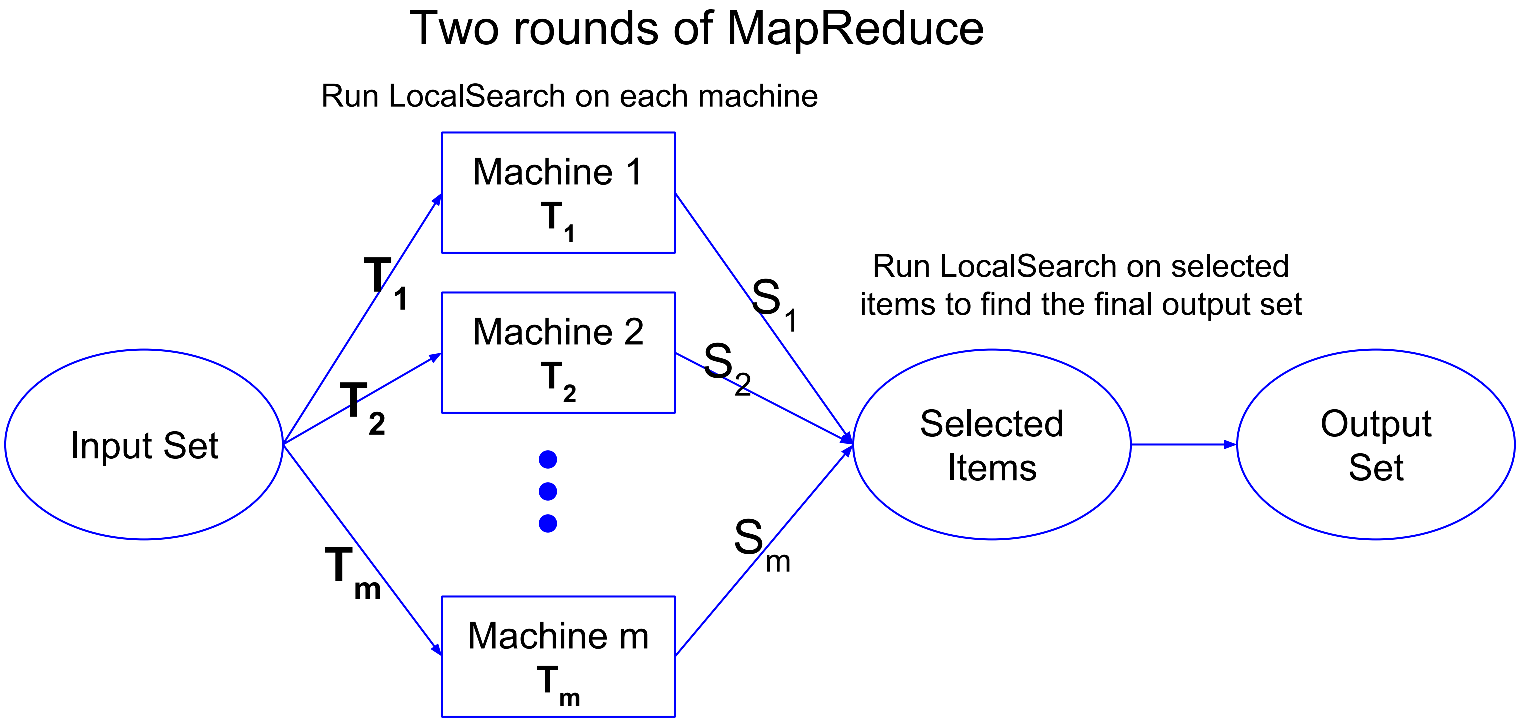
  diversity(S) **= sum of pairwise distances of points in *S*.**

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

- 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



- Arbitrary Partitioning works. Random partitioning is better.

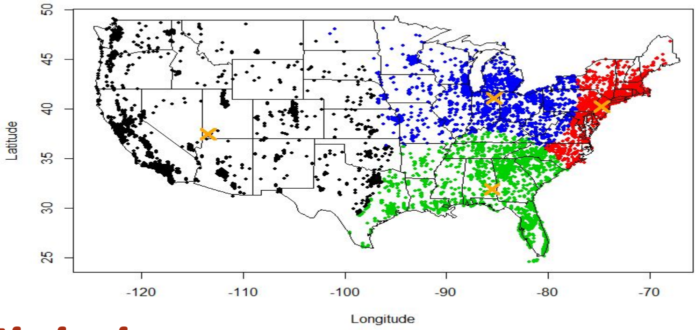
Composable Core-set Results for Diversity Maximization

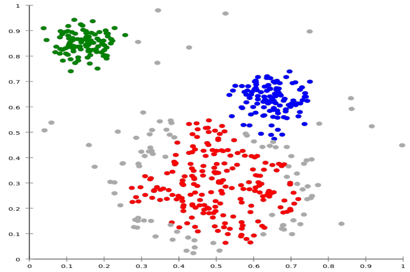
- Theorem(IndykMahabadiMahdianM.'14): The local search algorithm computes a *constant-factor* composable core-set for maximizing *sum of pairwise distances in **2 rounds**:*

- 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 : $\max_i \max_{u \in S_i} d(u, c_i)$

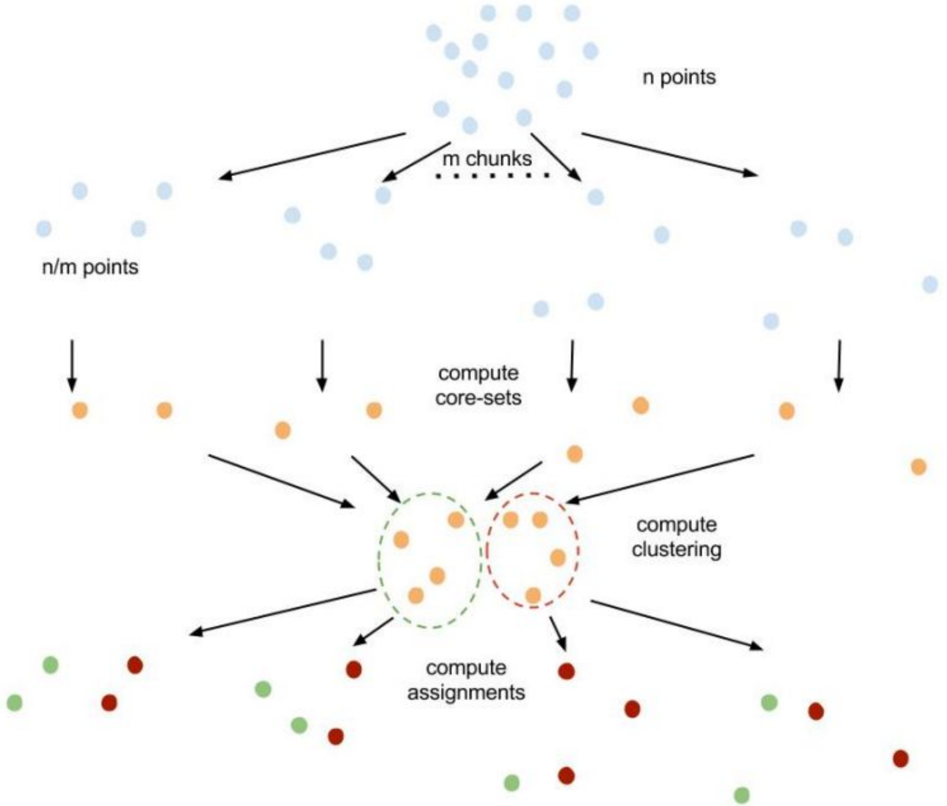
k-means : $\sum_i \sum_{u \in S_i} d(u, c_i)^2$

k-median : $\sum_i \sum_{u \in S_i} d(u, c_i)$

Metric space *(d, X)*

α-approximation algorithm: cost less than α*OPT

Mapping Core-sets for Capacitated Clustering



Google™

Capacitated *ℓp* clustering

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

➔ Generalizes *balanced* ***k***-median, ***k***-means & ***k***-center.
➔ Objective is *not* minimizing cut size (cf. "balanced partitioning" in the library)

Theorem: For any ***p*** and ***k<√n***, distributed balanced clustering with

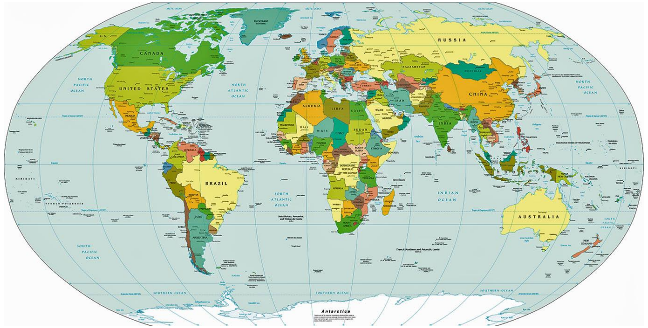
- approx ratio: '*small constant*' * '*best single machine guarantee*'
- # rounds: ***2***
- memory: **(*n*/*m*)²** with ***m*** machines

➔ Improves [**BMVKV'12**] and [**BEL'13**]

(Bateni,Bhaskara,Lattanzi,Mirrokni, NIPS'14)

Google

Empirical study for distributed clustering



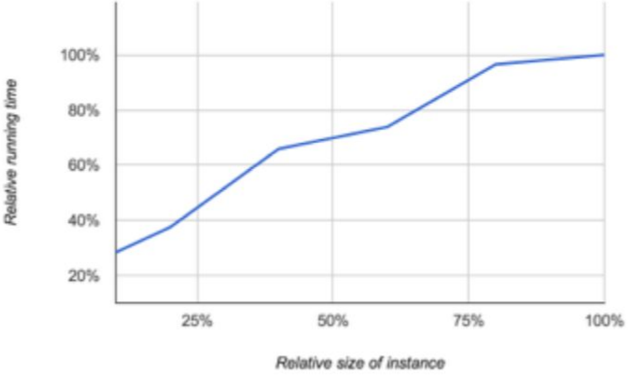
Test in terms of scalability and quality of solution

Two "base" instances & subsamples

- US graph ~30M nodes
- World graph ~500M nodes

	Size of seq. inst	Increase in OPT
US	1/300	1.52
World	1/1000	1.58

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

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

Some applications

- Data summarization
- Feature selection
- Exemplar clustering

<u>Special case</u>: "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 ≤ $\frac{\sqrt{k}}{\log k}$

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.

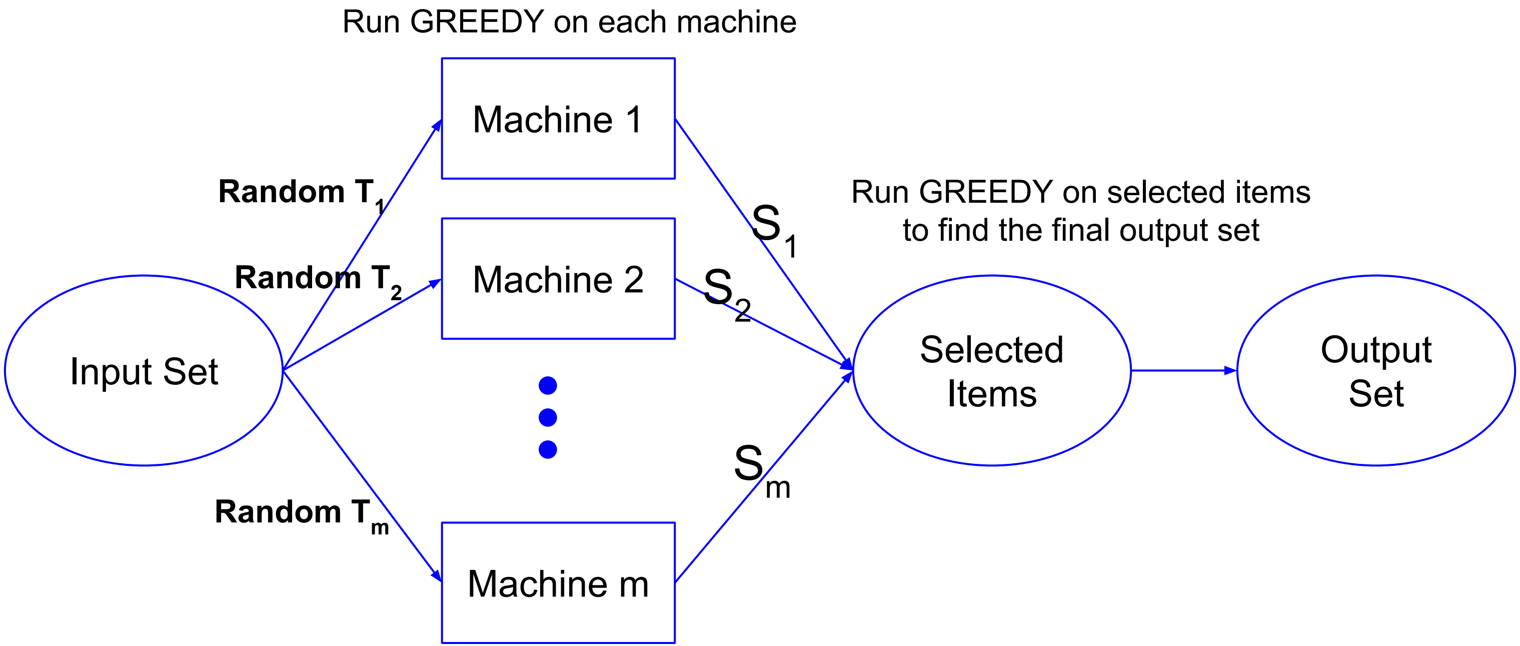
- cover various topics/meanings
- target all kinds of users

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

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



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 ½ approximation with size k. Impossible to get better than 1-1/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{k'/k}*-approximate randomized composable core-set.

Low-Rank Approximation

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

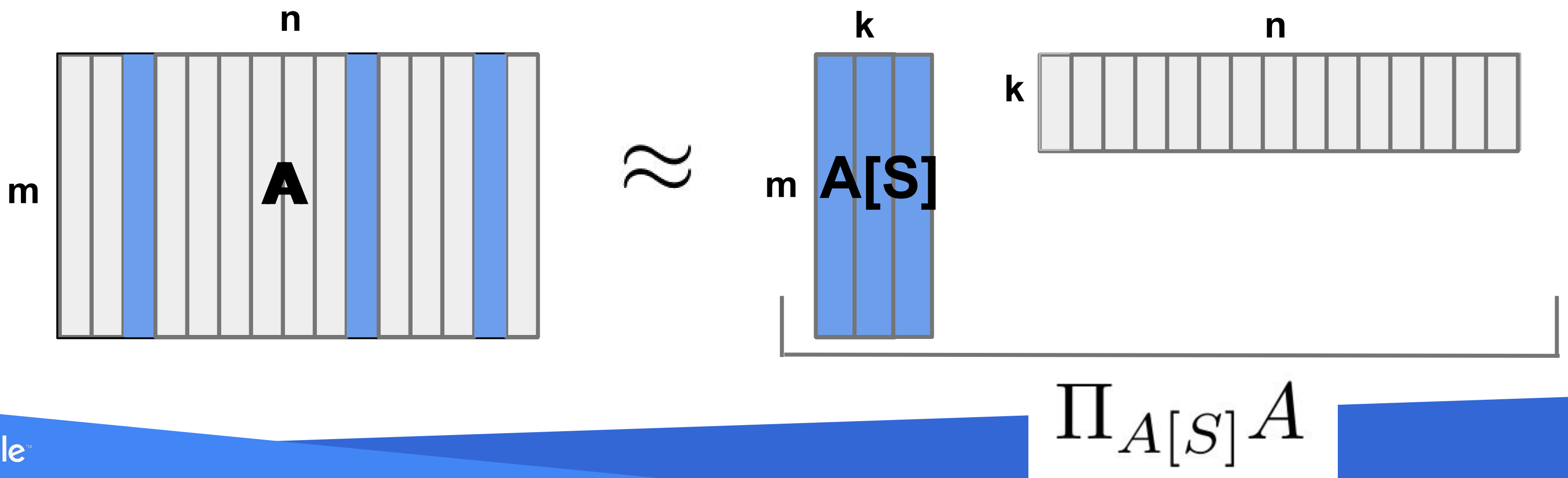
$$\underset{X,\ \mathrm{rank}(X)=k}{\arg\min} \quad \|A - X\|_F^2$$

- Optimal solution: k-rank SVD
- Applications:
  - Dimensionality reduction
  - Signal denoising
  - Compression
  - ...

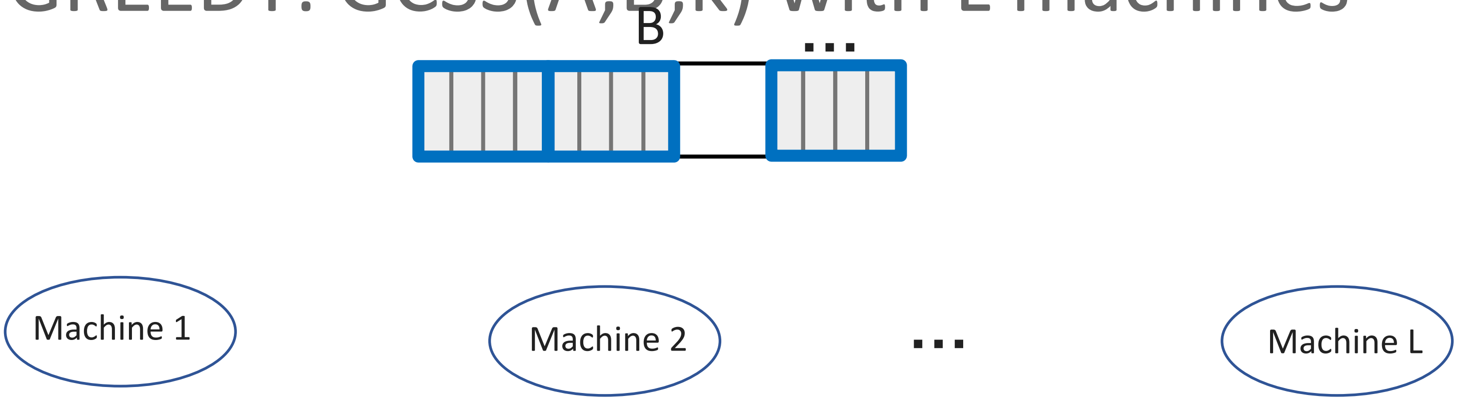
Column Subset Selection (CSS)

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

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

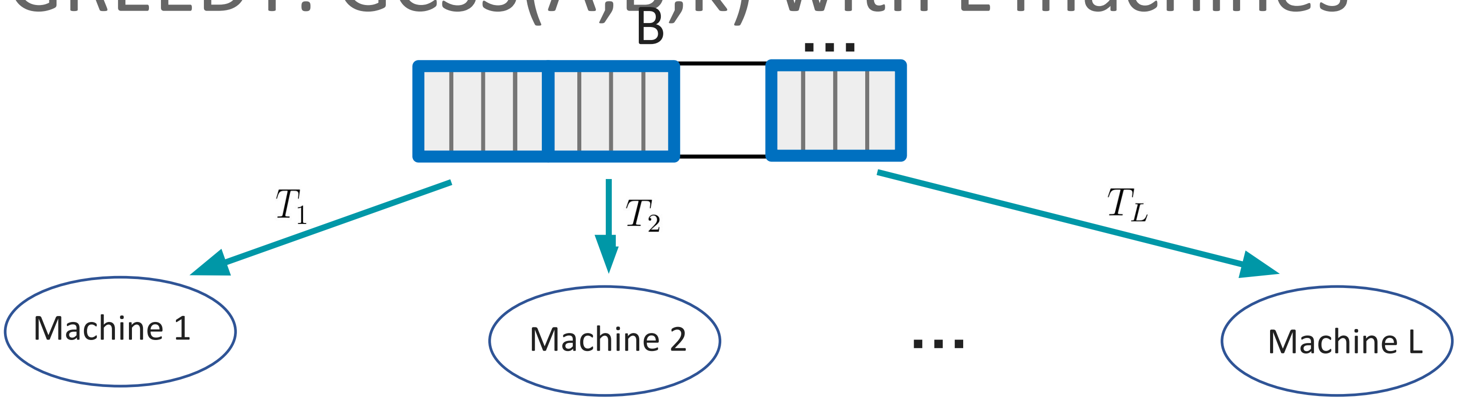


DISTGREEDY: GCSS(A,B,k) with L machines

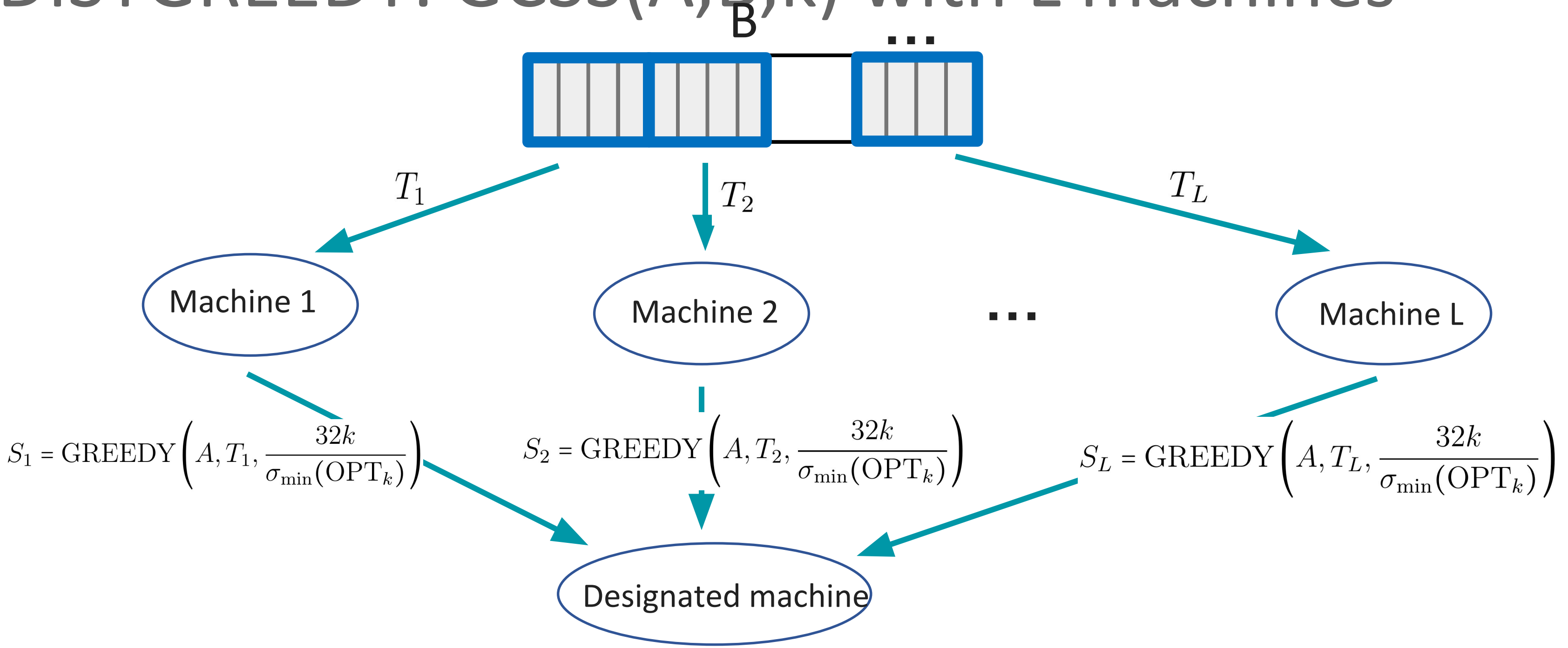


Google

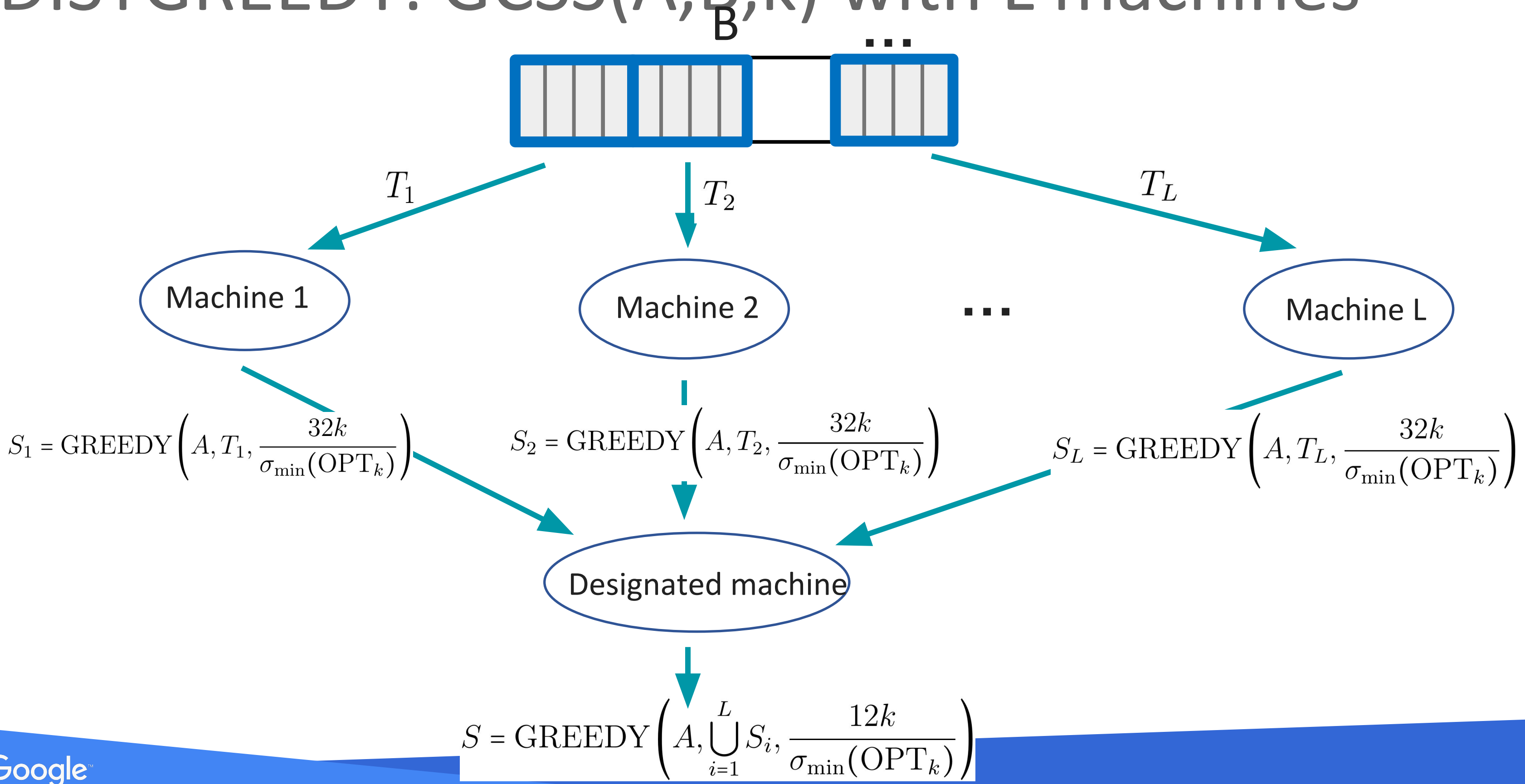
DISTGREEDY: GCSS(A,B,k) with L machines



DISTGREEDY: GCSS(A,B,k) with L machines



DISTGREEDY: GCSS(A,B,k) with L machines



DISTGREEDY for column subset selection

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

Condition number $\frac{\sigma_{\max}(OPT_k)}{\sigma_{\min}(OPT_k)}$

Multi-round result: $O(\frac{\kappa(OPT)}{\varepsilon})$ rounds gives objective value $\Omega\Big((1-\varepsilon)f(OPT_k)\Big)$

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

n	Rand	2-Phase	DISTGREEDY	PCA
500	54.9	81.8 (1.0)	80.2 (72.3)	85.8 (1.3)
1000	59.2	84.4 (1.0)	82.9 (16.4)	88.6 (1.4)
2500	67.6	87.9 (1.0)	85.5 (2.4)	90.6 (1.7)

- **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.

Google

Coverage Problems

Problems: Given a set system (***n*** sets and ***m*** 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-λ)*m*** elements with least number of sets

Coverage Problems

<u>**Problems**</u>: Given a set system (***n*** sets and ***m*** 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-λ)*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*** sets and ***m*** 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-λ)*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) = cardinality of union of family R of subsets

$$f(R) = |\cup_{S \in R} S|$$

Google™

Submodular Maximization vs. Maximum Coverage

Coverage function is a special case of submodular maximization:

f(R) = cardinality of union of family R of subsets

$$f(R) = |\cup_{S \in R} S|$$

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) = cardinality of union of family R of subsets

$$f(R) = |\cup_{S \in R} S|$$

So problem solved?

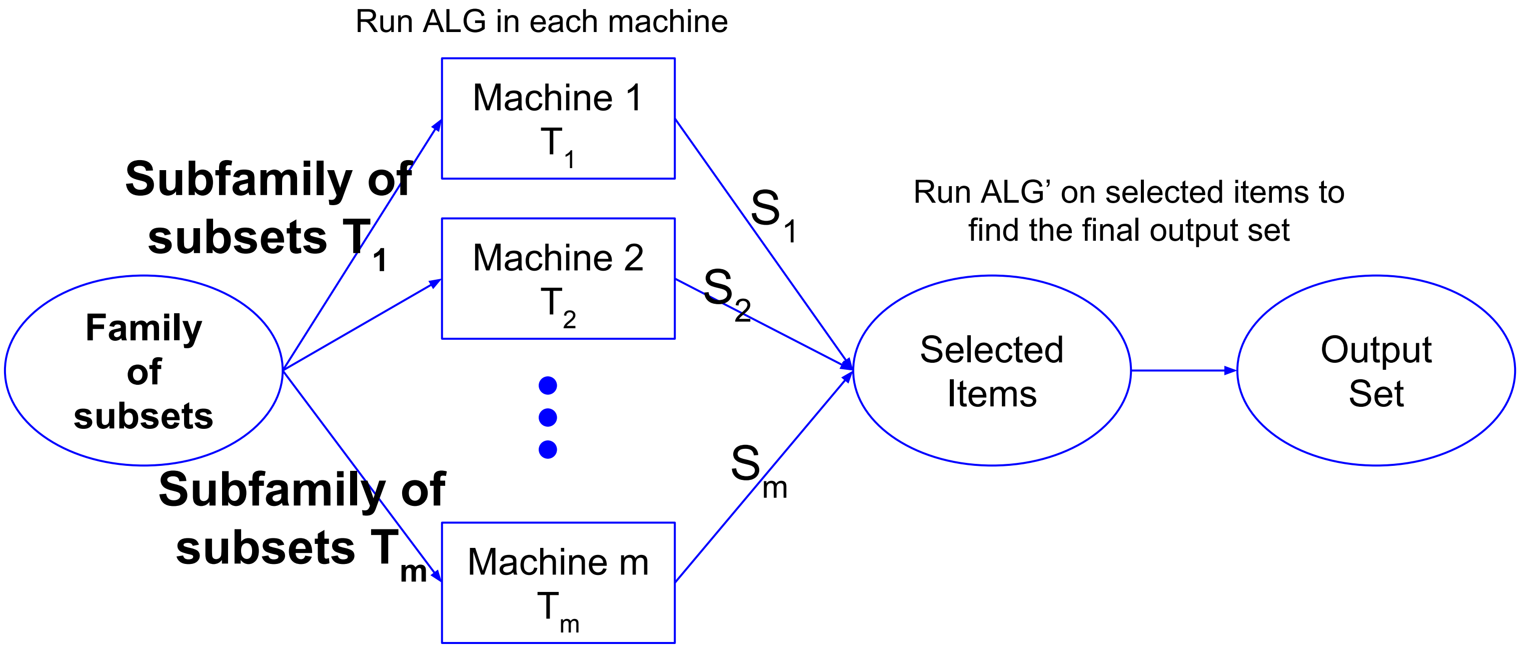
[**MirrokniZadimoghaddam STOC'15**]: Randomized composable core-sets work

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

No. This solution has several issues for coverage problems:

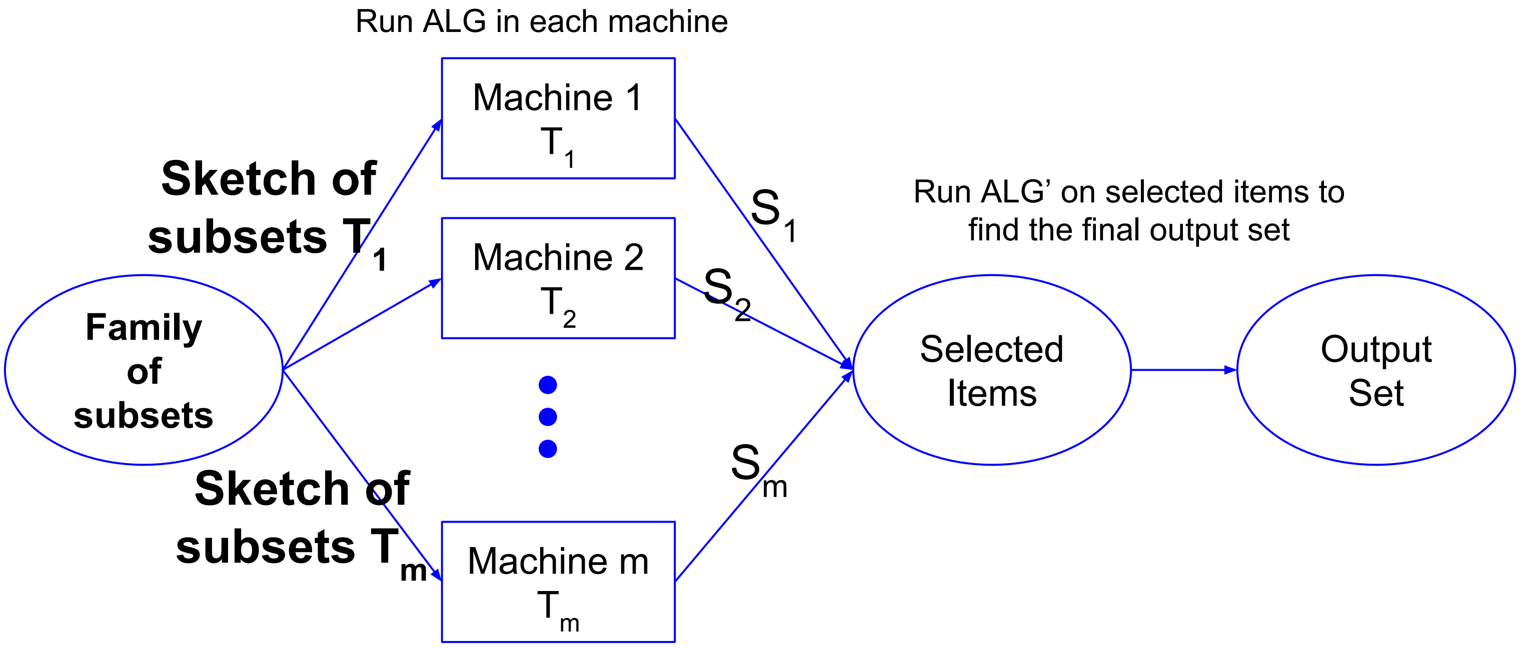
- *It requires **expensive oracle access** to computing cardinality of union!*
- *Distributed Computation: Send whole "sets" around ?*
- *Streaming: Handles set arrival model, does not handle "element" arrival model!*

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)



Question: Does any approximation-preserving sketch work?

Approximation-preserving sketching is not sufficient.

Idea: Use sketching to define ***a (1±ε)-approx oracle to cardinality of union function***?

[**BateniEsfandiariMirrokni'16**]:

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

Google

Approximation-preserving sketching is not sufficient.

Idea: Use sketching to define ***a (1±ε)-approx oracle to cardinality of union function***?

[**BateniEsfandiariMirrokni'16**]:

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

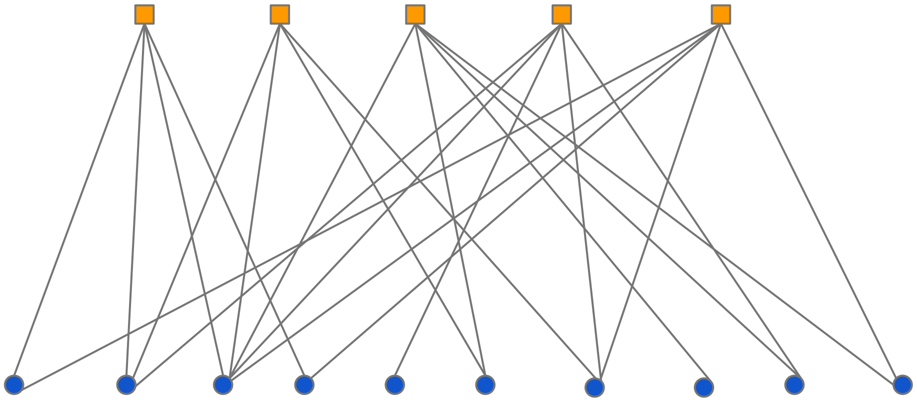
Bipartite graph ***G*(*U*, *V*, *E*)**

- ***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 - λ** fraction of elements.

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



Sketching Technique

Construction

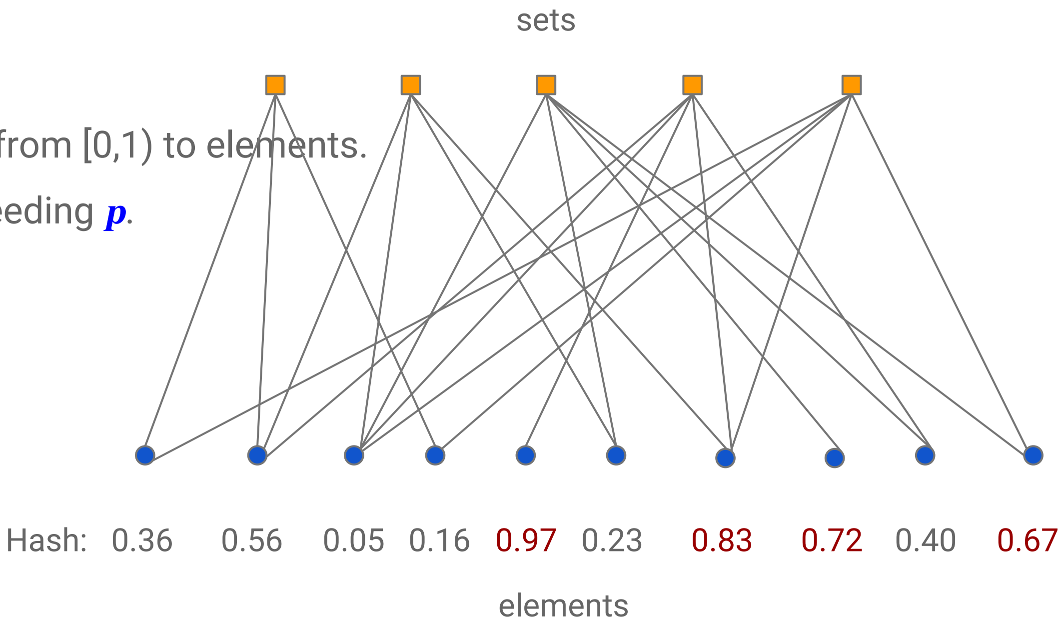
- Dependent sampling: Assign hash values from [0,1) to elements.
- Remove any element with hash value exceeding p.
- Arbitrarily remove edges to have max-degree Δ for elements.

Sample parameters

1) $p = 0.6$
2) $\Delta = 2$

Parameters

1) Δ is easy to compute.
2) p can be found via a round of MapReduce.



Approach

Build graph → Sketch construction → Core-set method → Final greedy → Extract results

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(#sets)
    - Challenge: how to choose parameters in distributed or streaming models
2. Any α-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(#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]

Problem	Credit	# rounds	Approximation	Load per machine	Comment
k-cover	[39]	$O(\frac{1}{\varepsilon\delta}\log m)$	$1-\frac{1}{e}-\varepsilon$	$O(mkn^{\delta})$	submodular functions
k-cover	[42]	2	0.54	$\max(mk^2, mn/k)$	submodular functions
k-cover	[19]	$\frac{1}{\varepsilon}$	$1-\frac{1}{e}-\varepsilon$	$\frac{\max(mk^2, mn/k)}{\varepsilon}$	submodular functions
k-cover	Here	3	$1-\frac{1}{e}-\varepsilon$	$\tilde{O}(n+m)$	-
Set cover w outliers	Here	3	$(1+\varepsilon)\log\frac{1}{\lambda}$	$\tilde{O}(n+m)$	-
Set cover	[43]	$\log(nm)$	$(1+\varepsilon)\log n$	$\Omega(mn^{1-\varepsilon})$	Submodular cover
Set cover	Here	r	$(1+\varepsilon)\log n$	$\tilde{O}(nm^{O(\frac{1}{r})}+m)$	-

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]

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

Empirical Study

Public datasets

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

Name	Type	$\|S\|$	$\|\mathcal{E}\|$	$\|E\|$
livejournal-3	dominating set	3,997,962	3,997,962	72,803,204,325
livejournal-2	dominating set	3,997,962	3,997,962	3,377,182,611
dblp-3	dominating set	317,080	317,080	333,505,724
dblp-2	dominating set	317,080	317,080	27,437,914
gutenberg	bag of words	41,716	99,949,091	1,068,977,156
s-gutenberg	bag of words	925	10,620,424	27,337,479
reuters	bag of words	199,328	138,922	15,334,605
planted-A	planted	10,100	10,000	1,220,000
planted-B	planted	100,100	1,000,000	1,201,100,000
planted-C	planted	100,500	10,000,000	2,410,100,000
planted-D	planted	101,000	10,000,000	1,210,100,000
wiki-main	contribution graph	2,953,425	10,619,081	75,151,304
wiki-talk	contribution graph	1,736,343	1,017,617	7,299,920

Instance	Footprint	Quality
wiki-main	0.06%	94.4%
wiki-main	2.4%	99.5%
wiki-main	7.7%	99.9%
wiki-talk	1.5%	99.2%
planted-A	8.2%	96%

Instance	Footprint	Quality
reuters	10%	96%
dblp-2	1.7%	92%
dblp-2	3.1%	96%
reuters	1.2%	87%
reuters	3.6%	92%

– 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)

<u>Goal</u>: Pick k “representative” features

$$
\begin{array}{cc}
 & \text{features} \\
\text{entities} & \begin{bmatrix}
+ & + & + & + & + & + & + & + & + & + \\
+ & + & + & + & - & - & - & - & - & - \\
+ & - & - & - & + & + & + & - & - & - \\
- & + & - & - & + & - & - & + & + & - \\
- & - & + & - & - & + & - & + & - & + \\
- & - & - & + & - & - & + & - & + & +
\end{bmatrix}
\end{array}
$$

Based on composable core sets

k	Random clusters	Best cluster method	Set cover (pairs)
500	0.8538	0.851	0.862
1000	0.8864	0.8912	0.8936
2500	0.9236	0.9234	0.9118

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 ℓ_p Clustering, NIPS'14 by BateniBhaskaraLattanziM.

Beyond Metric Spaces. Only *Randomized* partitioning apply.

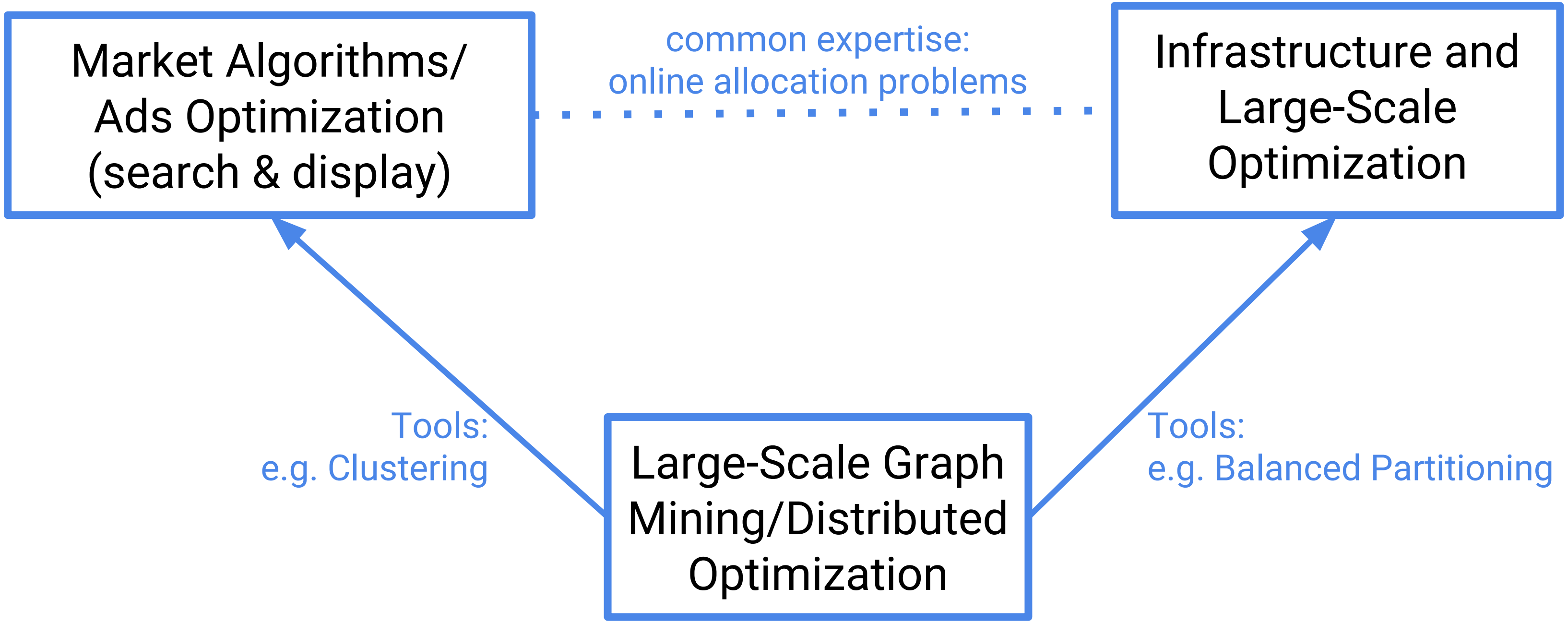
3. Submodular Maximization, STOC'15 by M. Zadimoghaddam
4. Feature Selection (Column Subset Selection), ICML'16 by Alschulter et al.

Needs adaptive sampling/sketching techniques

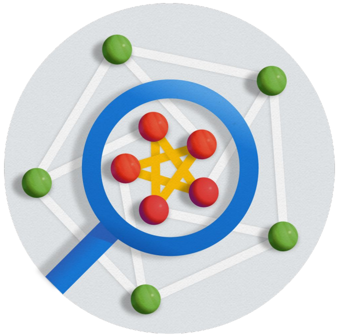
5. Coverage Problems: by BateniEsfandiariM

Our team: Google NYC Algorithms Research Team

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



THANK YOU



mirrokni@google.com

Google

Ωmega

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: $\log_{\{1+\frac{\epsilon}{n}\}} k^2 = O(\frac{n}{\epsilon} \log k)$
  - Approximation factor is constant.