Streaming Maximum Coverage

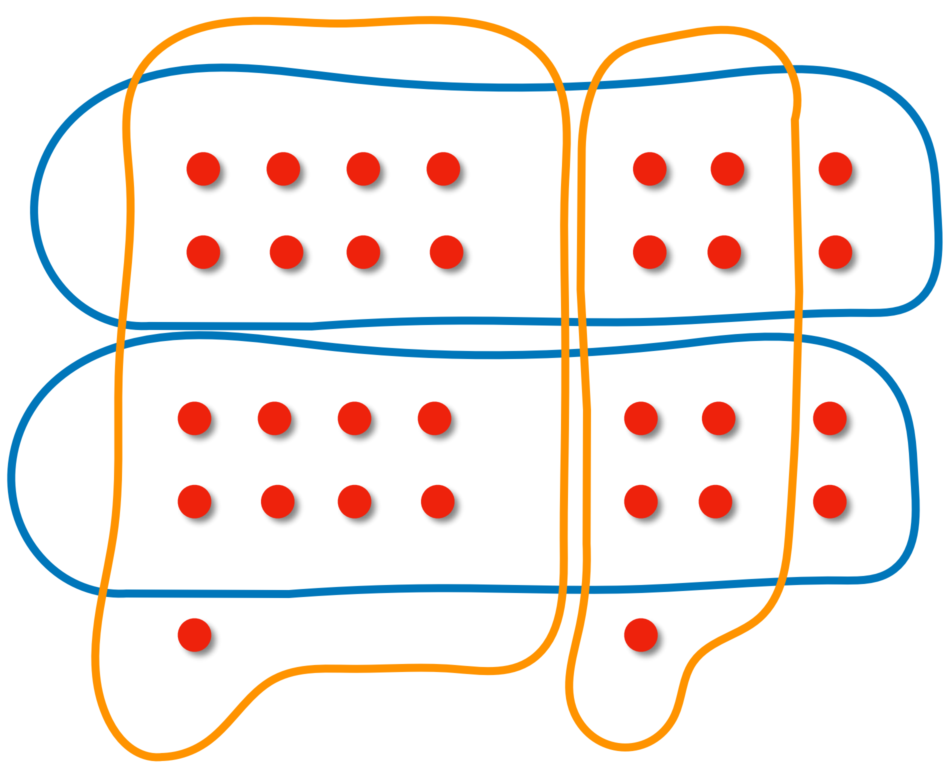
Hoa Vu
University of Massachusetts, Amherst
(joint work with Andrew McGregor)

Max-Cover

Input: **m** subsets of U = {1,2,...,**n**}

Goal: find **k** sets with maximum coverage

k=2

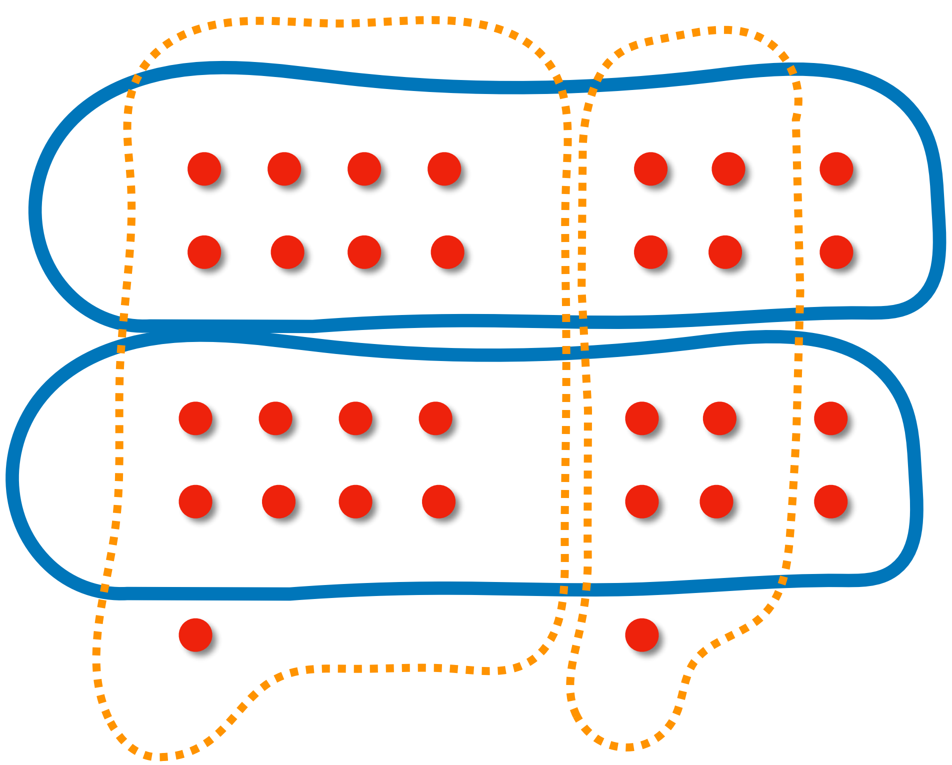


Max-Cover

Input: **m** subsets of U = {1,2,...,**n**}

Goal: find **k** sets with maximum coverage

k=2



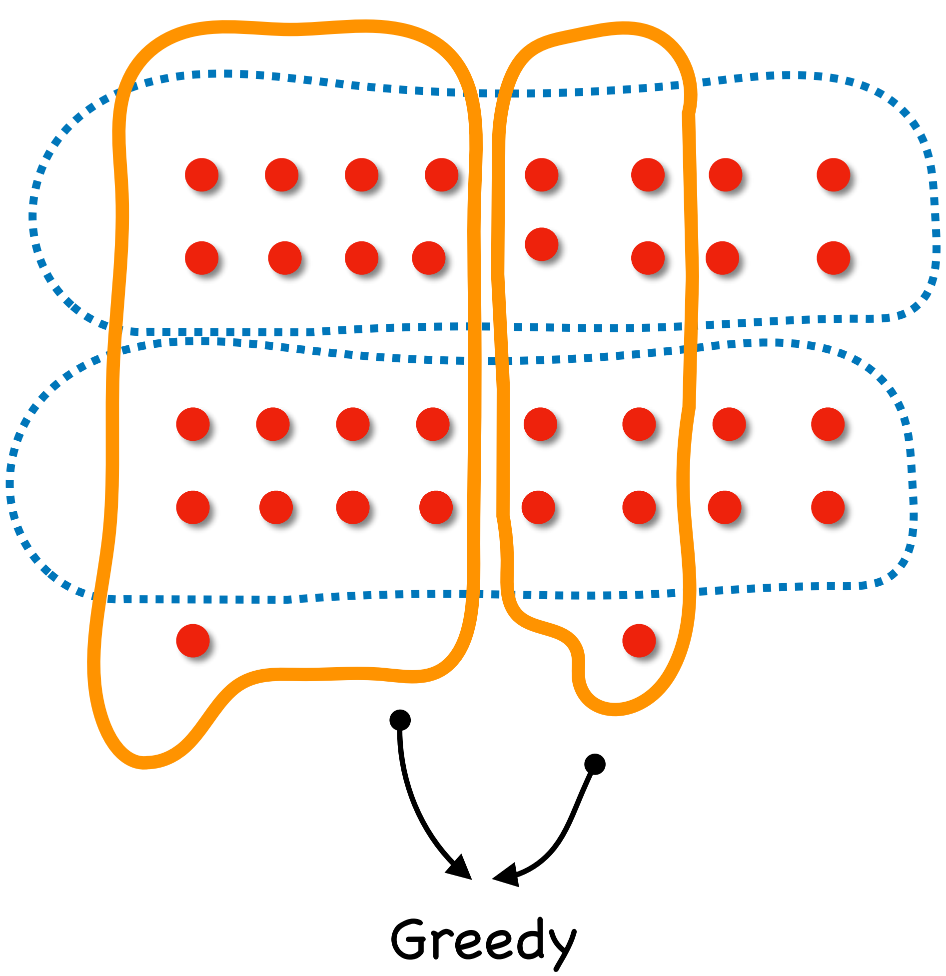
Classical NP-Hard problem

Facility and sensor allocation,
information retrieval, blog monitoring,...

Max-Cover

Greedy: pick the sets with largest coverage gain at each step

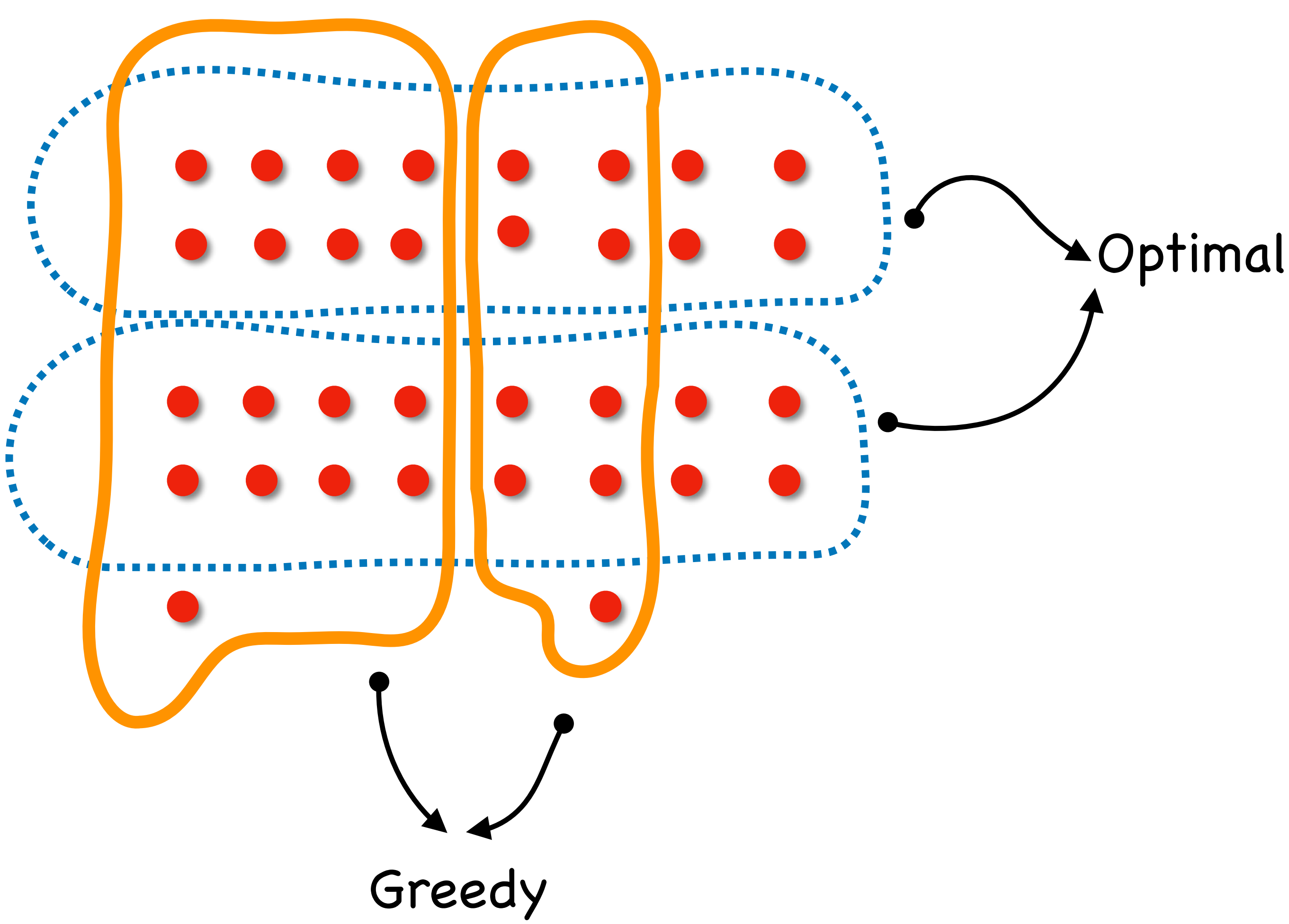
k=2



Max-Cover

Greedy: pick the sets with largest coverage gain at each step

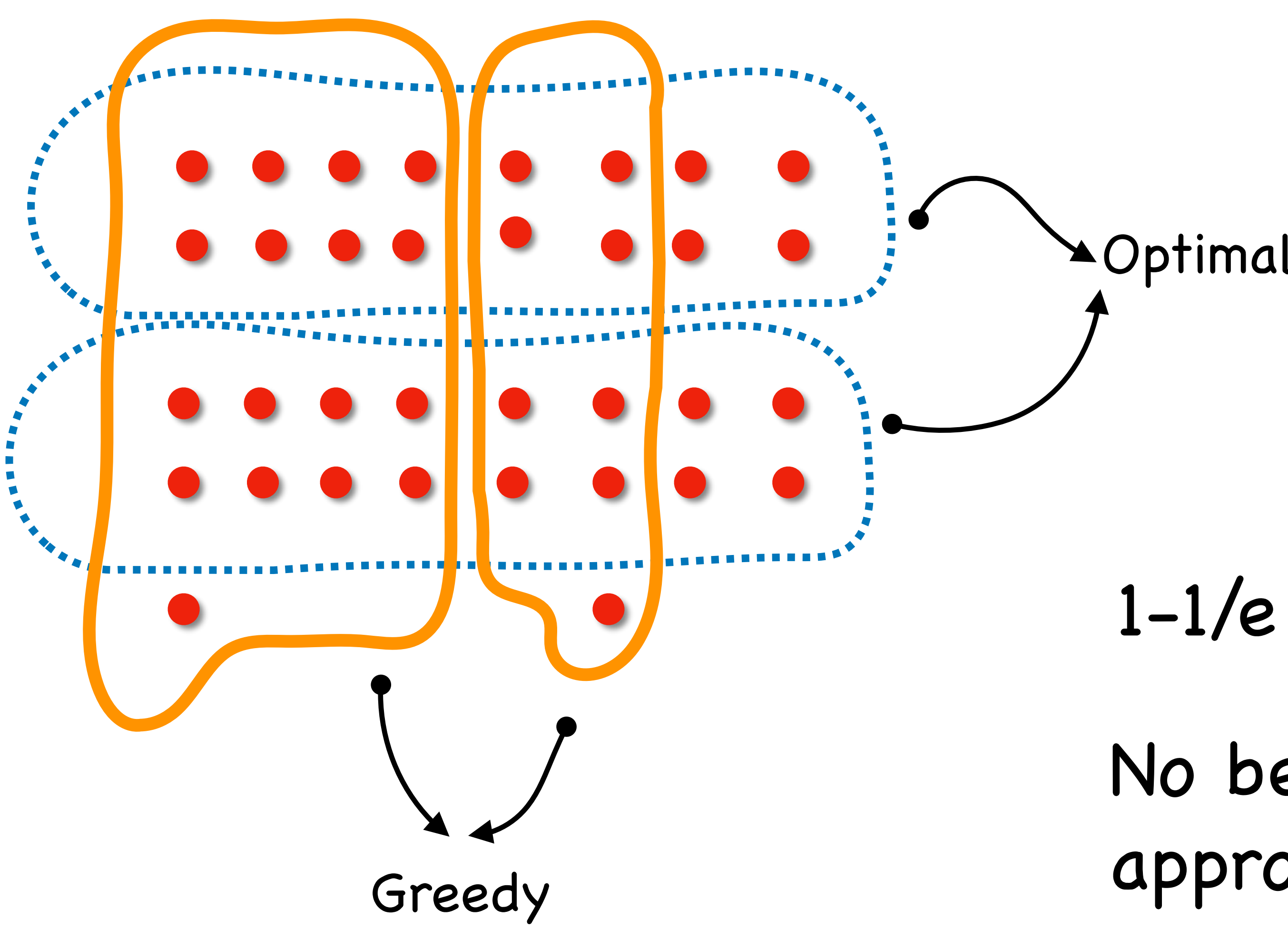
k=2



Max-Cover

Greedy: pick the sets with largest coverage gain at each step

k=2



1-1/e = 0.63…approximation

No better polynomial time approximation unless P=NP

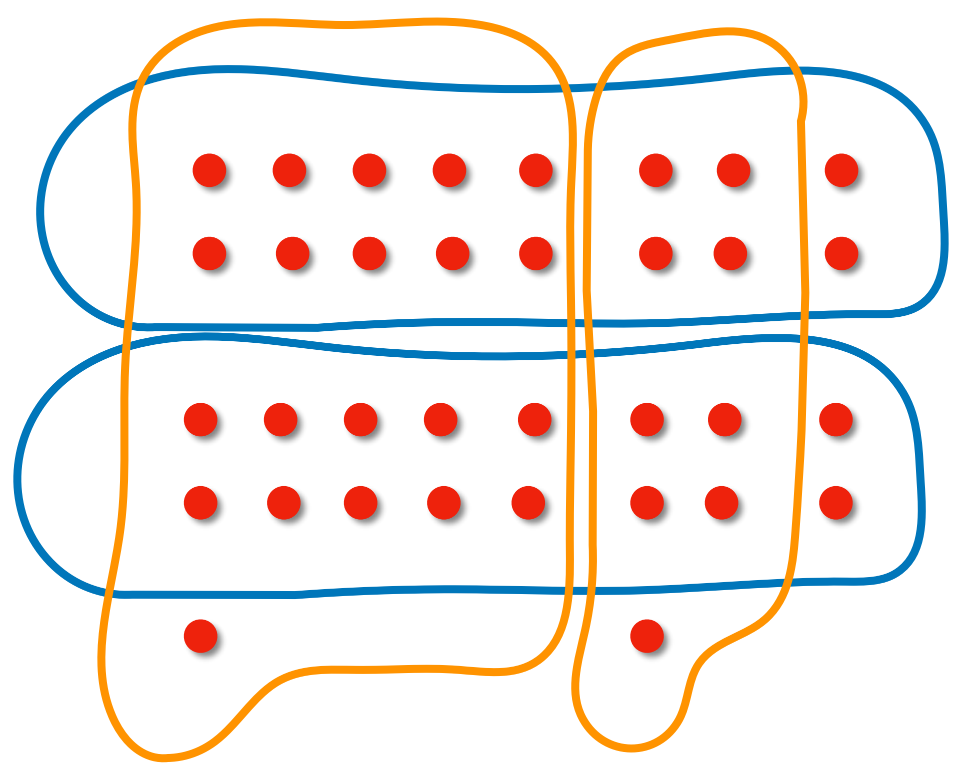
Streaming set model (Saha & Getoor)

m sets are encoded as (set ID, list of elements)

Return k set IDs as a solution to Max-Cover and a

$$(1 \pm \epsilon)|cover(solution)|$$

Set Cover and Max Cover well studied in this model



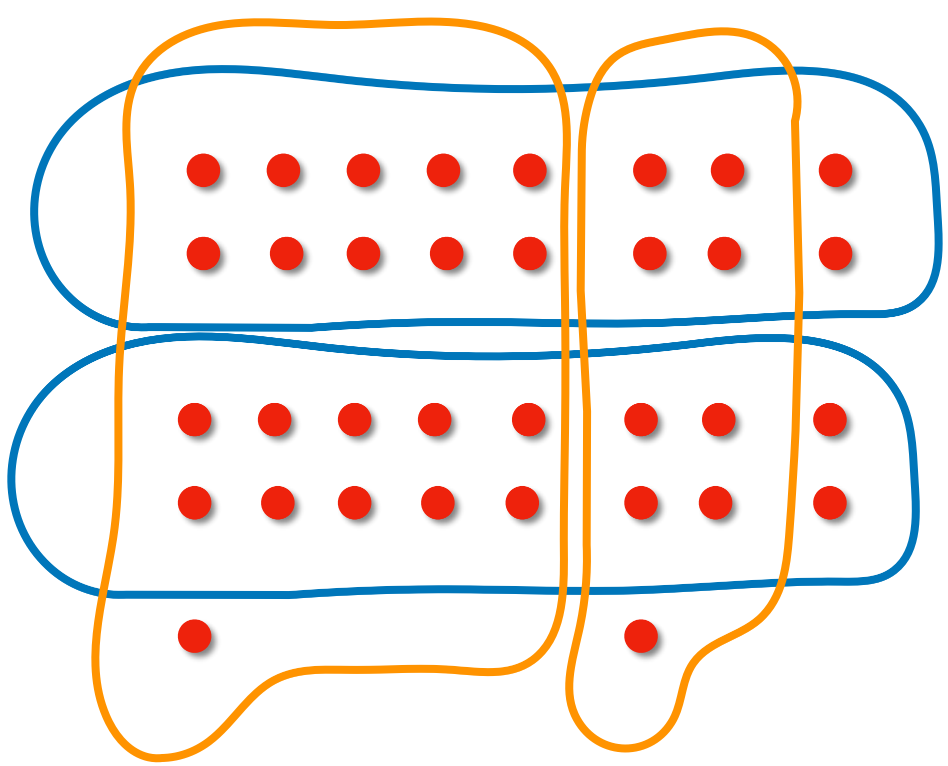
Streaming set model (Saha & Getoor)

m sets are encoded as (set ID, list of elements)

Return k set IDs as a solution to Max-Cover and a

$$(1 \pm \epsilon)|cover(solution)|$$

Set Cover and Max Cover well studied in this model



Set Cover

Assadi et al, 16

Chakrabarti & Wirth, 16

Indyk et al, 16

Assadi 17,...

Max Cover

Saha-Getoor, 08

McGregor-Vu, 17

Assadi, 17

Bateni et al., 17,...

Main results

Use sublinear o(mn) space

Main results

# of passes	1	1	1	$O(1/\epsilon)$
space				
approx.				

Main results

Use sublinear o(mn) space

Main results

# of passes	**1**	**1**	**1**	$O(1/\epsilon)$
space	$\tilde{O}(\epsilon^{-3} m)$			
approx.	$1 - \epsilon$ *			
	* exponential time			

Main results

Use sublinear o(mn) space

Main results

# of passes	1	1	1	$O(1/\epsilon)$
space	$\tilde{O}(\epsilon^{-3}m)$	$\tilde{O}(\epsilon^{-2}m)$		
approx.	$1-\epsilon$ *	$1-1/e-\epsilon$		
	* exponential time			

Main results

Use sublinear o(mn) space

Main results

# of passes	1	1	1	$O(1/\epsilon)$
space	$\tilde{O}(\epsilon^{-3}m)$	$\tilde{O}(\epsilon^{-2}m)$	$\tilde{O}(\epsilon^{-3}k)$	
approx.	$1-\epsilon$ *	$1-1/e-\epsilon$	$1/2-\epsilon$	
	* exponential time			

Main results

Use sublinear o(mn) space

Main results

# of passes	1	1	1	$O(1/\epsilon)$
space	$\tilde{O}(\epsilon^{-3}m)$	$\tilde{O}(\epsilon^{-2}m)$	$\tilde{O}(\epsilon^{-3}k)$	$\tilde{O}(\epsilon^{-2}k)$
approx.	$1-\epsilon$ *	$1-1/e-\epsilon$	$1/2-\epsilon$	$1-1/e-\epsilon$
	* exponential time			

Main results

Use sublinear o(mn) space

Main results

Does not depend on n
(ignoring polylog factors)

# of passes	1	1	1	$O(1/\epsilon)$
space	$\tilde{O}(\epsilon^{-3}m)$	$\tilde{O}(\epsilon^{-2}m)$	$\tilde{O}(\epsilon^{-3}k)$	$\tilde{O}(\epsilon^{-2}k)$
approx.	$1-\epsilon$ *	$1-1/e-\epsilon$	$1/2-\epsilon$	$1-1/e-\epsilon$
	* exponential time			

Main results

Independently discovered by Bateni et al.

Use sublinear o(mn) space

This talk

Main results

# of passes	**1**	**1**	**1**	$O(1/\epsilon)$
space	$\boxed{\tilde{O}(\epsilon^{-3}m)}$	$\boxed{\tilde{O}(\epsilon^{-2}m)}$	$\tilde{O}(\epsilon^{-3}k)$	$\tilde{O}(\epsilon^{-2}k)$
approx.	$\boxed{1-\epsilon}$ *	$\boxed{1-1/e-\epsilon}$	$1/2-\epsilon$	$1-1/e-\epsilon$
	* exponential time			

Main results

Lower bounds

Theorem (McGregor-Vu, 17): Any constant pass (randomized) algorithm with a 1-1/e+0.01 approximation requires

$$\Omega(m/k^2)$$

space.

Main results

Lower bounds

Theorem (McGregor-Vu, 17): Any constant pass (randomized) algorithm with a 1-1/e+0.01 approximation requires

$$\Omega(m/k^2)$$

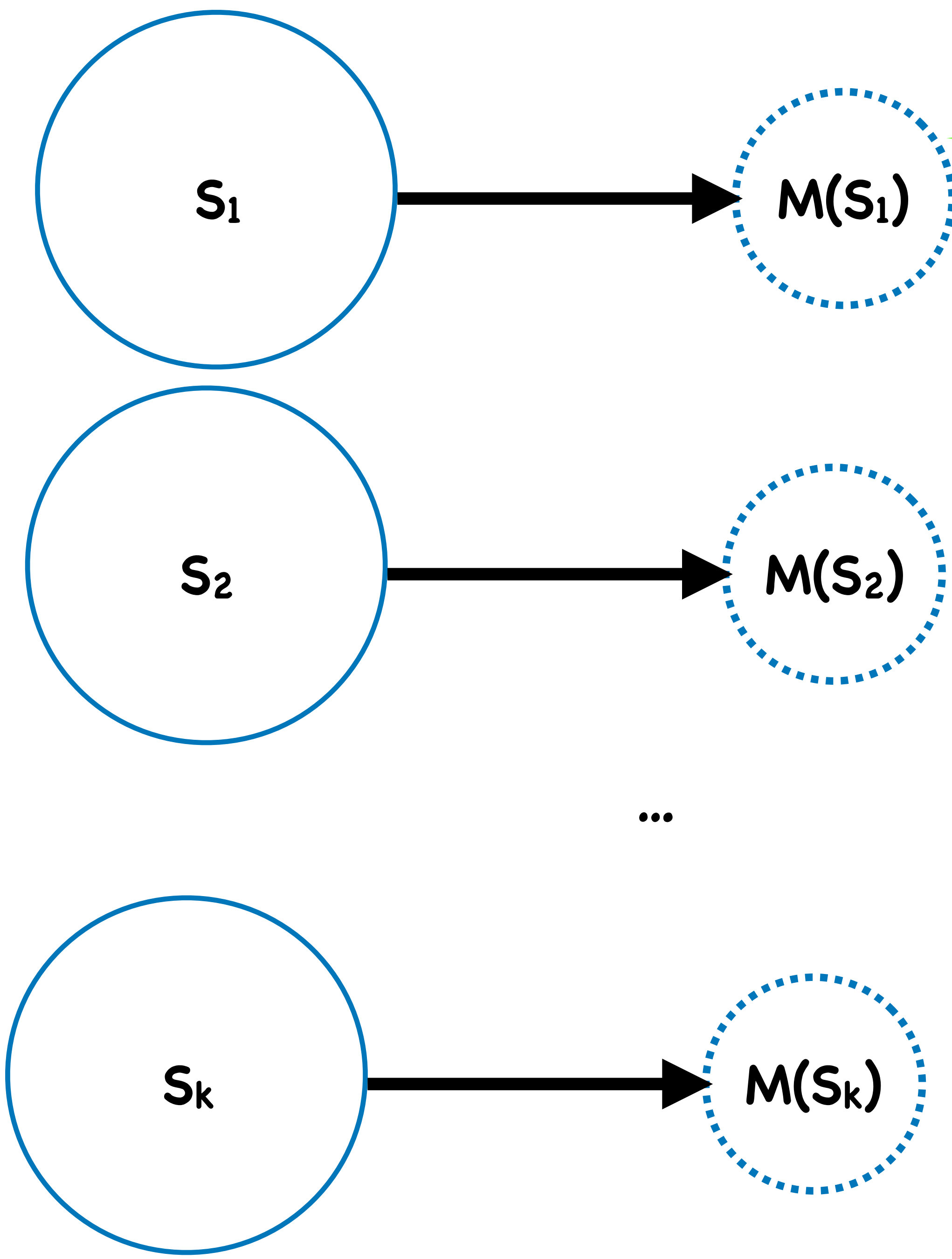
space.

Theorem (Assadi, 17): For k = O(1), any constant pass (randomized) algorithm with a $1-\epsilon$ approximation requires

$$\Omega(\epsilon^{-2}m)$$

space.

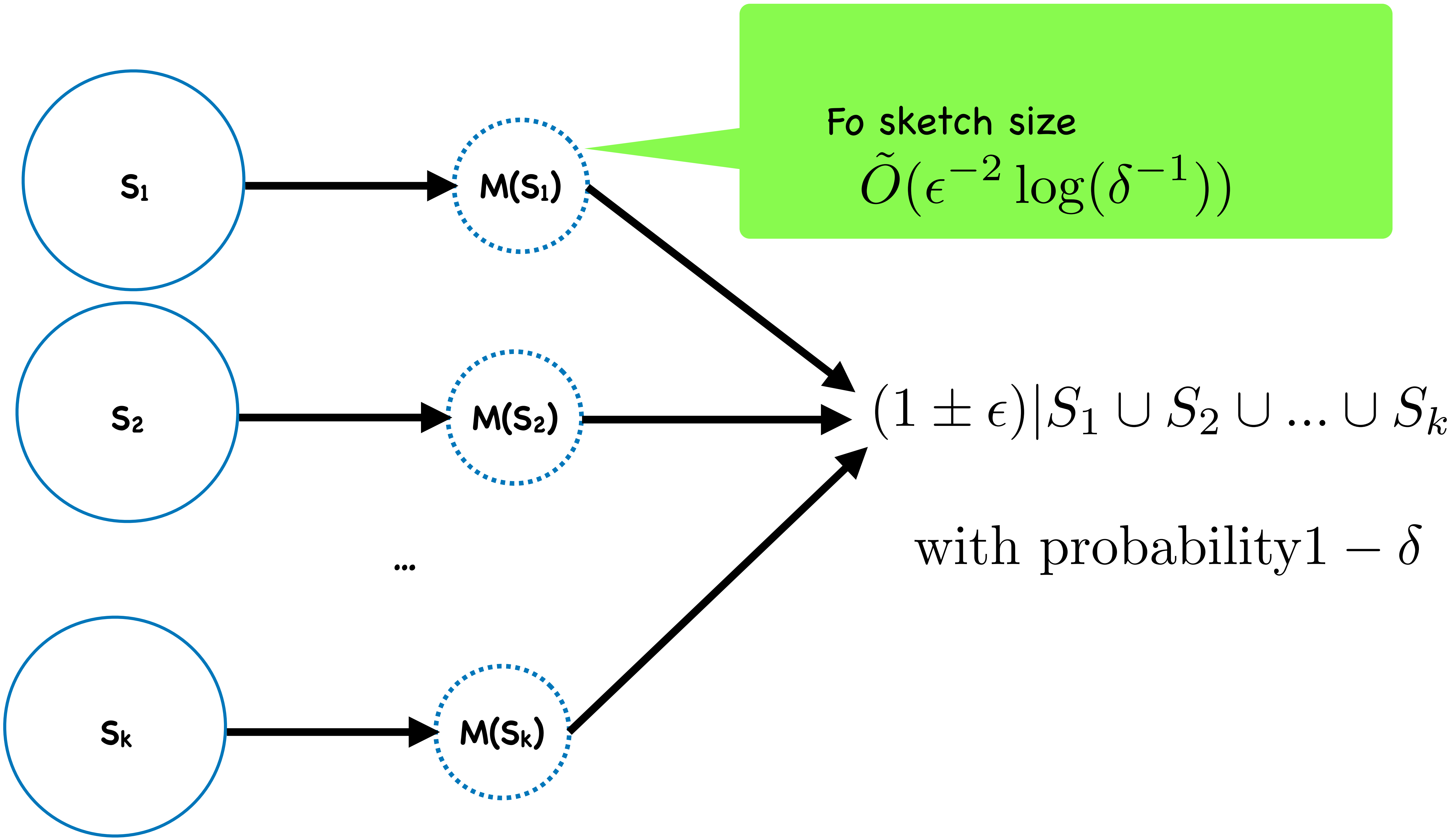
F_0 sketch algorithm



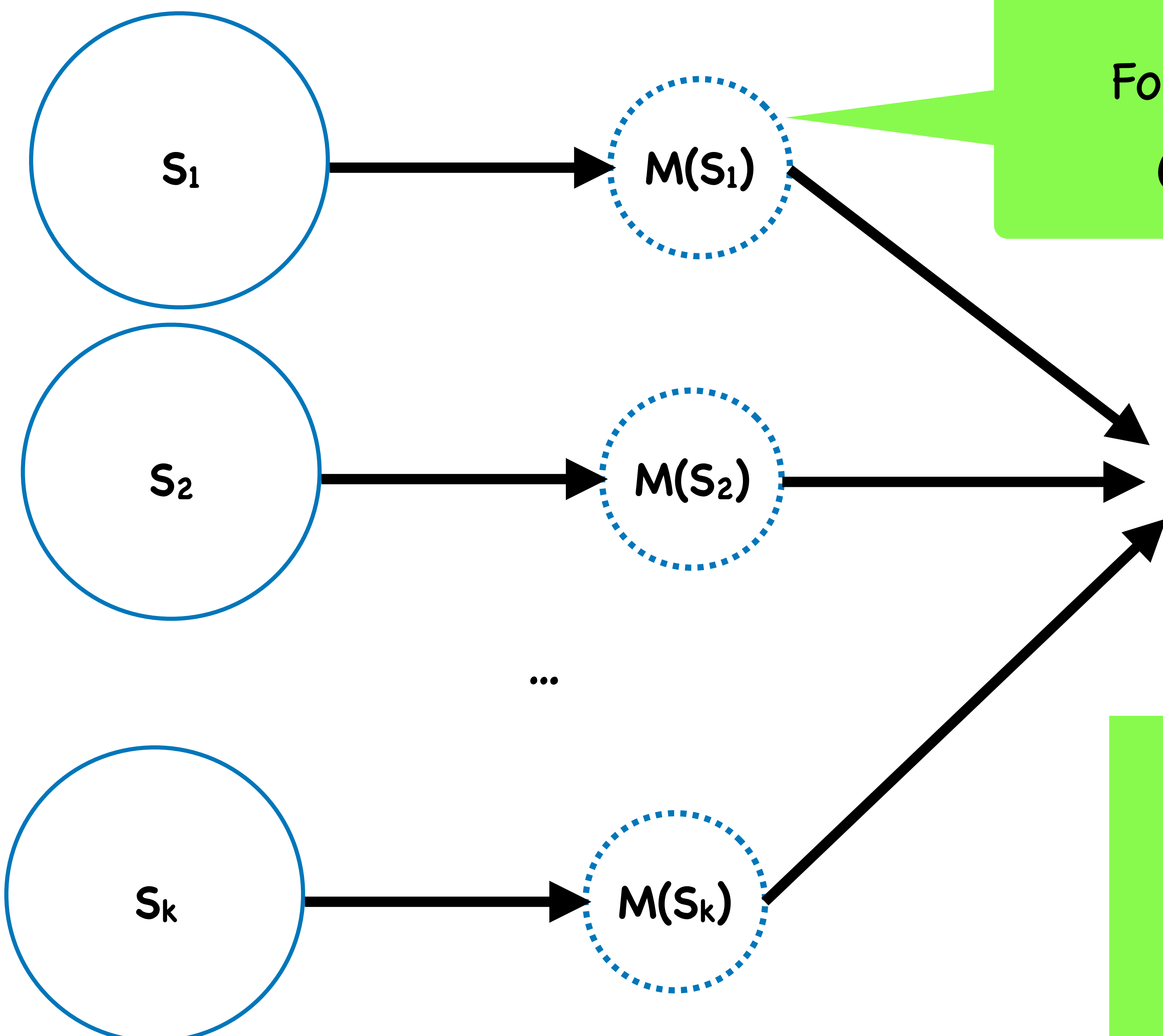
Fo sketch size

$$\tilde{O}(\epsilon^{-2}\log(\delta^{-1}))$$

F_0 sketch algorithm



F_0 sketch algorithm



Fo sketch size
$\tilde{O}(\epsilon^{-2}\log(\delta^{-1}))$

$$(1 \pm \epsilon)|S_1 \cup S_2 \cup \ldots \cup S_k|$$

with probability $1 - \delta$

$$\delta = \frac{1}{100\binom{m}{k}}$$

$$\text{space} = \tilde{O}(\epsilon^{-2} k m)$$

Improvement

$$\text{space} = \tilde{O}(\epsilon^{-2} km)$$

Improvement

$$\text{space} = \tilde{O}(\epsilon^{-2}\cancel{k}m)$$

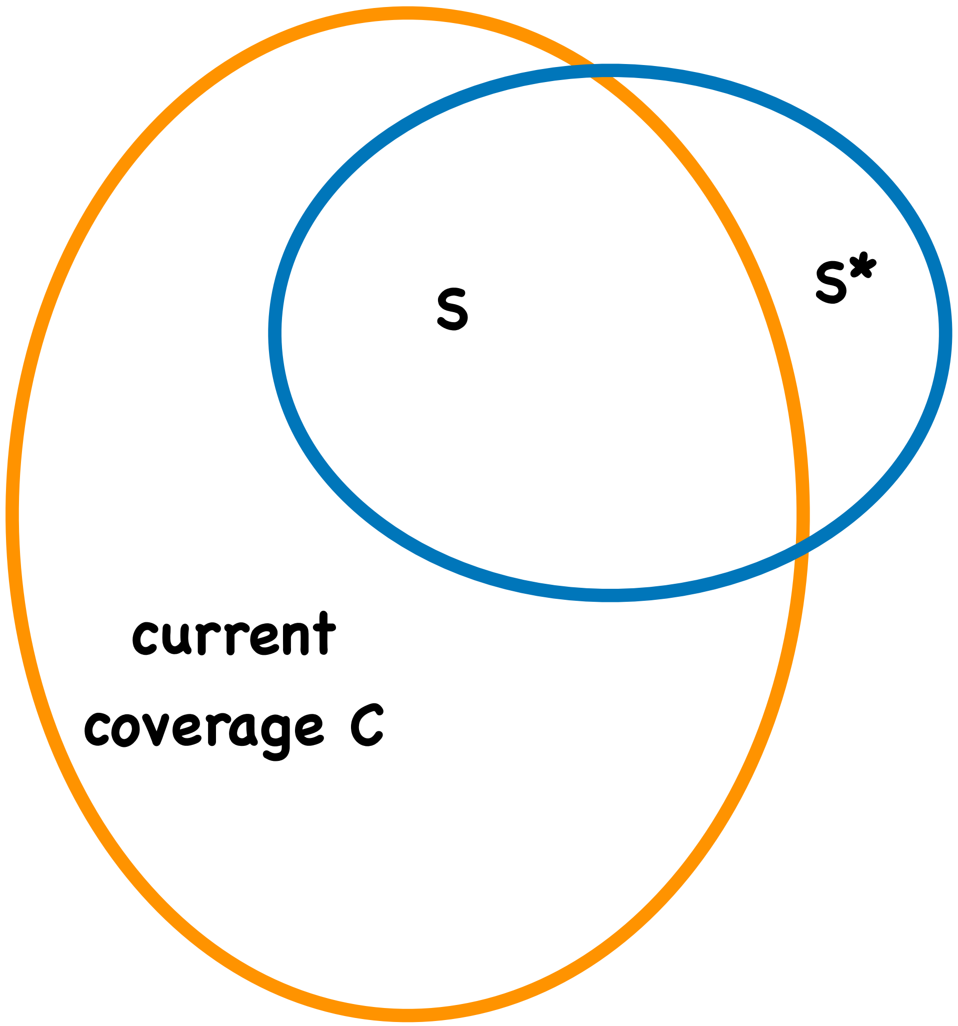
$$1 - 1/e - \epsilon \text{ approx.}$$

Improvement

$$\text{space} = \tilde{O}(\epsilon^{-2}\underset{\epsilon^{-1}}{\cancel{k}}\,m)$$

$1 - \epsilon$ approx.

Algorithm ideas

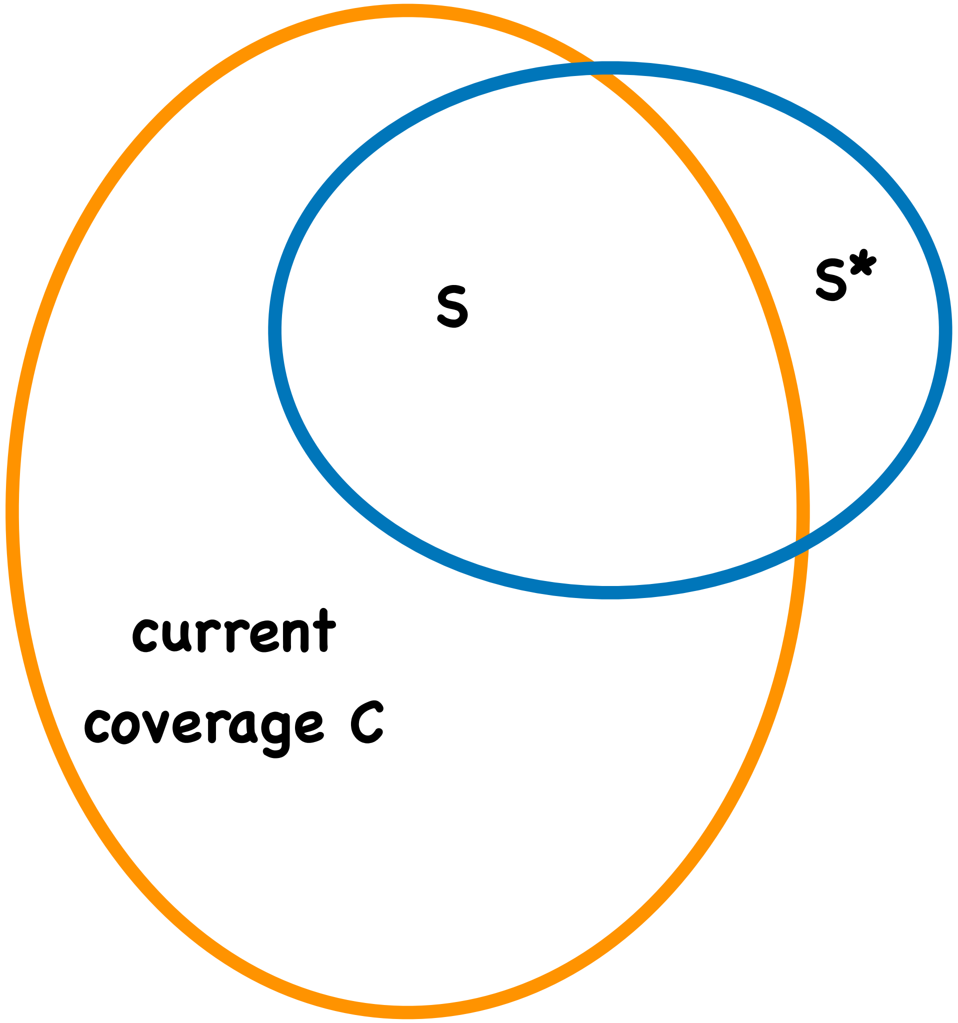


S* = S \ C coverage gain of S

If |S*| is small, store S*.

If |S*| is large, pick S and update C.

Algorithm ideas

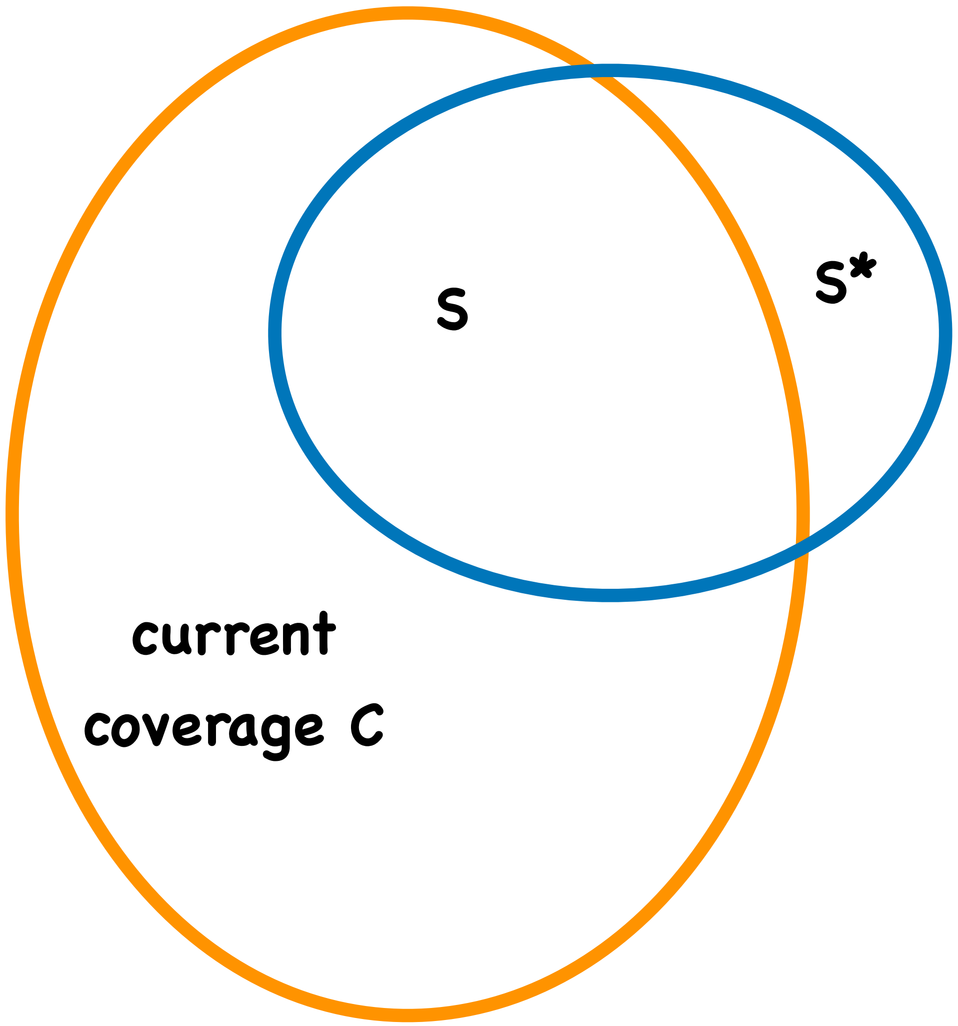


$S^* = S \setminus C$ coverage gain of S

If $|S^*|$ is small, store S^*.

If $|S^*|$ is large, pick S and update C.

Algorithm ideas

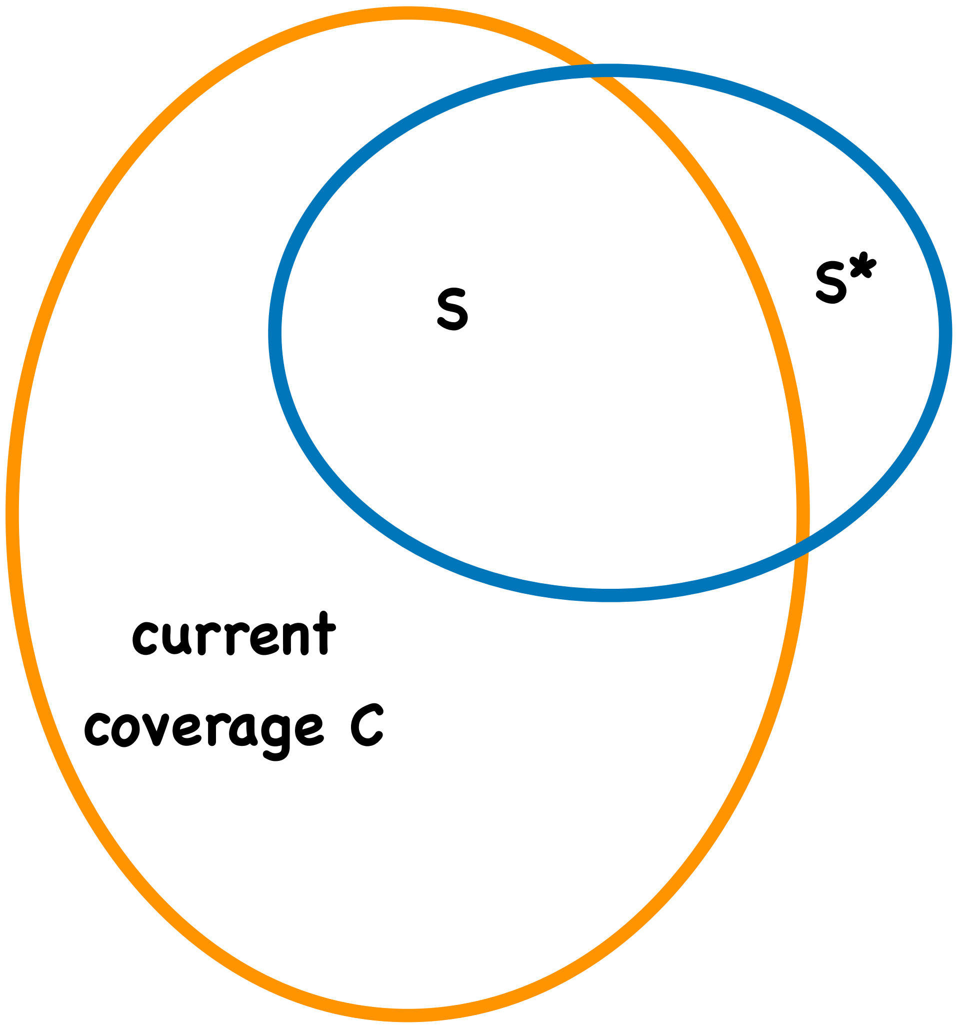


$S^* = S \setminus C$ coverage gain of S

If $|S^*|$ is small, store S^*.

If $|S^*|$ is large, pick S and update C.

Algorithm ideas

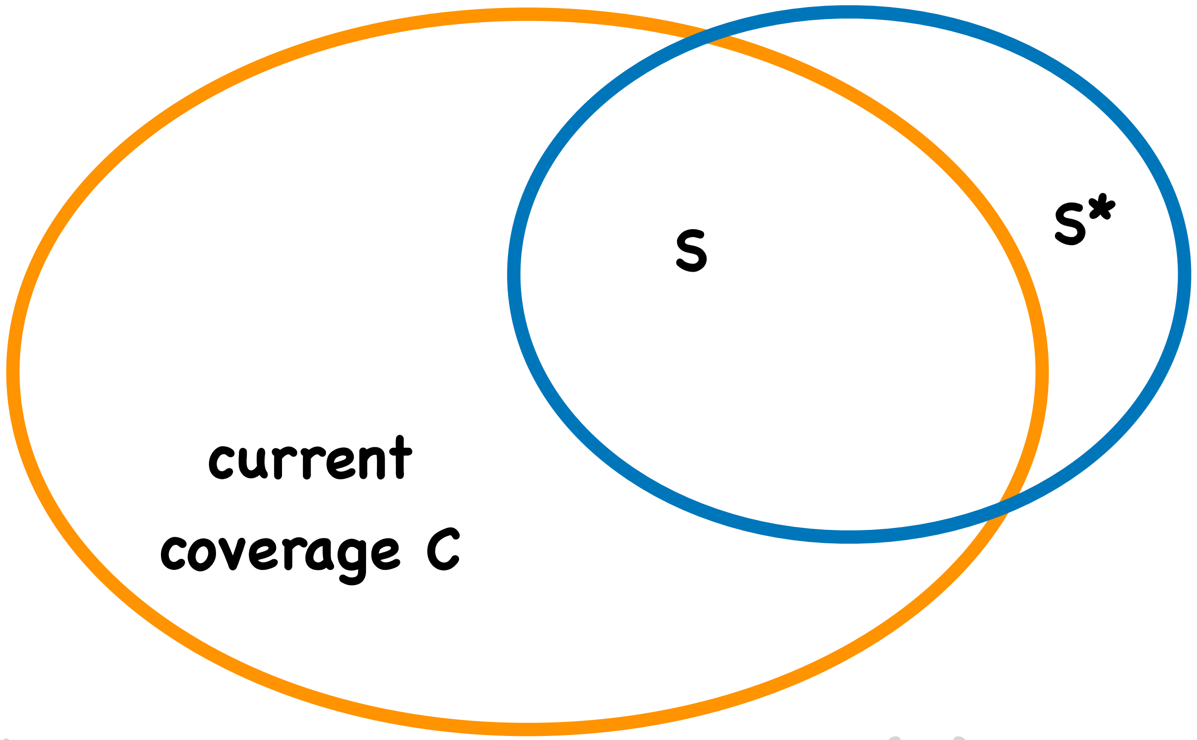


$S^* = S \setminus C$ coverage gain of S

If $|S^*|$ is small, store S^*.

If $|S^*|$ is large, pick S and update C.

Algorithm ideas



More formally:

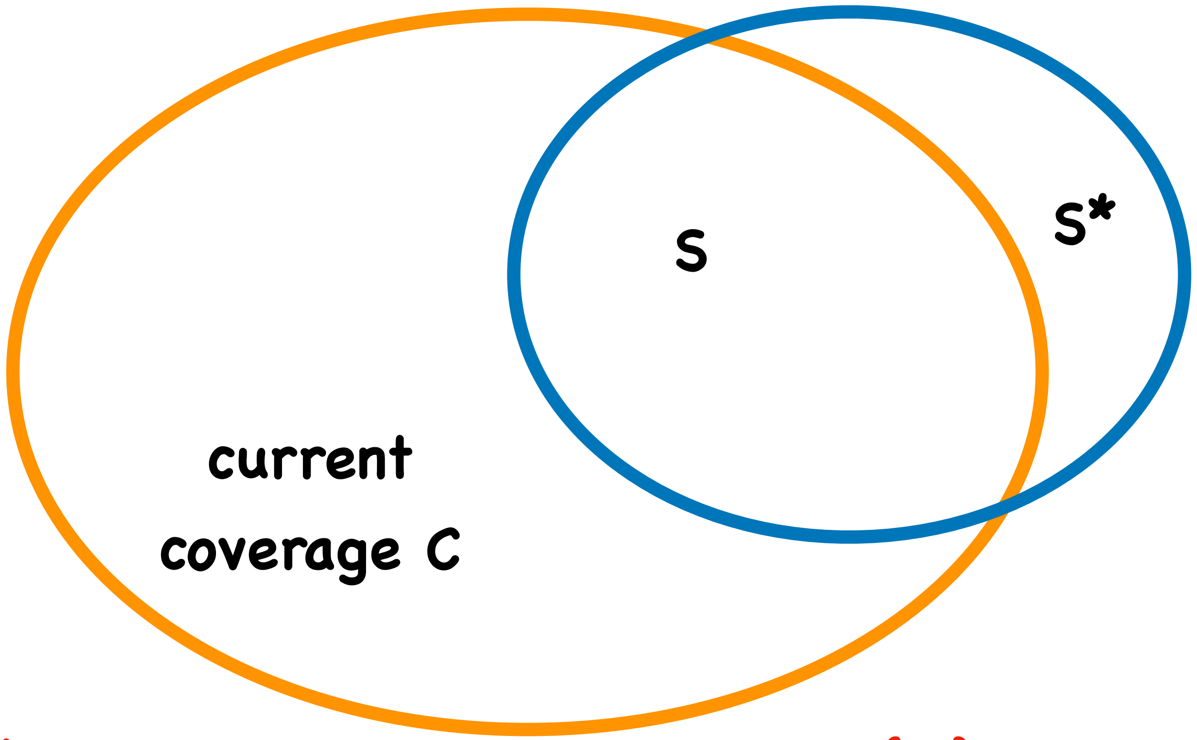
For each set S in the stream:

1) If S covers more than OPT/(kε) new elements, I = I U {S} and update C <- C U S.

2) Otherwise, store S* = S \ C in the memory.

3) Post-processing: find the best remaining sets from the memory.

Algorithm ideas



More formally:

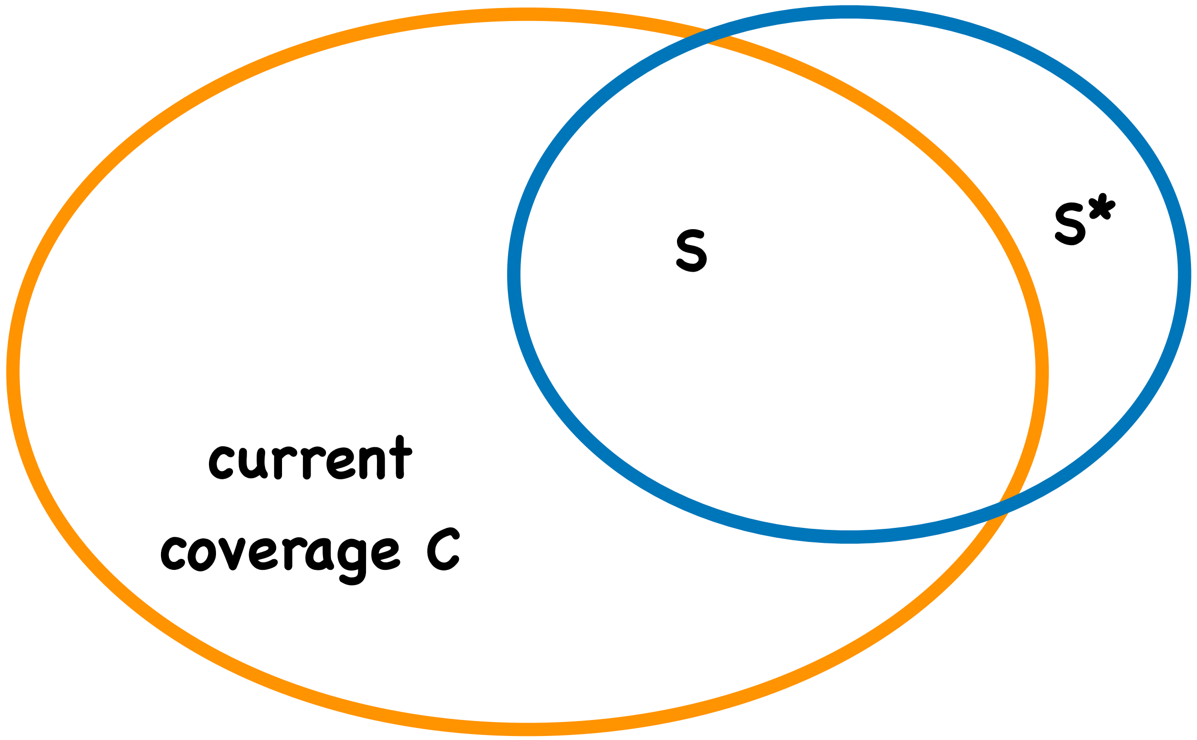
For each set S in the stream:

1) If S covers more than OPT/(kε) new elements, I = I U {S} and update C <- C U S.

2) Otherwise, store S* = S \ C in the memory.

3) Post-processing: find the best remaining sets from the memory.

Algorithm ideas



More formally:

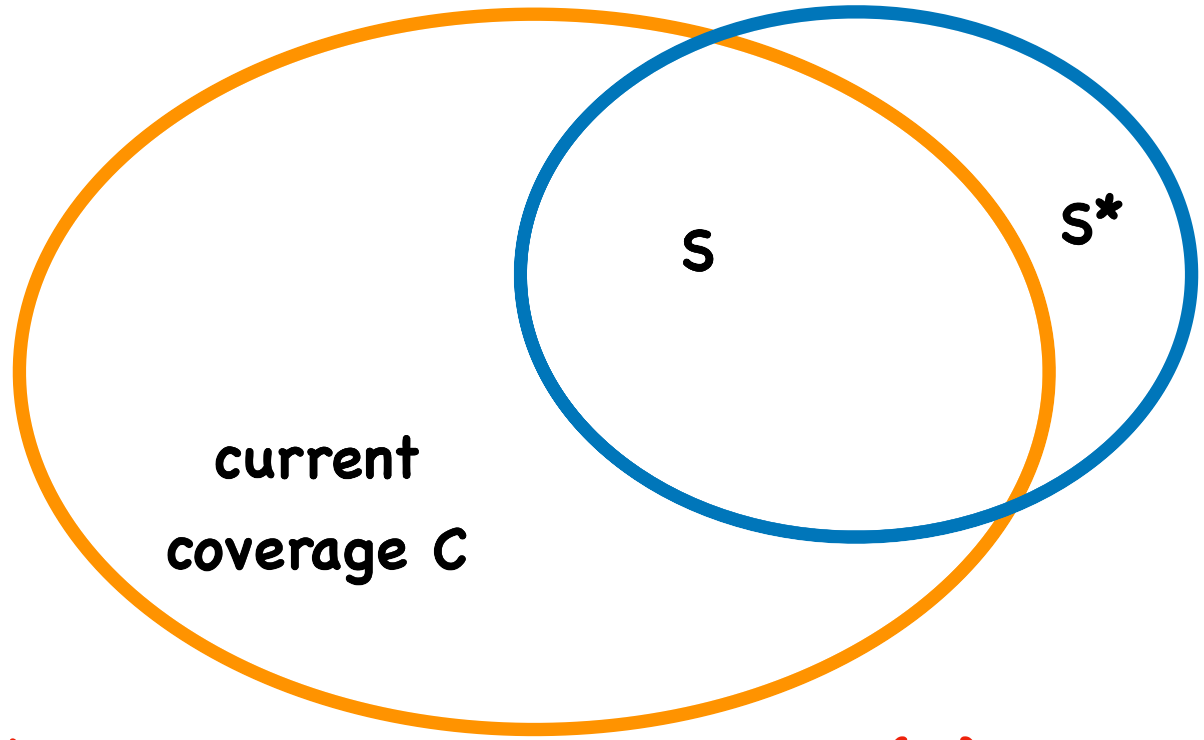
For each set S in the stream:

1) If S covers more than OPT/(kε) new elements, I = I U {S} and update C <- C U S.

2) Otherwise, store S* = S \ C in the memory.

3) Post-processing: find the best remaining sets from the memory.

Algorithm ideas



More formally:

For each set S in the stream:

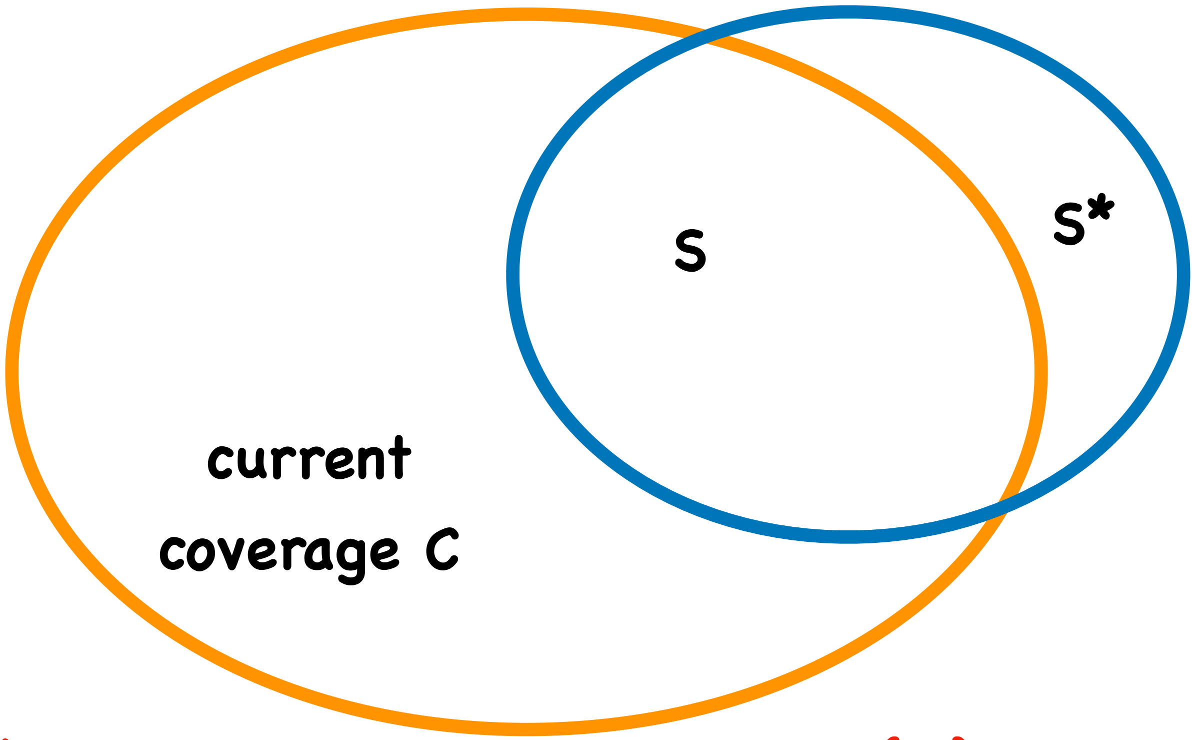
1) If S covers more than OPT/(kε) new elements, I = I U {S} and update C <- C U S.

2) Otherwise, store S* = S \ C in the memory.

3) Post-processing: find the best remaining sets from the memory.

use $OPT + \dfrac{mOPT}{k\epsilon}$ space

Algorithm ideas



More formally:

For each set S in the stream:

1) If S covers more than OPT/(kε) new elements, I = I U {S} and update C <- C U S.

2) Otherwise, store S* = S \ C in the memory.

3) Post-processing: find the best remaining sets from the memory.

Algorithm ideas

Lemma: The algorithm is a $1-\varepsilon$ approximation.

Proof sketch: Suppose y sets (with coverage A) picked during the stream and k-y sets pick at post-processing.

The result coverage is at least

Algorithm ideas

Lemma: The algorithm is a $1-\varepsilon$ approximation.

Proof sketch: Suppose y sets (with coverage A) picked during the stream and k-y sets pick at post-processing.

The result coverage is at least

Algorithm ideas

Lemma: The algorithm is a 1-ε approximation.

Proof sketch: Suppose y sets (with coverage A) picked during the stream and k-y sets pick at post-processing.

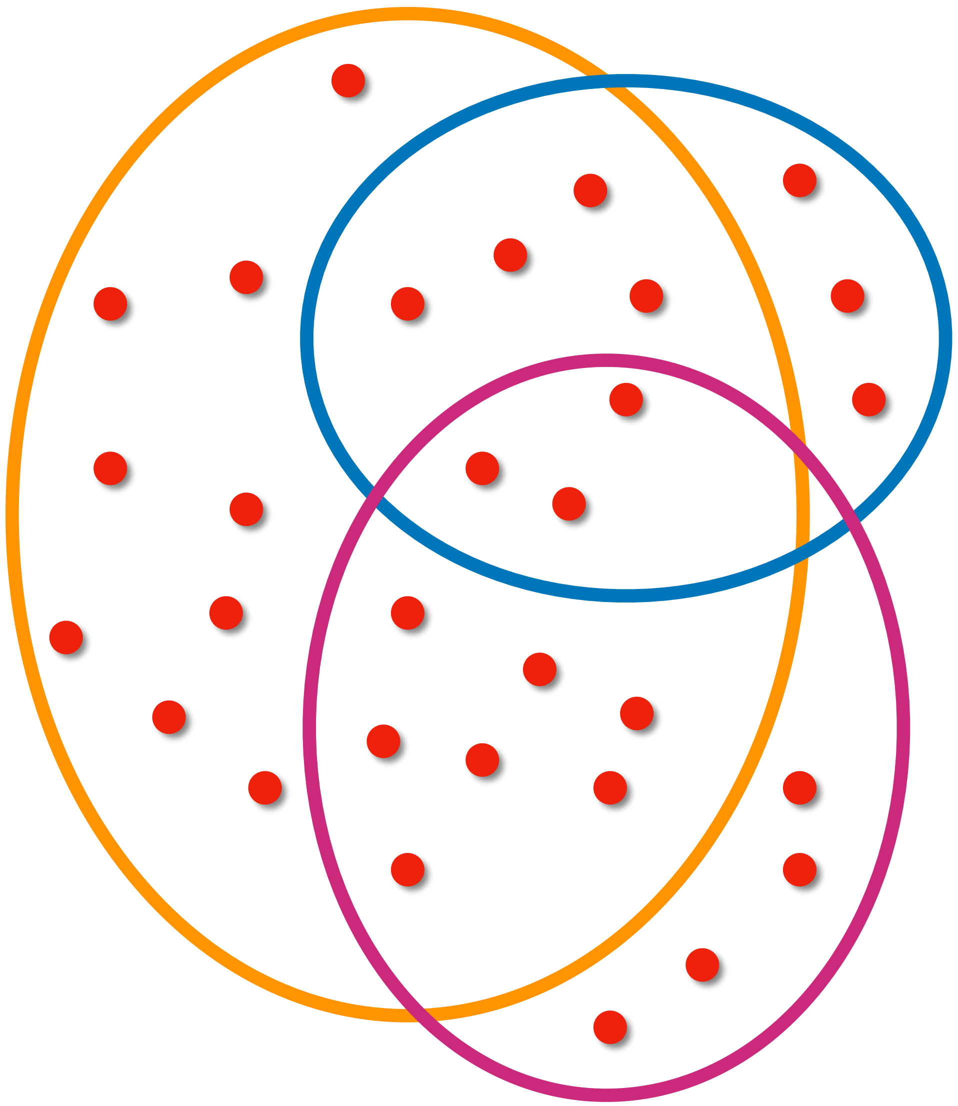
The result coverage is at least

$$|A| > y \cdot \frac{OPT}{k\epsilon}$$

$$\begin{aligned}
&|A| + \frac{k-y}{k}[OPT - |A|] \\
&= \left(1 - \frac{y}{k}\right) OPT + \frac{y}{k}|A| \\
&\geq \left(1 - \frac{y}{k}\right) OPT + \left(\frac{y}{k}\right)^2 \frac{1}{\epsilon} OPT \\
&\geq (1 - \epsilon/4) OPT
\end{aligned}$$

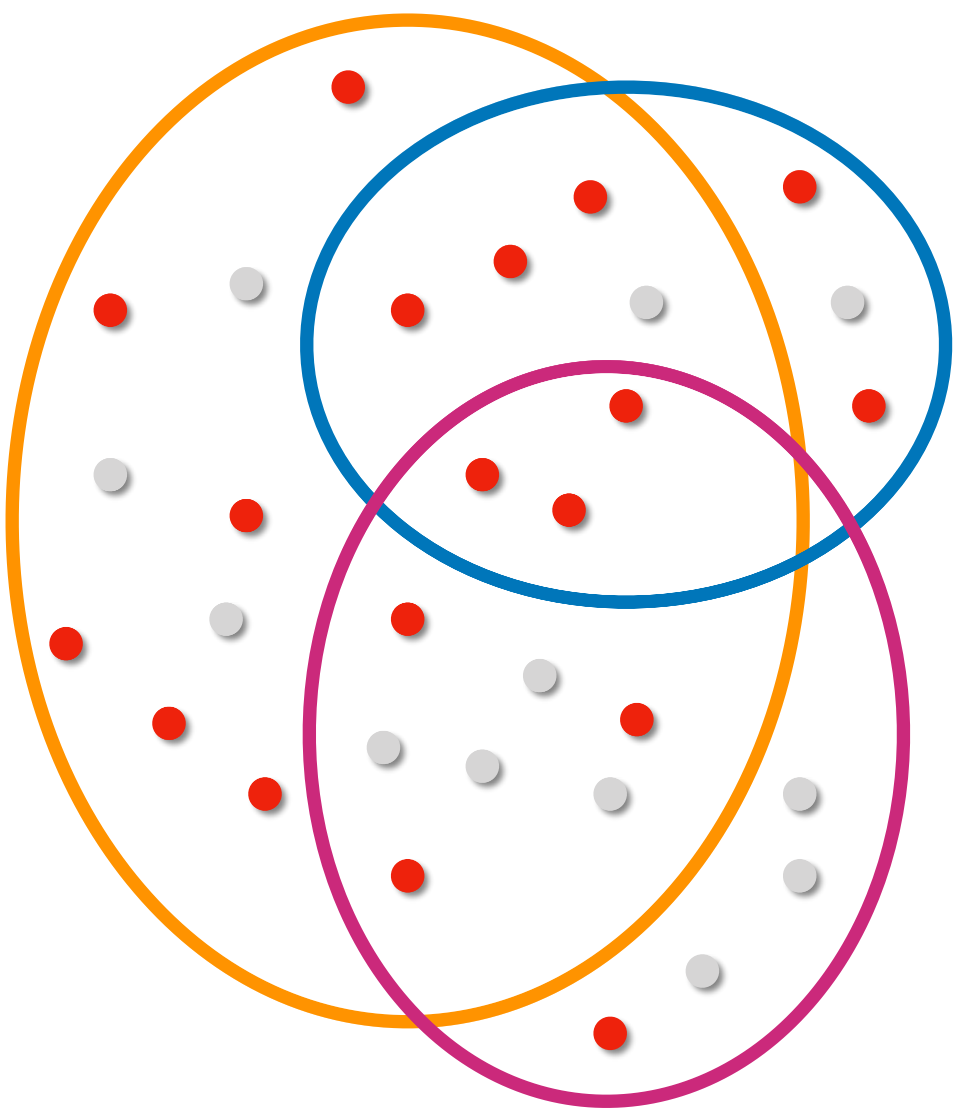
Algorithm ideas

Challenge: OPT = n in the worst case.



Algorithm ideas

Challenge: OPT = n in the worst case.



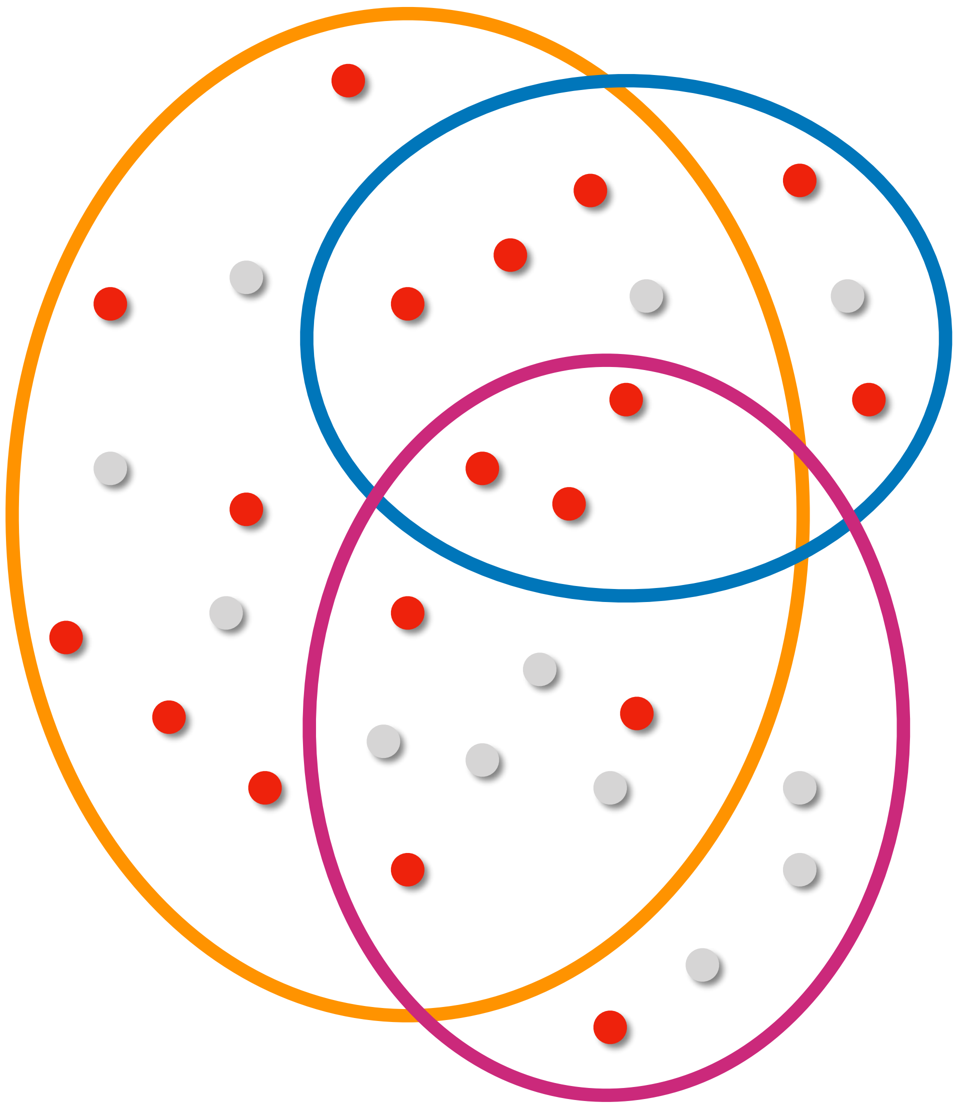
Subsampling: Subsample the universe U

with $p = \dfrac{ck \log m}{\epsilon^2 OPT}$

Run the algorithm on U′

Algorithm ideas

Challenge: OPT = n in the worst case.



Subsampling: Subsample the universe U

with $$p = \frac{ck \log m}{\epsilon^2 OPT}$$

Run the algorithm on U'.

Claim: Chernoff-Union argument

$$OPT' = \Theta(\epsilon^{-2} k \log m)$$

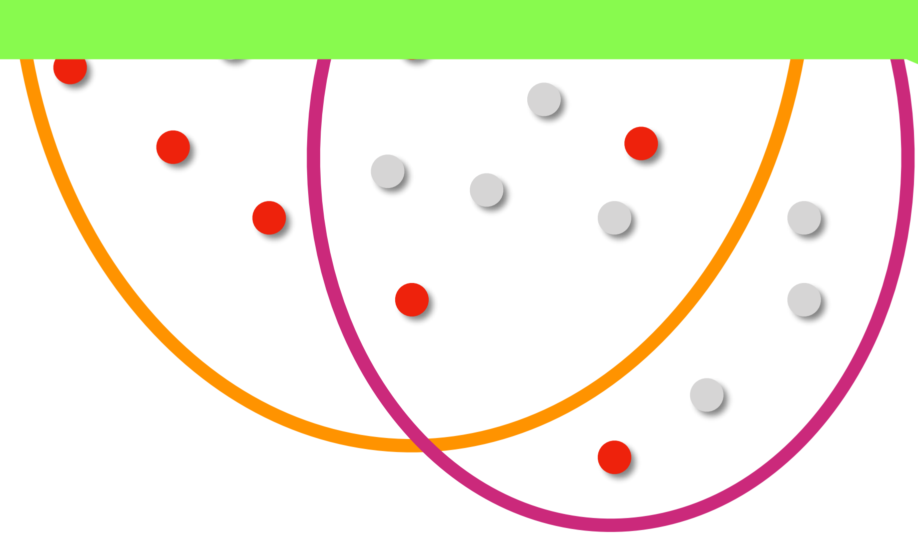
A good approx. in U' is also a good approx. in U.

Algorithm ideas

Challenge: OPT = n in the worst case.

$$\text{use } OPT' + \frac{mOPT'}{k\epsilon}$$

$$= \frac{m}{\epsilon^3} \text{ space}$$



Subsampling: Subsample the universe U

with $$p = \frac{ck \log m}{\epsilon^2 OPT}$$

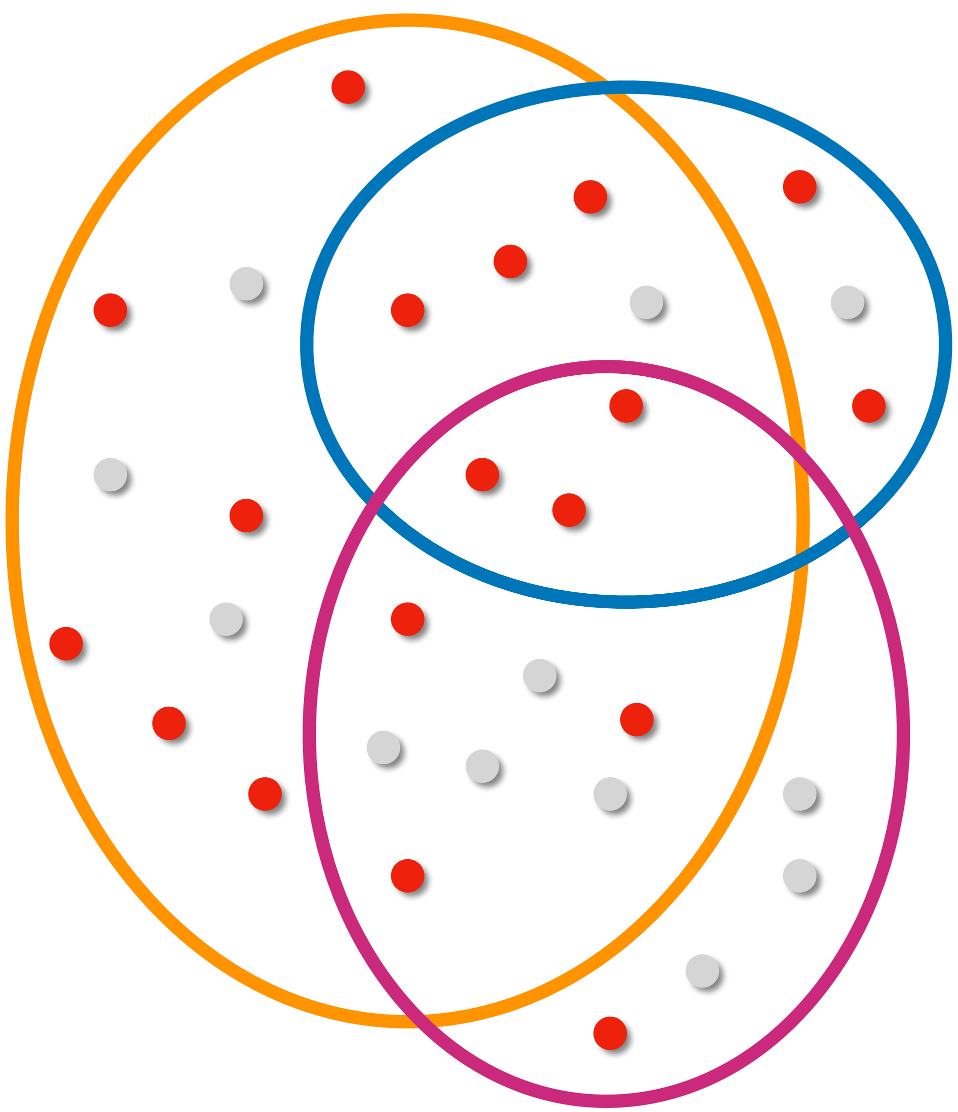
Run the algorithm on U′.

Claim: Chernoff-Union argument

$$OPT' = \Theta(\epsilon^{-2} k \log m)$$

A good approx. in U′ is also a good approx. in U.

Algorithm ideas



Other challenges:

OPT is unknown. Need guessing.

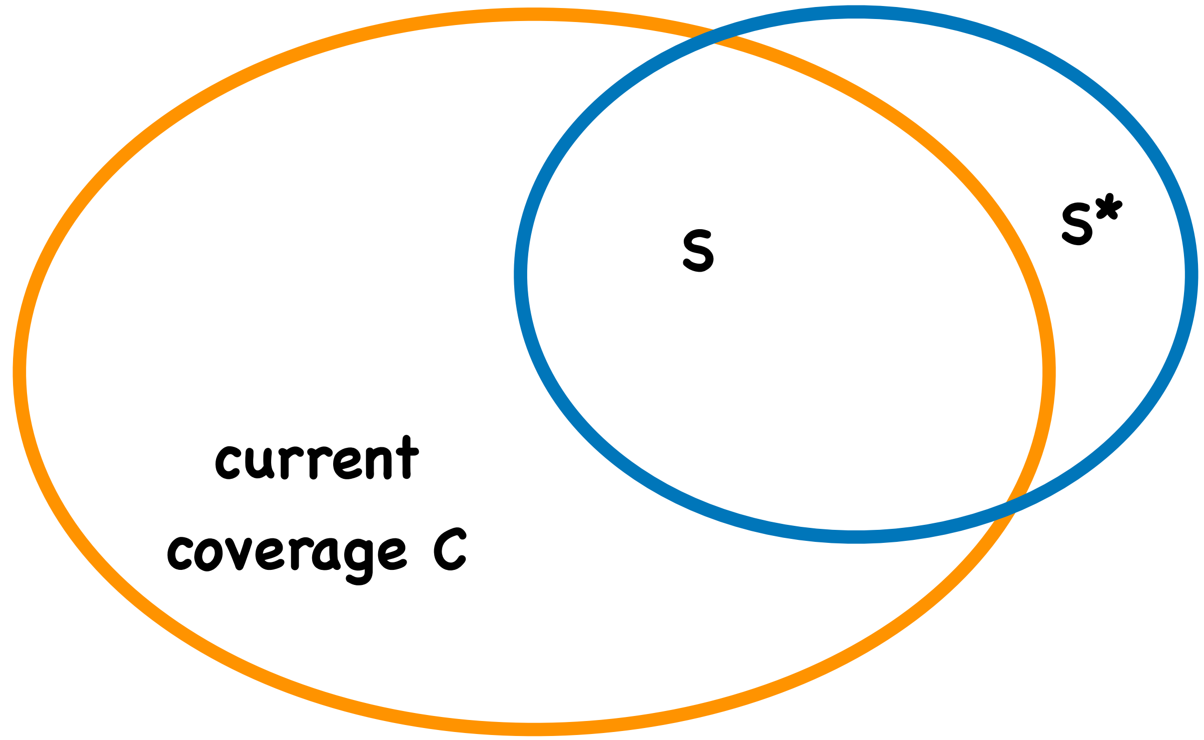
$$p = \frac{ck \log m}{\epsilon^2 OPT}$$

Small guesses —— large space

Large guesses —— inaccurate solution

Limited independent hash function analysis

Polynomial time version



More formally:

For each set S in the stream:

1) If S covers more than OPT/(k$\cancel{\delta}$) new elements, I = I U {S} and update C <- C U S.

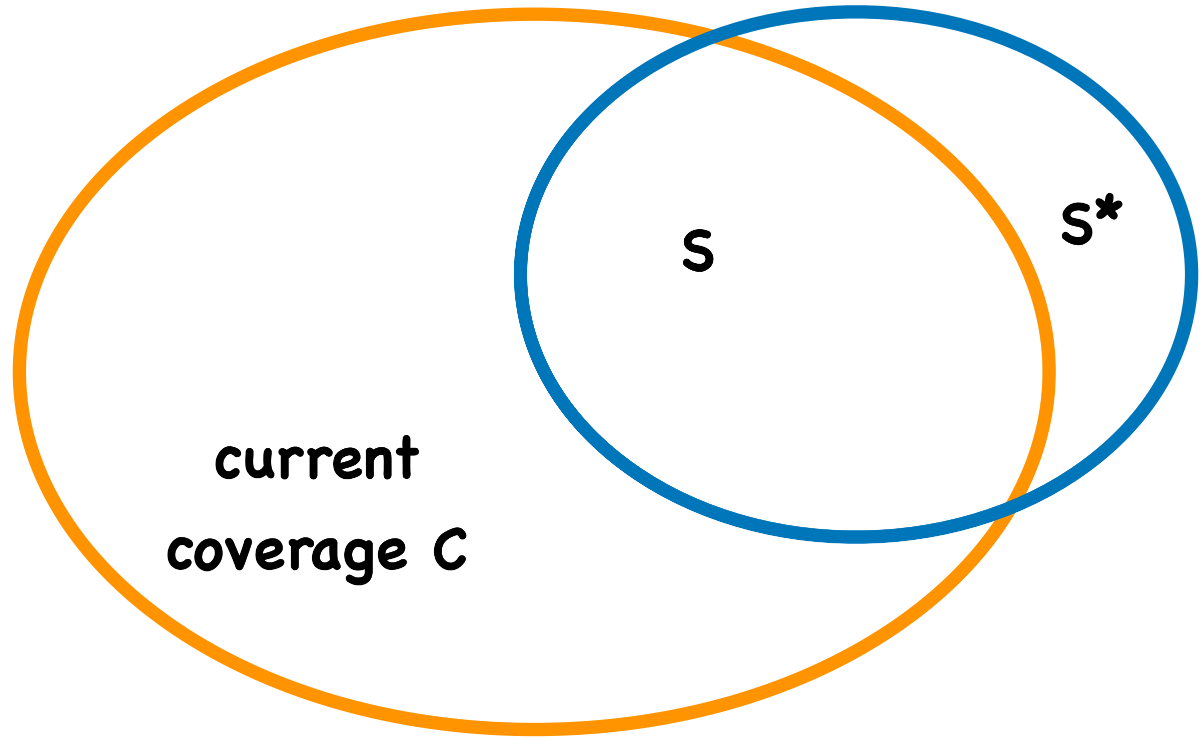
2) Otherwise, store S* = S \ C in the memory.

3) Post-processing: find ~~the best~~ remaining sets from the memory.
using greedy.

Polynomial time version

1-1/e approx.

$\frac{m}{\epsilon^2}$ space after subsampling



1) If S covers more than OPT/(k$\cancel{\epsilon}$) new elements, I = I U {S} and update C <- C U S.

2) Otherwise, store S* = S \ C in the memory.

3) Post-processing: find ~~the best~~ remaining sets from the memory. using greedy.

Lower bound

For k = O(1), any constant pass (randomized) algorithm with a 1-1/e+0.01 approximation requires

$$\Omega(m)$$

space.

k-player DISJOINTNESS: Each player i has a bit string x_i of length m.

Lower bound

k-player DISJOINTNESS: Each player i has a bit string x_i of length m.

Lower bound

k-player DISJOINTNESS: Each player i has a bit string x_i of length m.

player	bit 1	bit 2	...	bit m	
1	1	0	...	0	x_1
2	0	1	...	0	x_2
...	...	...	...	1	
k	0	0	...	0	x_k

Lower bound

k-player DISJOINTNESS: Each player has a bit string of length m.

player	bit 1	bit 2	...	bit m
1	1	0	...	0
2	0	1	...	0
...	...	...	...	1
k	0	0	...	0

NO Instance

at most one 1 in each column

Lower bound

k-player DISJOINTNESS: Each player has a bit string of length m.

player	bit 1	bit 2	...	bit m
1	1	0	...	0
2	0	1	...	0
...	...	...	...	1
k	0	0	...	0

NO Instance

at most one 1 in each column

player	bit 1	bit 2	...	bit m
1	1	1	...	0
2	0	1	...	0
...	...	1	...	1
k	0	1	...	0

YES Instance

one unique column with all 1

Lower bound

k-player DISJOINTNESS: Each player has a bit string of length m.

player	bit 1	bit 2	...	bit m
1	1	0	...	0
2	0	1	...	0
...	...	...	...	1
k	0	0	...	0

NO Instance

at most one 1 in each column

player	bit 1	bit 2	...	bit m
1	1	1	...	0
2	0	1	...	0
...	...	1	...	1
k	0	1	...	0

YES Instance

one unique column with all 1

player 1 -> player 2 -> ... -> player k -> YES/NO

Any randomized protocol requires Ω(m) communication.

Lower bound

k-player DISJOINTNESS: Each player has a bit string of length m.

player	bit 1	bit 2	...	bit m
1	1	0	...	0
2	0	1	...	0
...	...	...	...	1
k	0	0	...	0

NO Instance

at most one 1 in each column

player	bit 1	bit 2	...	bit m
1	1	1	...	0
2	0	1	...	0
...	...	1	...	1
k	0	1	...	0

YES Instance

one unique column with all 1

player 1 -> player 2 -> ... -> player k -> YES/NO

Any randomized protocol requires Ω(m) communication.

Lower bound

k-player DISJOINTNESS: Each player has a bit string of length m.

player	bit 1	bit 2	...	bit m
1	1	0	...	0
2	0	1	...	0
...	...	...	...	1
k	0	0	...	0

NO Instance

at most one 1 in each column

player	bit 1	bit 2	...	bit m
1	1	1	...	0
2	0	1	...	0
...	...	1	...	1
k	0	1	...	0

YES Instance

one unique column with all 1

player 1 -> player 2 -> ... -> player k -> YES/NO

Any randomized protocol requires $\Omega(m)$ communication.

Lower bound

k-player DISJOINTNESS: Each player has a bit string of length m.

player	bit 1	bit 2	...	bit m
1	1	0	...	0
2	0	1	...	0
...	...	...	...	1
k	0	0	...	0

NO Instance

at most one 1 in each column

player	bit 1	bit 2	...	bit m
1	1	1	...	0
2	0	1	...	0
...	...	1	...	1
k	0	1	...	0

YES Instance

one unique column with all 1

player 1 -> player 2 -> ... -> player k -> YES/NO

Any randomized protocol requires $\Omega(m)$ communication.

Lower bound

k-player DISJOINTNESS: Each player has a bit string of length m.

player	bit 1	bit 2	...	bit m
1	1	0	...	0
2	0	1	...	0
...	...	...	...	1
k	0	0	...	0

NO Instance

at most one 1 in each column

player	bit 1	bit 2	...	bit m
1	1	1	...	0
2	0	1	...	0
...	...	1	...	1
k	0	1	...	0

YES Instance

one unique column with all 1

player 1 -> player 2 -> ... -> player k -> YES/NO

Any randomized protocol requires $\Omega(m)$ communication.

Lower bound

k-player DISJOINTNESS: Use public randomness, generates S(i,j)

player	bit 1	bit 2	...	bit m
1	S(1,1)	S(1,2)	...	S(1,m)
2	S(2,1)	S(2,2)	...	S(2,m)
...	...	...	...	...
k	S(k,1)	S(k,2)	...	S(k,m)

$$S(1,j), S(2,j), \ldots, S(k,j)$$

have the same size and partition [n]

$$\bigcup_j S(i,j) = [n]$$

Lower bound

k-player DISJOINTNESS: Use public randomness, generates S(i,j)

player	bit 1	bit 2	...	bit m
1	1 S(1,1)	0	...	0
2	0	1 S(2,2)	...	1 S(m,2)
...	...	...	...	...
k	0	0	...	0

$$S(1,j), S(2,j), \ldots, S(k,j)$$

have the same size and partition [n]

$$\bigcup_j S(i,j) = [n]$$

If $x_{i,j}$=1, player i put S(i,j) in the stream.

Lower bound

NO instance

player	bit 1	bit 2	...	bit m
1	1 S(1,1)	0	...	0
2	0	1 S(2,2)	...	1 S(m,2)
...	...	...	...	...
k	0	0	...	0

NO Instance

max cover < (1-1/e+0.01)n

Chernoff-Union bound argument

Sets are random subsets of size n/k.

The expected coverage of k sets is

$$(1-(1-1/k)^k)n < (1-1/e+0.01)n$$

Lower bound

YES instance

player	bit 1	bit 2	...	bit m
1	1 S(1,1)	1 S(1,2)	...	0
2	0	1 S(2,2)	...	1 S(2,m)
...	...	...	...	...
k	0	1 S(k,2)	...	0

YES Instance

max cover = n

The sets in the all-1 column cover [n].

Lower bound

player	bit 1	bit 2	...	bit m
1	1 S(1,1)	1 S(1,2)	...	0
2	0	1 S(2,2)	...	1 S(2,m)
...	...	...	...	...
k	0	1 S(k,2)	...	0

YES Instance

max cover = n

player	bit 1	bit 2	...	bit m
1	1 S(1,1)	0	...	0
2	0	1 S(2,2)	...	1 S(m,2)
...	...	...	...	...
k	0	0	...	0

NO Instance

max cover < (1-1/e+0.01)n

Lower bound

player	bit 1	bit 2	...	bit m
1	1 S(1,1)	1 S(1,2)	...	0
2	0	1 S(2,2)	...	1 S(2,m)
...	...	...	...	...
k	0	1 S(k,2)	...	0

YES Instance

max cover = n

player	bit 1	bit 2	...	bit m
1	1 S(1,1)	0	...	0
2	0	1 S(2,2)	...	1 S(m,2)
...	...	...	...	...
k	0	0	...	0

NO Instance

max cover < (1-1/e+0.01)n

A streaming algorithm with 1-1/e+0.01 approx. provides a communication protocol

$$\Longrightarrow \quad \Omega(m) \text{ space}$$

Other results in the literature

Multiple pass algorithm

Knapsack, matroid constraints

Sliding windows

Maximum k-vertex-cover

Streaming (monotone/non-monotone) submodular maximization

Other results in the literature

Multiple pass algorithm
(idea: thresholding greedy)

Knapsack, matroid constraints

Sliding windows

Maximum k-vertex-cover

Streaming (monotone/non-monotone) submodular maximization

Other results in the literature

Multiple pass algorithm

Knapsack, matroid constraints

Sliding windows (only consider the last w items/sets)

Maximum k-vertex-cover

Streaming (monotone/non-monotone) submodular maximization

Other results in the literature

Multiple pass algorithm

Knapsack, matroid constraints

Sliding windows (only consider the last w items/sets)

Maximum k-vertex-cover

Streaming (monotone/non-monotone) submodular maximization

Other results in the literature

Multiple pass algorithm

Knapsack, matroid constraints

Sliding windows

Maximum k-vertex-cover (find k vertices that cover the most number of edges)

Thank you!