MaxCover in MapReduce

Flavio Chierichetti, Ravi Kumar, Andrew Tomkins

Advisor

Klaus Berberich

Presented By:

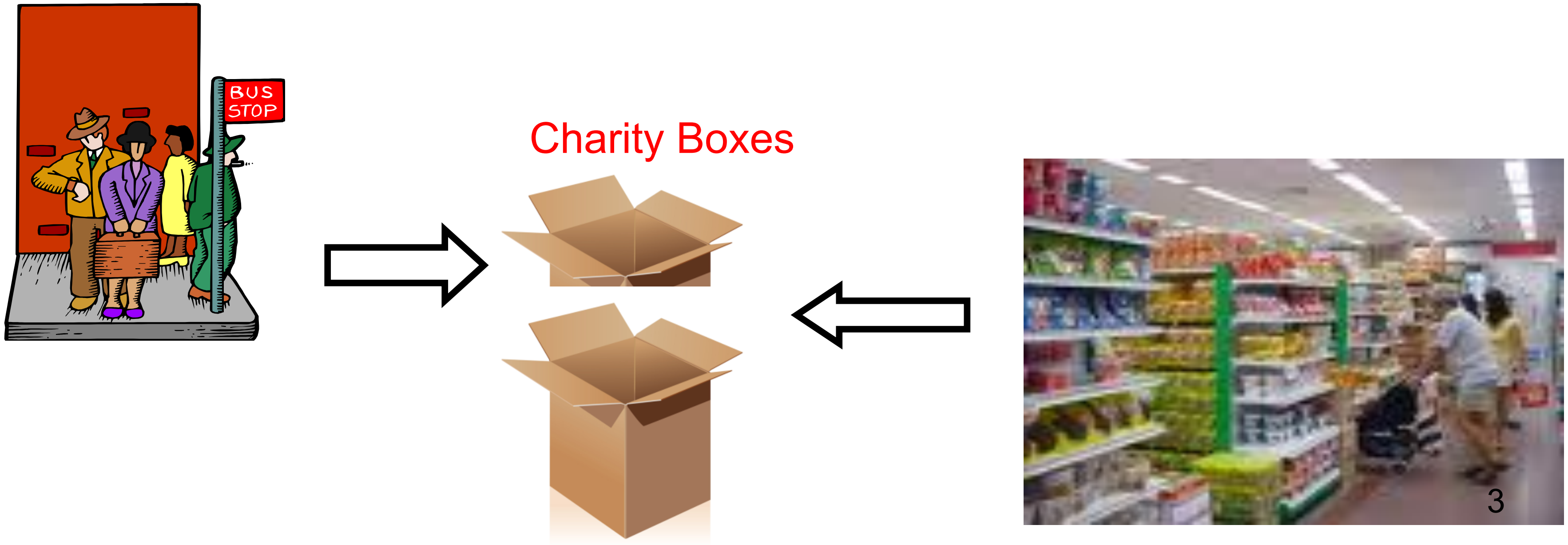
Isha Khosla

Outline

- Motivation
- Introduction
- Classical Approach: Greedy
- Proposed Algorithm: M_R Greedy
- Possible extension
- Experiments
- Weaknesses
- Conclusion

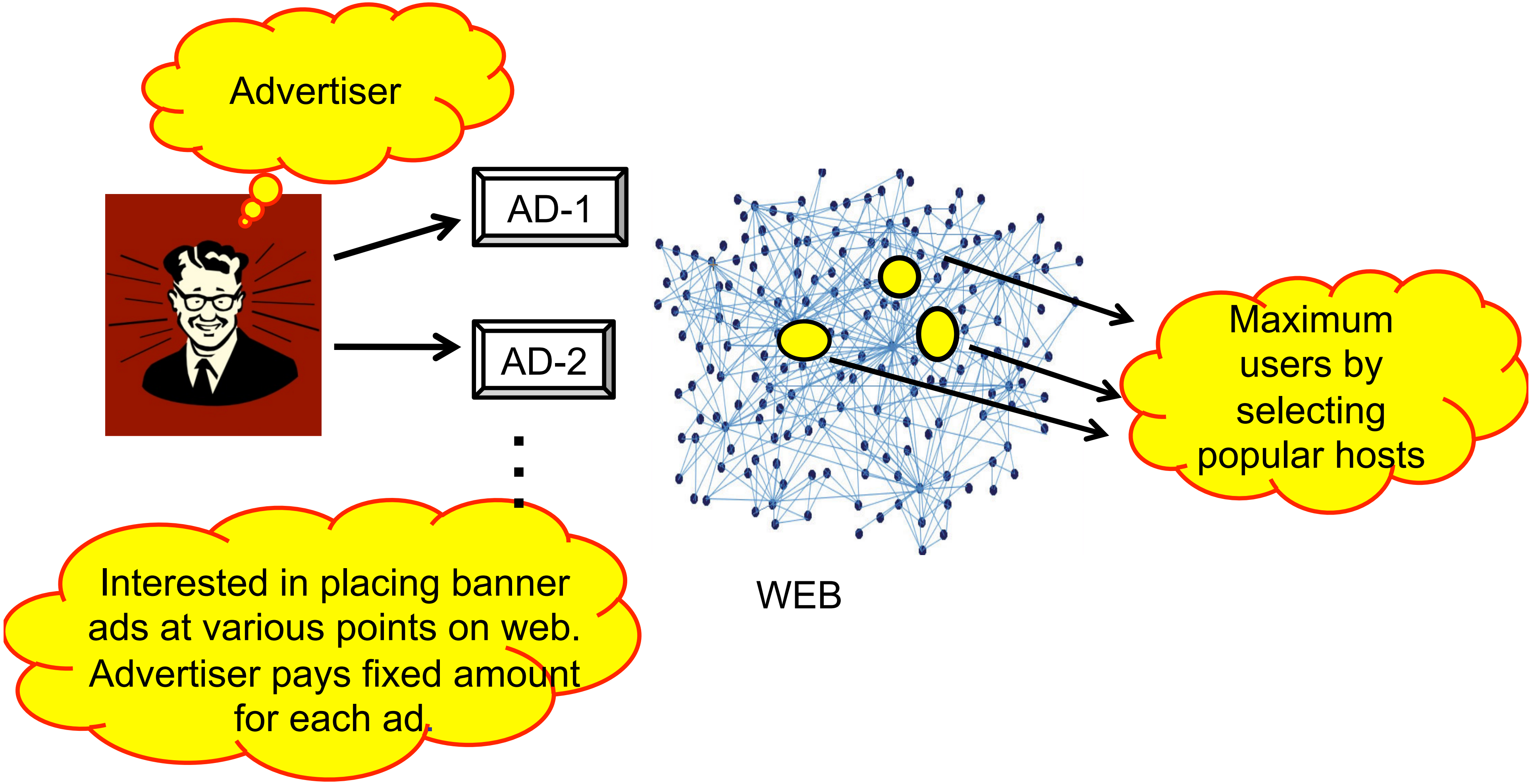
Motivation (1)

Where should a set of charity dropboxes be placed to be available to as many people as possible?



Charity Boxes

Motivation (2)



Advertiser

AD-1

AD-2

⋮

WEB

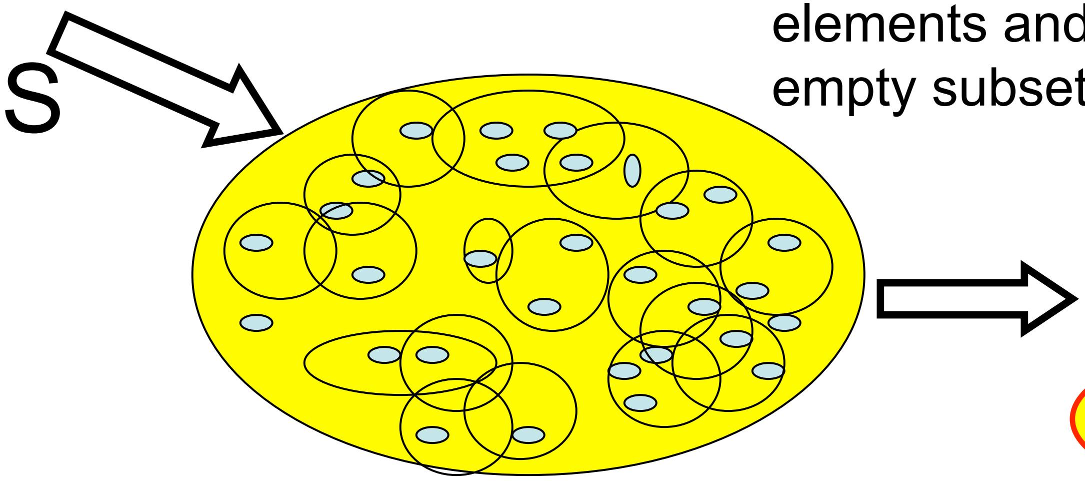
Maximum users by selecting popular hosts

Interested in placing banner ads at various points on web. Advertiser pays fixed amount for each ad.

Problem Setting

- Select k sets from a family of subsets of a universe.
- Union is as large as possible.

X= {1, 2,…,N} be a universe of n elements and |S| be a family of non-empty subsets of X

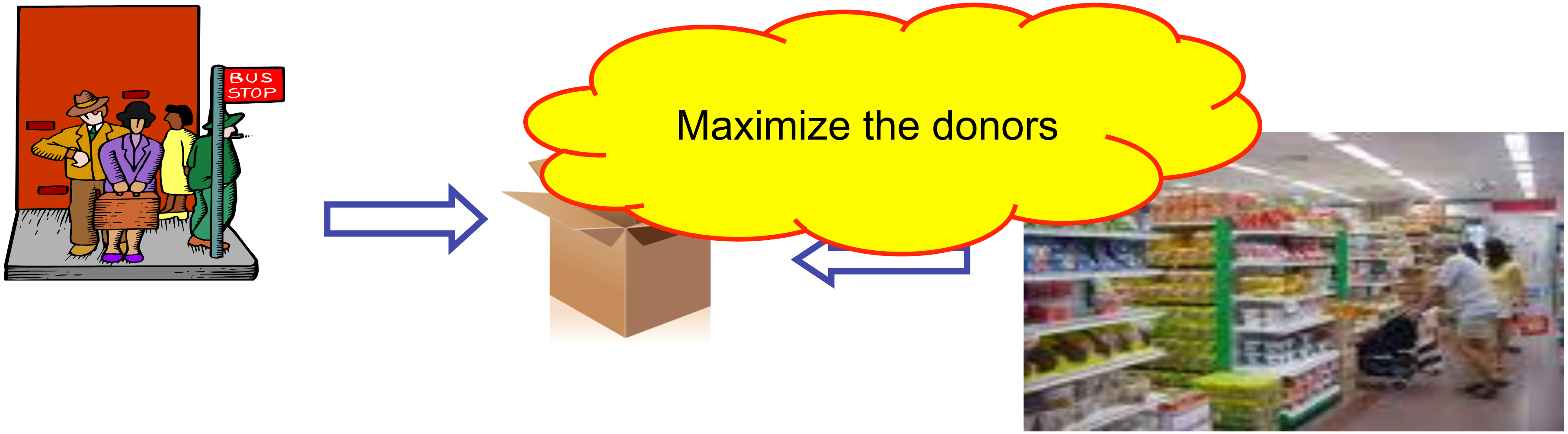


|S| = m sets

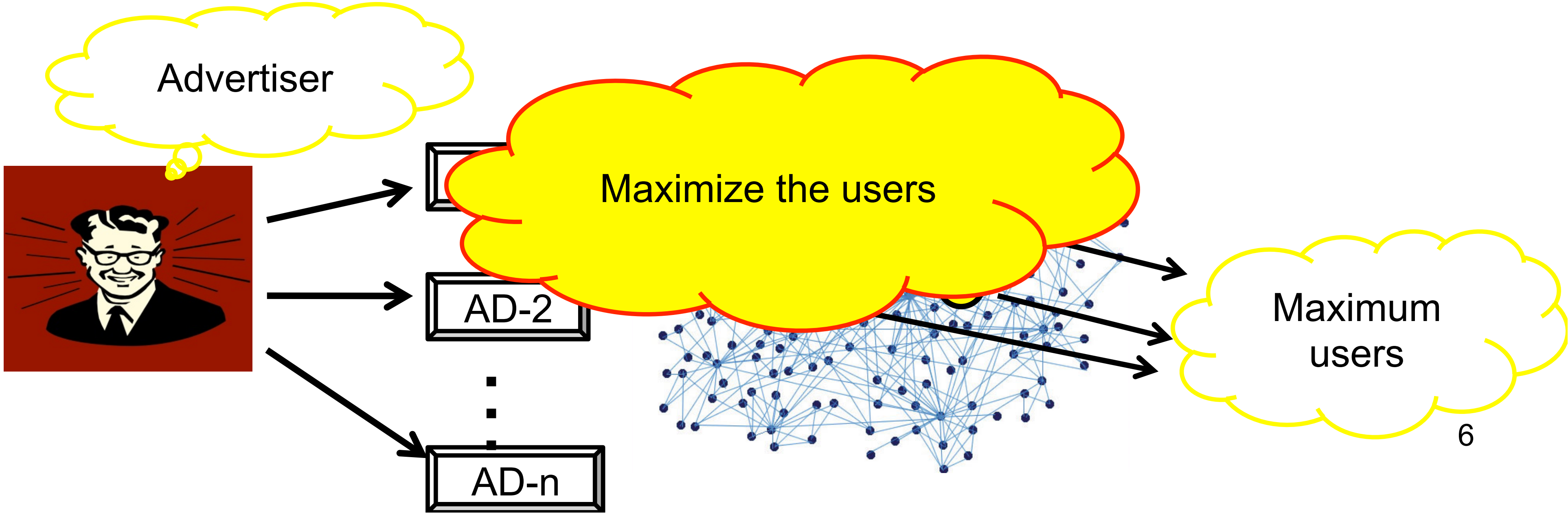
|X| =n elements

k sets ?

Choose a subset of S such that they cover max number of elements in X.



Maximize the donors



Advertiser

AD-2

AD-n

Maximize the users

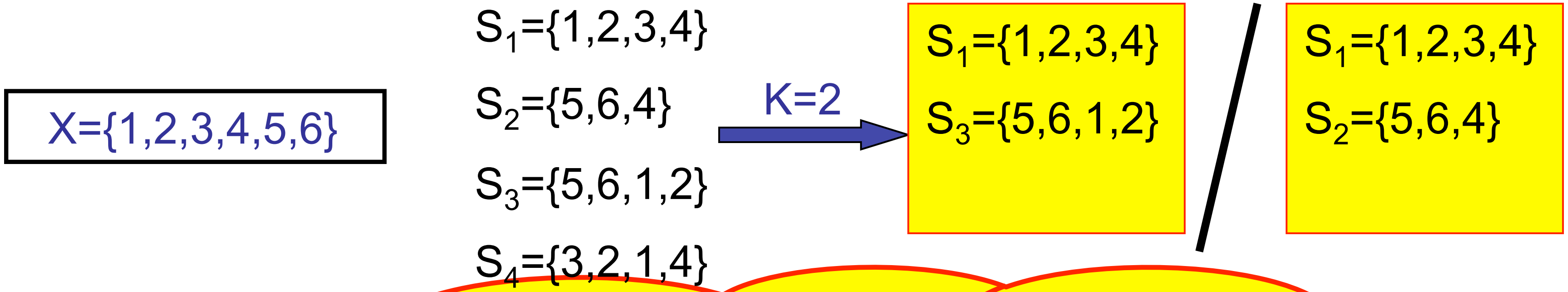
Maximum users

Outline

- Motivation
- Introduction
- Classical Approach : Greedy
- Proposed Algorithm: M_R Greedy
- Possible extension
- Experiments
- Weaknesses
- Conclusion

Formal Definition of Max k cover

Given an integer k >0, S*$\subseteq$S is a max k-cover if |S*| = k and the coverage of S* is maximized over all subsets of S of size k.



$X=\{1,2,3,4,5,6\}$

$S_1=\{1,2,3,4\}$

$S_2=\{5,6,4\}$

$S_3=\{5,6,1,2\}$

$S_4=\{3,2,1,4\}$

K=2

$S_1=\{1,2,3,4\}$

$S_3=\{5,6,1,2\}$

/

$S_1=\{1,2,3,4\}$

$S_2=\{5,6,4\}$

Finding the optimal solution is NP-hard.

So we focus on approximation algorithms !

α Approximation Algorithm

- Polynomial time, guaranteed to find "near optimal" solutions for every input.
- Suppose , I have a input set of 100 elements.
  - Optimal solution contains 80 elements
  - Let α =0.5
  - Approximate solution says…
  - Approx $\geq \alpha$ .optimal
  - In this case, approx : more than 40 elements.

α -Approximate k- Cover

For $\alpha > 0$, a set $S' \subseteq S$, $|S'| \leq k$, is an α approximate max k-cover if for any max k-cover S^*, $\text{cov}(S') \geq \alpha.\text{cov}(S^*)$.

Looking for a approximate algorithm to solve Max K cover problem

One approach to solve max k cover problem …

Use Greedy algorithm !

It achieves constant factor approximation to MAX K-COVER, 1-1/e ~ .63

Outline

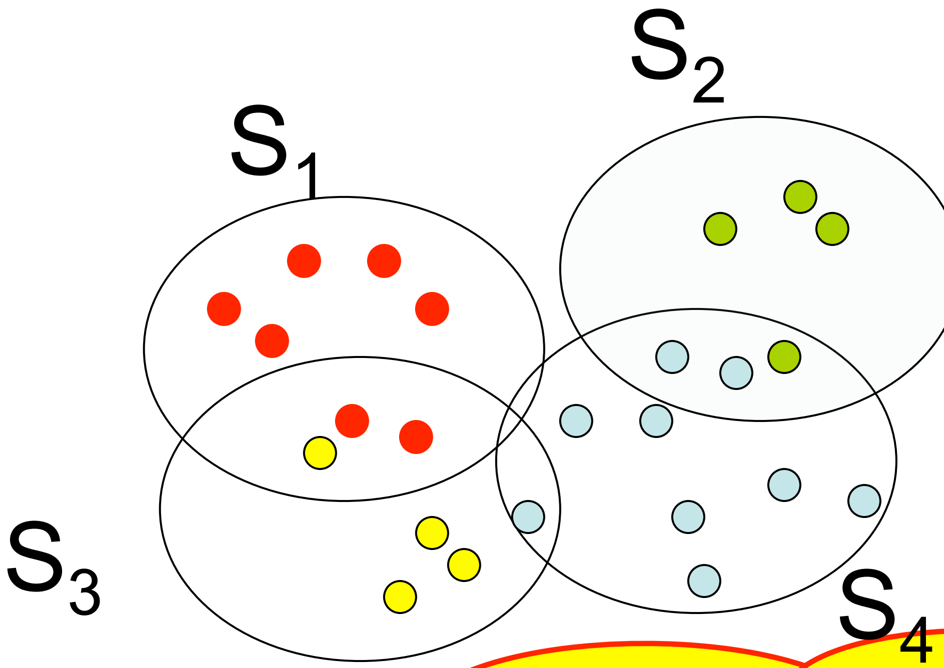
- Motivation
- Introduction
- Classical Approach : Greedy
- Proposed Algorithm: M_R Greedy
- Possible extension
- Experiments
- Weaknesses
- Conclusion

Greedy Algorithm

- When we have a choice to make, make the one that looks best ***right now***.
- Make a **locally optimal choice** in hope of getting a **globally optimal solution**.

```
Require: $S_1$; ……, $S_m$, and an integer k
while $k > 0$ do
    Let S be a set of maximum cardinality
    Output S
    Remove S and all elements of S from other remaining sets
    $k = k - 1$
```

Greedy Algorithm

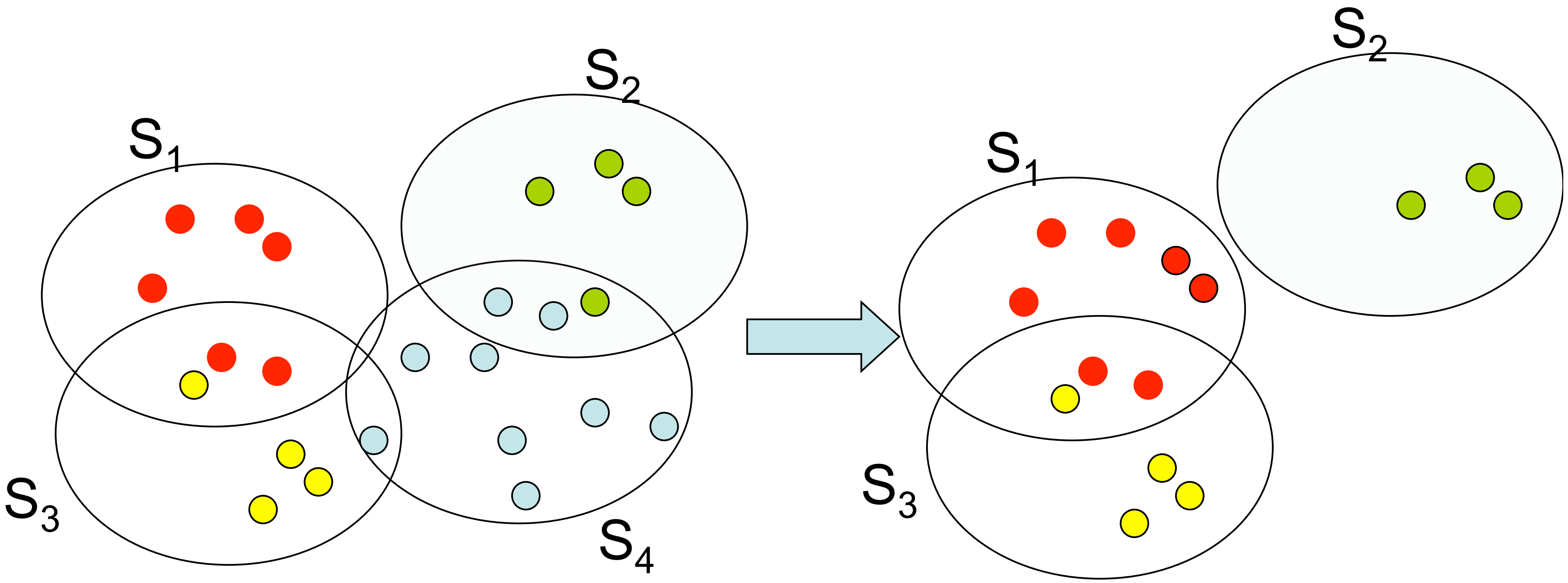


K=2

Step 1: Output the set which has maximum cardinality.

Here S_4, Solution Set C ={S_4}

Greedy Algorithm



Step 2: Remove S_4 and all elements of S_4 from other remaining sets, So next set to be considered is S_1,

Solution set C= $\{S_4, S_1\}$

Greedy Algorithm

- Sequential, it satisfies prefix optimality property.
- What is that?

Greedy algorithm can be easily extended to output a total ordering of the input sets S_1, … S_m, with the guarantee that the prefix of length k, for each k, of this ordering will be a (1-1/e)-approximation to the corresponding Max-k-Cover.

- Drawbacks
  - Bookkeeping is expensive if value of k is ve
  - For disk resident datasets, Greedy is not a s approach.

Updates expensive!

Outline

- Motivation
- Introduction
- Classical Approach : Greedy
- Proposed Algorithm: M_R Greedy
- Possible extension
- Experiments
- Weaknesses
- Conclusion

Map-Reduce Model

- Computations are distributed across several processors.
- Split as a sequence of map and reduce jobs.

- map
  - maps an input (key,value) pair to a **list** of intermediate key-value pairs
  - map (k, v) -> list(k, v)
- reduce
  - takes as input a key and a **list** of values for that key
  - maps the input to a **list** of values
  - reduce (k, list(v)) -> list(v)

Example

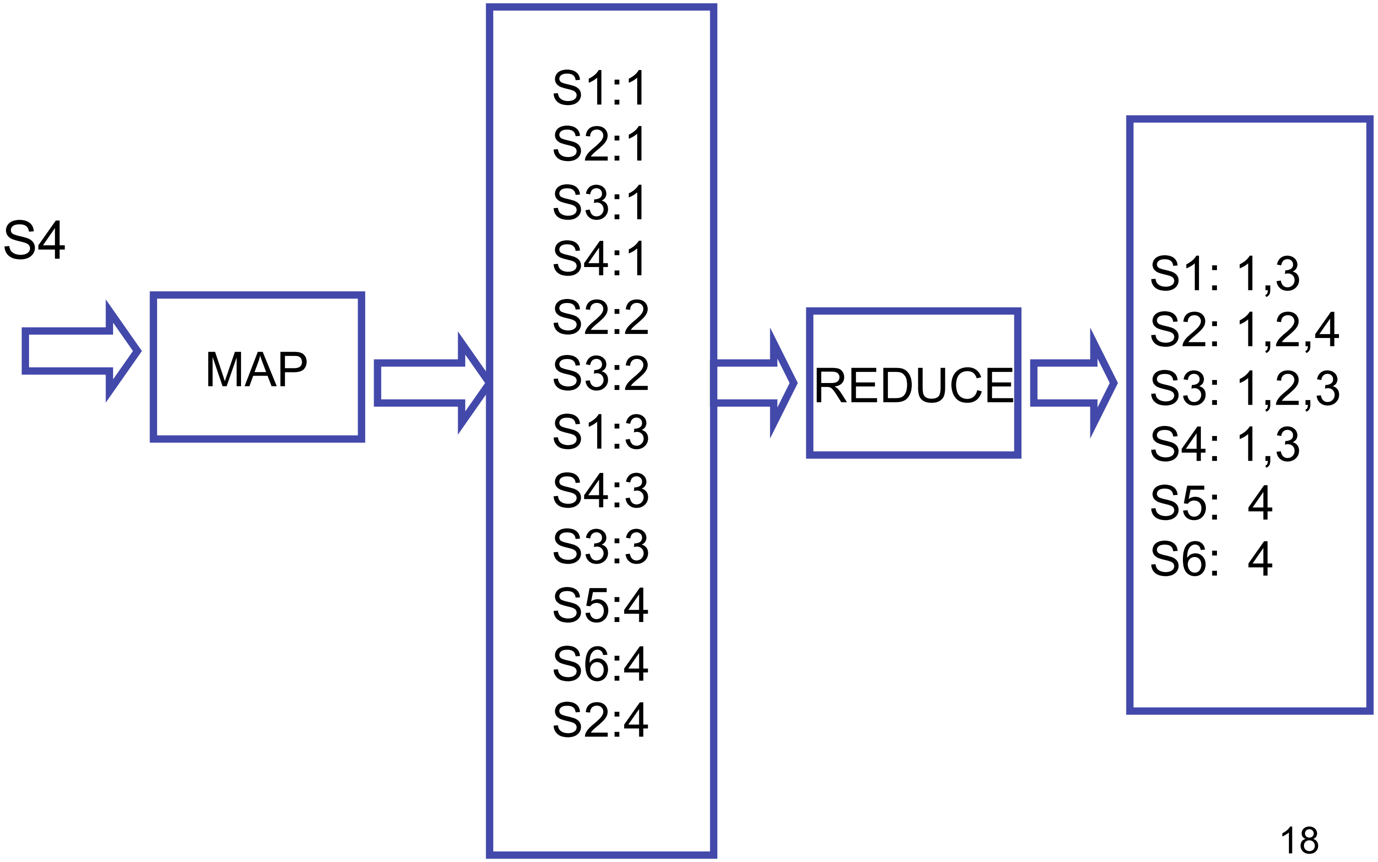
- Transposing of an adjacency list ?
- Key-element
- Value-set
- Input set

1: S1,S2,S3,S4

2: S2,S3

3: S1,S4,S3

4: S5,S6,S2



MAP

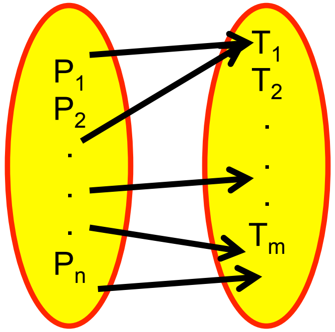
S1:1
S2:1
S3:1
S4:1
S2:2
S3:2
S1:3
S4:3
S3:3
S5:4
S6:4
S2:4

REDUCE

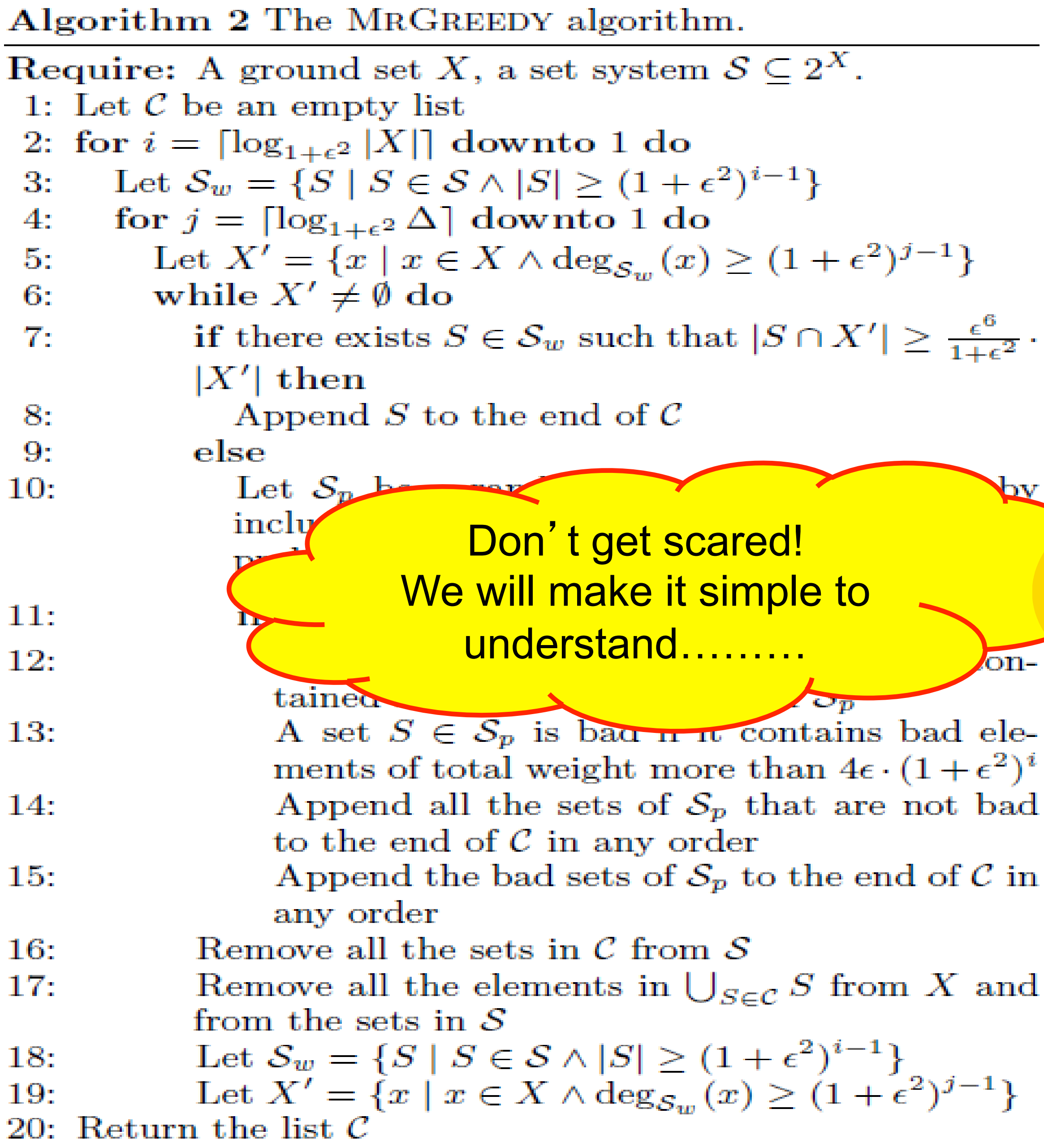
S1: 1,3
S2: 1,2,4
S3: 1,2,3
S4: 1,3
S5: 4
S6: 4

MRGreedy

- No k sequential choices.
- Idea is to add multiple sets to the solution in parallel.
- It also satisfies prefix optimality property same as Greedy.
- Run on MapReduce Framework
  - No need to keep datasets in main memory
  - update element set memberships. (edges)



MRGreedy Algorithm

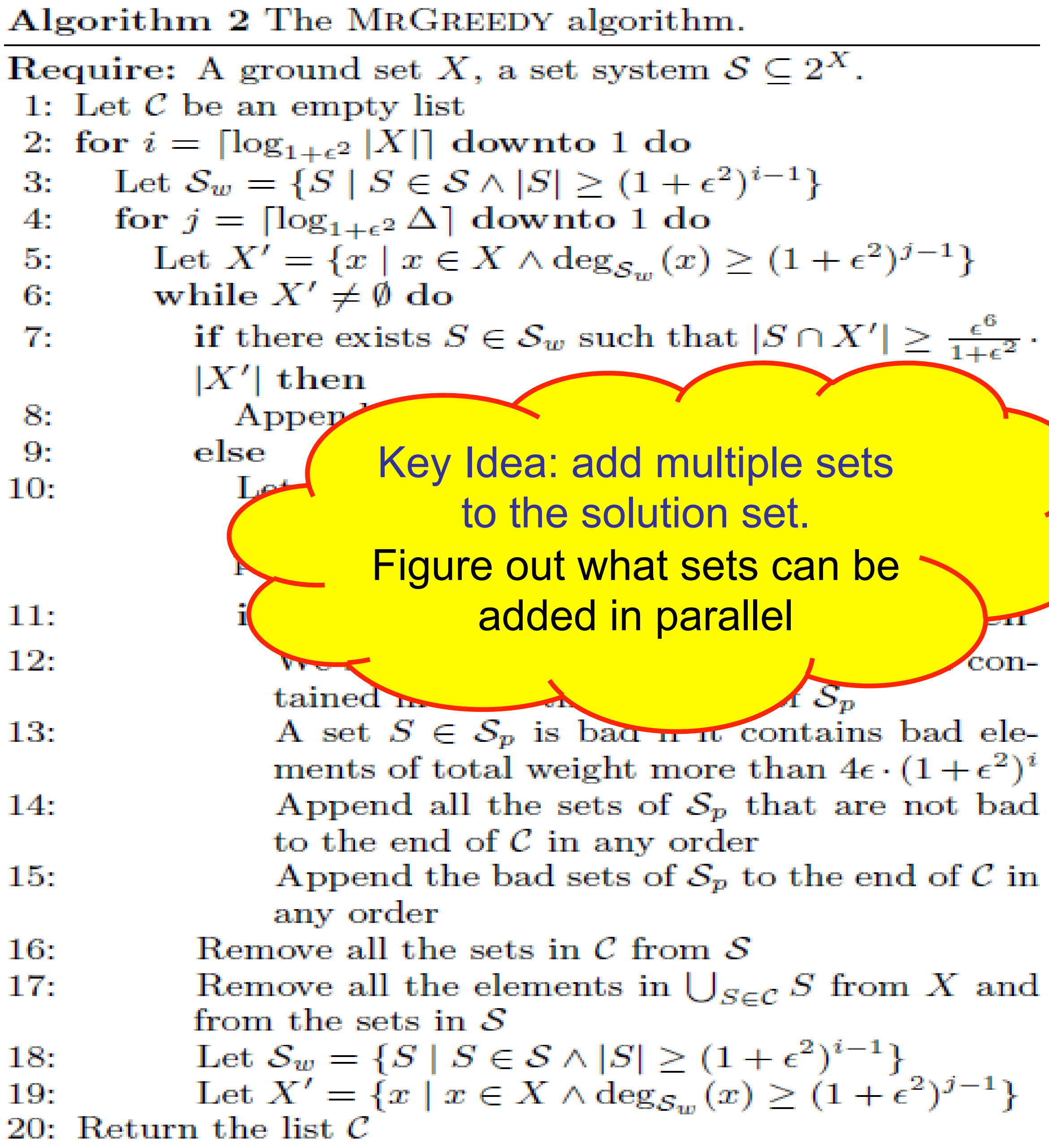


Algorithm 2 The MRGREEDY algorithm.

```
Require: A ground set $X$, a set system $\mathcal{S} \subseteq 2^X$.
Let $\mathcal{C}$ be an empty list
for $i = \lceil \log_{1+\epsilon^2} |X| \rceil$ downto 1 do
  Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
  for $j = \lceil \log_{1+\epsilon^2} \Delta \rceil$ downto 1 do
    Let $X' = \{x \mid x \in X \wedge \deg_{\mathcal{S}_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
    while $X' \neq \emptyset$ do
      if there exists $S \in \mathcal{S}_w$ such that $|S \cap X'| \geq \frac{\epsilon^6}{1+\epsilon^2} \cdot |X'|$ then
        Append $S$ to the end of $\mathcal{C}$
      else
        Let $\mathcal{S}_p$ be ... by
        inclu...
        p...
        ...
        ... con-
          tained ... $\mathcal{S}_p$
          A set $S \in \mathcal{S}_p$ is bad ... contains bad elements of total weight more than $4\epsilon \cdot (1+\epsilon^2)^i$
          Append all the sets of $\mathcal{S}_p$ that are not bad to the end of $\mathcal{C}$ in any order
          Append the bad sets of $\mathcal{S}_p$ to the end of $\mathcal{C}$ in any order
      Remove all the sets in $\mathcal{C}$ from $\mathcal{S}$
      Remove all the elements in $\bigcup_{S \in \mathcal{C}} S$ from $X$ and from the sets in $\mathcal{S}$
      Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
      Let $X' = \{x \mid x \in X \wedge \deg_{\mathcal{S}_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
Return the list $\mathcal{C}$
```

Don’t get scared!
We will make it simple to understand………

MRGreedy Algorithm



Algorithm 2 The MRGREEDY algorithm.

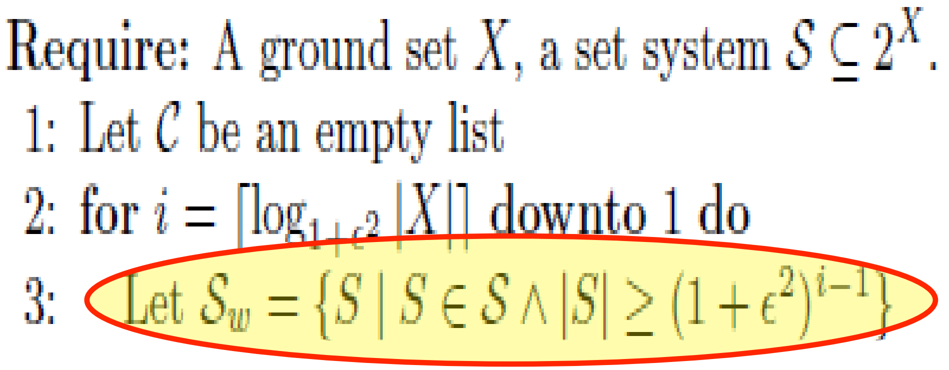
```
Require: A ground set $X$, a set system $\mathcal{S} \subseteq 2^X$.
Let $\mathcal{C}$ be an empty list
for $i = \lceil \log_{1+\epsilon^2} |X| \rceil$ downto 1 do
    Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
    for $j = \lceil \log_{1+\epsilon^2} \Delta \rceil$ downto 1 do
        Let $X' = \{x \mid x \in X \wedge \deg_{\mathcal{S}_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
        while $X' \neq \emptyset$ do
            if there exists $S \in \mathcal{S}_w$ such that $|S \cap X'| \geq \frac{\epsilon^6}{1+\epsilon^2} \cdot |X'|$ then
                Appen
            else
                L
                
                
            i                                                en
                W                                            con-
                tained                                   $\mathcal{S}_p$
                A set $S \in \mathcal{S}_p$ is bad              contains bad elements of total weight more than $4\epsilon \cdot (1+\epsilon^2)^i$
                Append all the sets of $\mathcal{S}_p$ that are not bad to the end of $\mathcal{C}$ in any order
                Append the bad sets of $\mathcal{S}_p$ to the end of $\mathcal{C}$ in any order
            Remove all the sets in $\mathcal{C}$ from $\mathcal{S}$
            Remove all the elements in $\bigcup_{S \in \mathcal{C}} S$ from $X$ and from the sets in $\mathcal{S}$
            Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
            Let $X' = \{x \mid x \in X \wedge \deg_{\mathcal{S}_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
Return the list $\mathcal{C}$
```

Key Idea: add multiple sets to the solution set.

Figure out what sets can be added in parallel

Learn Algorithm in Steps (1)

- Step 1: Consider a empty list C,
  - We have a ground set X={1, 2…..n}, so total number of sets possible 2^x

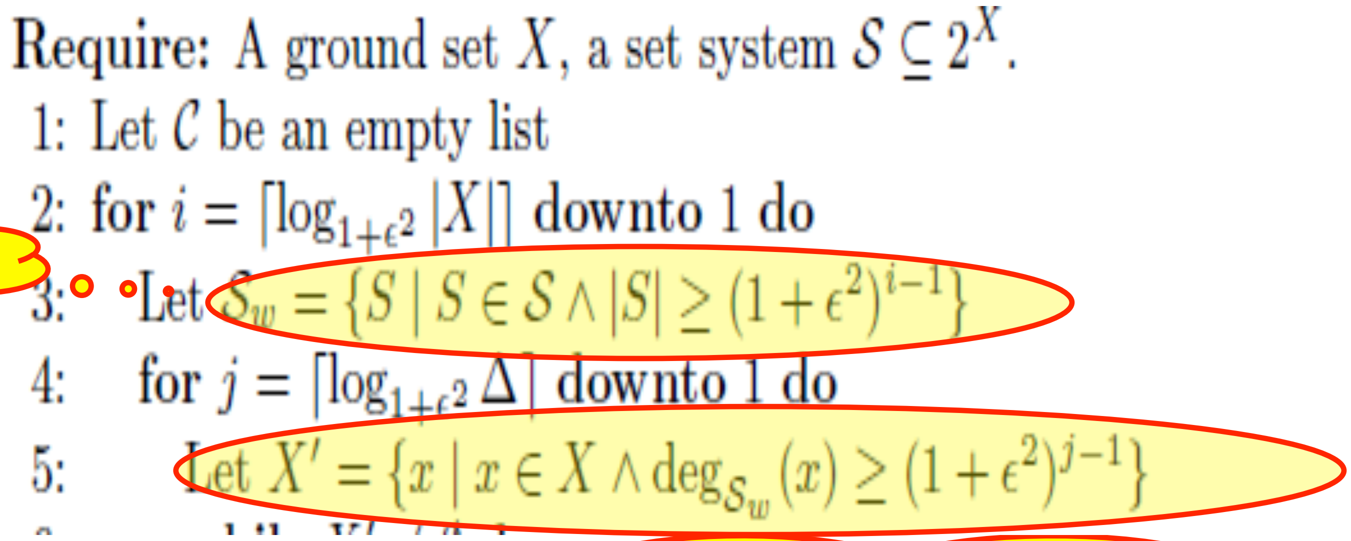


```
Require: A ground set $X$, a set system $\mathcal{S} \subseteq 2^X$.
Let $C$ be an empty list
for $i = \lceil \log_{1+\epsilon^2} |X| \rceil$ downto 1 do
    Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
```

Consider i as set to some constant and select a set known as S_w which has cardinality more than some constant.

Learn Algorithm in Steps (2)

- Step 2:



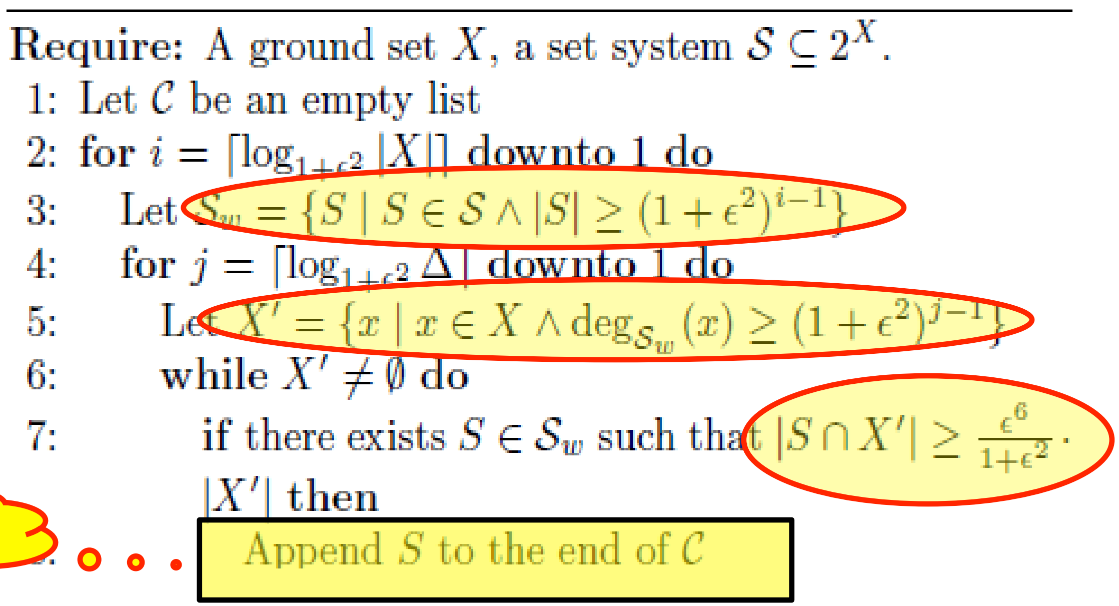
```
Require: A ground set $X$, a set system $\mathcal{S} \subseteq 2^X$.
Let $C$ be an empty list
for $i = \lceil \log_{1+\epsilon^2} |X| \rceil$ downto 1 do
    Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
    for $j = \lceil \log_{1+\epsilon^2} \Delta \rceil$ downto 1 do
        Let $X' = \{x \mid x \in X \wedge \deg_{\mathcal{S}_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
```

Degree(x)?

Δ is maximum degree of an element, we get X' elements from set S_w which has degree more than some constant

Learn Algorithm in Steps (3)

- Step 3:



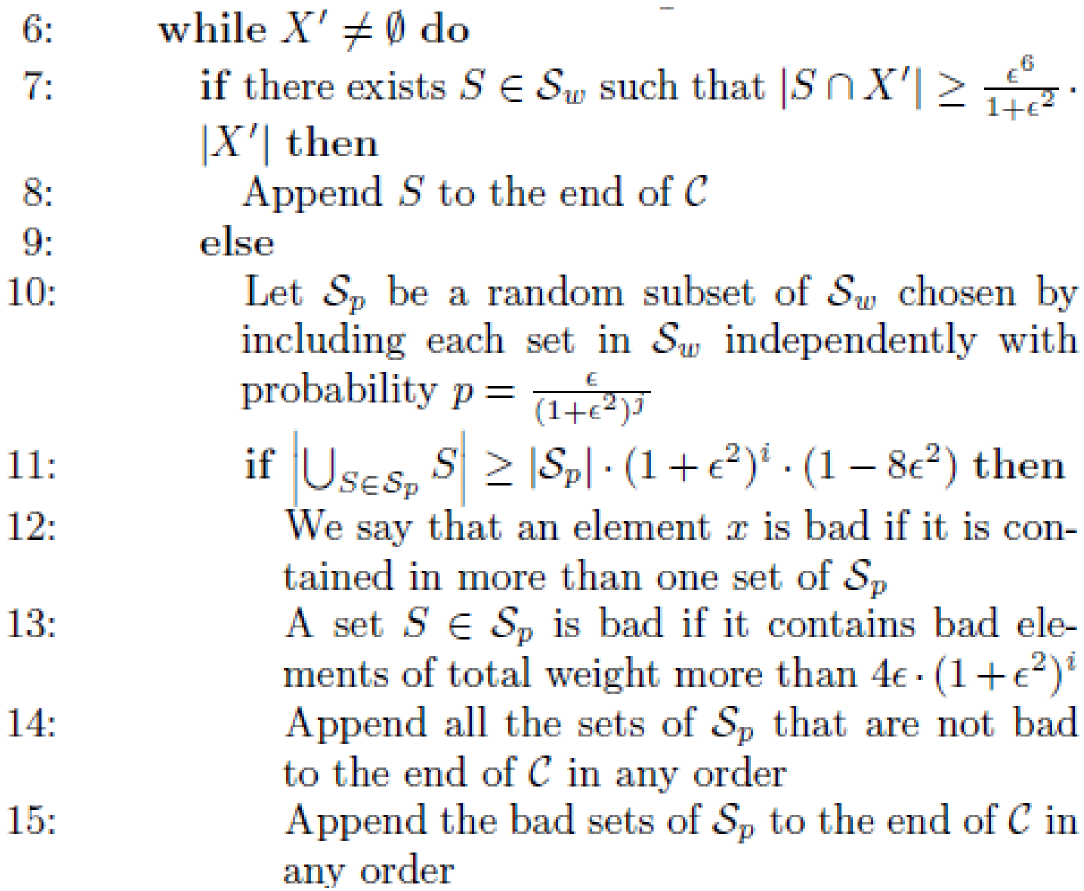
```
Require: A ground set $X$, a set system $\mathcal{S} \subseteq 2^X$.
Let $\mathcal{C}$ be an empty list
for $i = \lceil \log_{1+\epsilon^2} |X| \rceil$ downto 1 do
    Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
    for $j = \lceil \log_{1+\epsilon^2} \Delta \rceil$ downto 1 do
        Let $X' = \{x \mid x \in X \wedge \deg_{\mathcal{S}_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
        while $X' \neq \emptyset$ do
            if there exists $S \in \mathcal{S}_w$ such that $|S \cap X'| \geq \frac{\epsilon^6}{1+\epsilon^2} \cdot |X'|$ then
                Append $S$ to the end of $\mathcal{C}$
```

Interesting!

Start a while loop until X' is empty and check existence of a set such that intersection of X' with S is again greater than some constant value.

Learn Algorithm in Steps (4)

- Step 4:

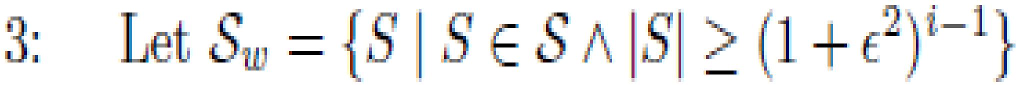


```
while $X' \neq \emptyset$ do
  if there exists $S \in \mathcal{S}_w$ such that $|S \cap X'| \geq \frac{\epsilon^6}{1+\epsilon^2} \cdot |X'|$ then
    Append $S$ to the end of $\mathcal{C}$
  else
    Let $\mathcal{S}_p$ be a random subset of $\mathcal{S}_w$ chosen by including each set in $\mathcal{S}_w$ independently with probability $p = \frac{\epsilon}{(1+\epsilon^2)^j}$
    if $\left|\bigcup_{S \in \mathcal{S}_p} S\right| \geq |\mathcal{S}_p| \cdot (1+\epsilon^2)^i \cdot (1-8\epsilon^2)$ then
      We say that an element $x$ is bad if it is contained in more than one set of $\mathcal{S}_p$
      A set $S \in \mathcal{S}_p$ is bad if it contains bad elements of total weight more than $4\epsilon \cdot (1+\epsilon^2)^i$
      Append all the sets of $\mathcal{S}_p$ that are not bad to the end of $\mathcal{C}$ in any order
      Append the bad sets of $\mathcal{S}_p$ to the end of $\mathcal{C}$ in any order
```

Choose random subset S_p with certain probability, and from them decide bad and non bad sets and then append them to the list.

Realization in MapReduce

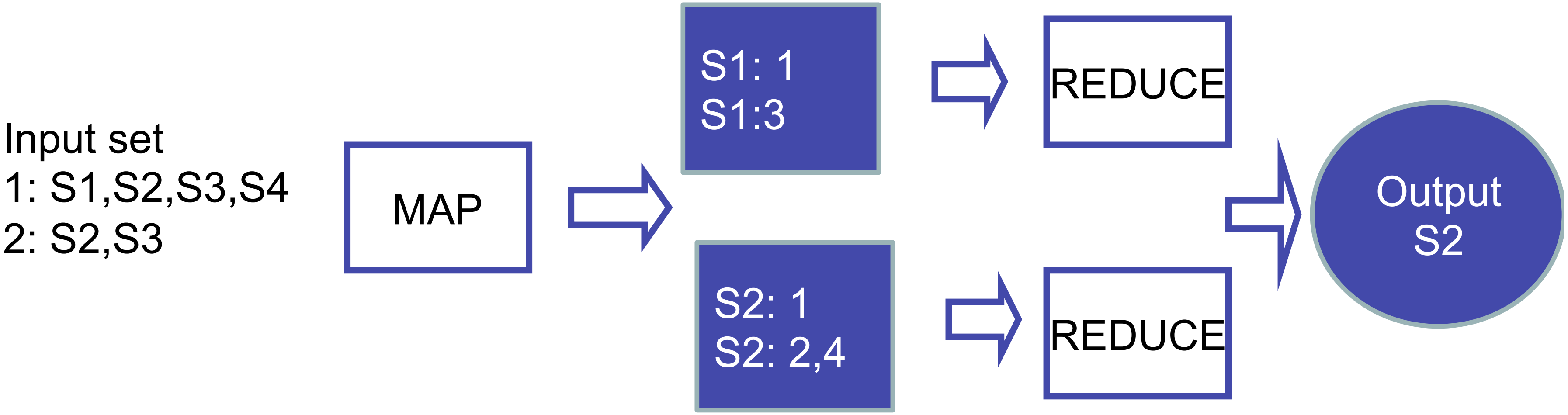
- Some lines of the above algorithm can be realized in MapReduce framework.



```
3:    Let $\mathcal{S}_w = \{S \mid S \in \mathcal{S} \wedge |S| \geq (1+\epsilon^2)^{i-1}\}$
```

1. Map
   1. ForEach e_i
   2. Iterate through List (S)
   3. Emit (S_j, e_i) where S_jЄ S
2. Reduce (S_j, List(e)) where e are elements
   1. If sizeof(List(e)) ≥ constant
   2. Emit (S_j)

Realization in MapReduce



IF COUNT IS MORE THAN OR EQUAL TO 3

Some facts about M_R Greedy

- The approximation guarantee of M_RGreedy is 1-1/e-O(ε).

- Running time O(poly(ε) .log^{3}nm)
  - n is number of elements
  - m is number of sets

- Best of two worlds:
  - nearly matching the performance of Greedy (1-1/e).
  - Algorithm that can be implemented in the scalable Map-Reduce framework.

Outline

- Motivation
- Introduction
- Classical Approach : Greedy
- Proposed Algorithm: M_R Greedy
- Experiments
- Weaknesses
- Conclusion

Experiments: Goal

1. The coverage of M_RGreedy is almost indistinguishable from Greedy and it outperforms Naive for various values of k and for instances with various characteristics.

2. Algorithm exploits and achieves parallelism in practice

3. Feasible to implement M_RGreedy in practice.

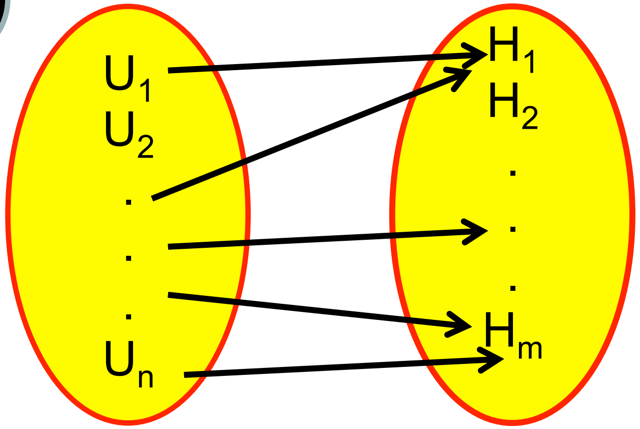
Naive greedy

- It simply sorts the sets by sizes and takes the prefix as solution.
- Suppose you have the following sets and k=3
  - S1={1,2,3,4,5}
  - S2={2,3,4,5}
  - S3={6,7,1}
  - S4={8,9}

Solution set ={S1,S2,S3}

Experiments: 5 Data Sets (1)

1



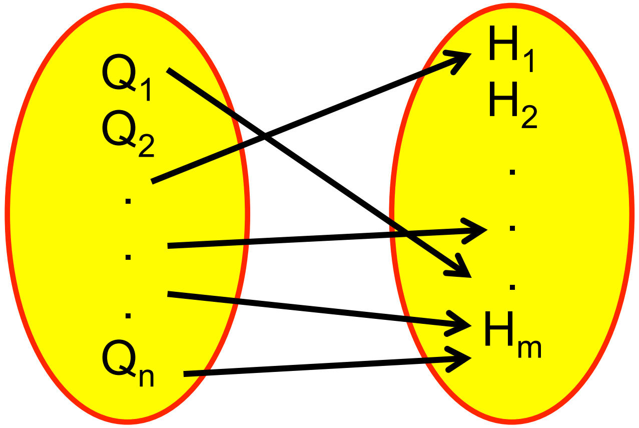
User – Hosts

CP- k hosts visited by maximum users

m=5.64 M
n=2.96 M
E=72.8 M
Δ=2115
w*(S)=1.19M

Edges

2



Query – Hosts

CP- k hosts addresses maximum queries

m=625 K
n=239 K
E=2.8 M
Δ=10
w*(S)=164 K

Experiments: 5 Data Sets (2)

3

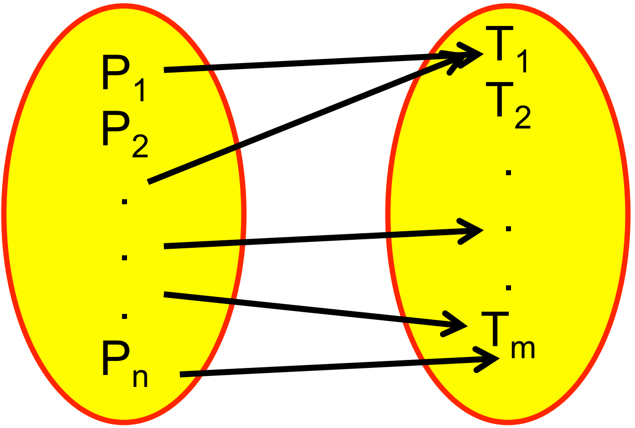


Photo-Tags

CP- k tags used by maximum photos

m=89 K
n=704 K
E=2.7 M
Δ=145
w*(S)=54.3 K

4

Page-ads

m=321 K
n=357 K
E=9.1 M
Δ=24825
w*(S)=164 K

5

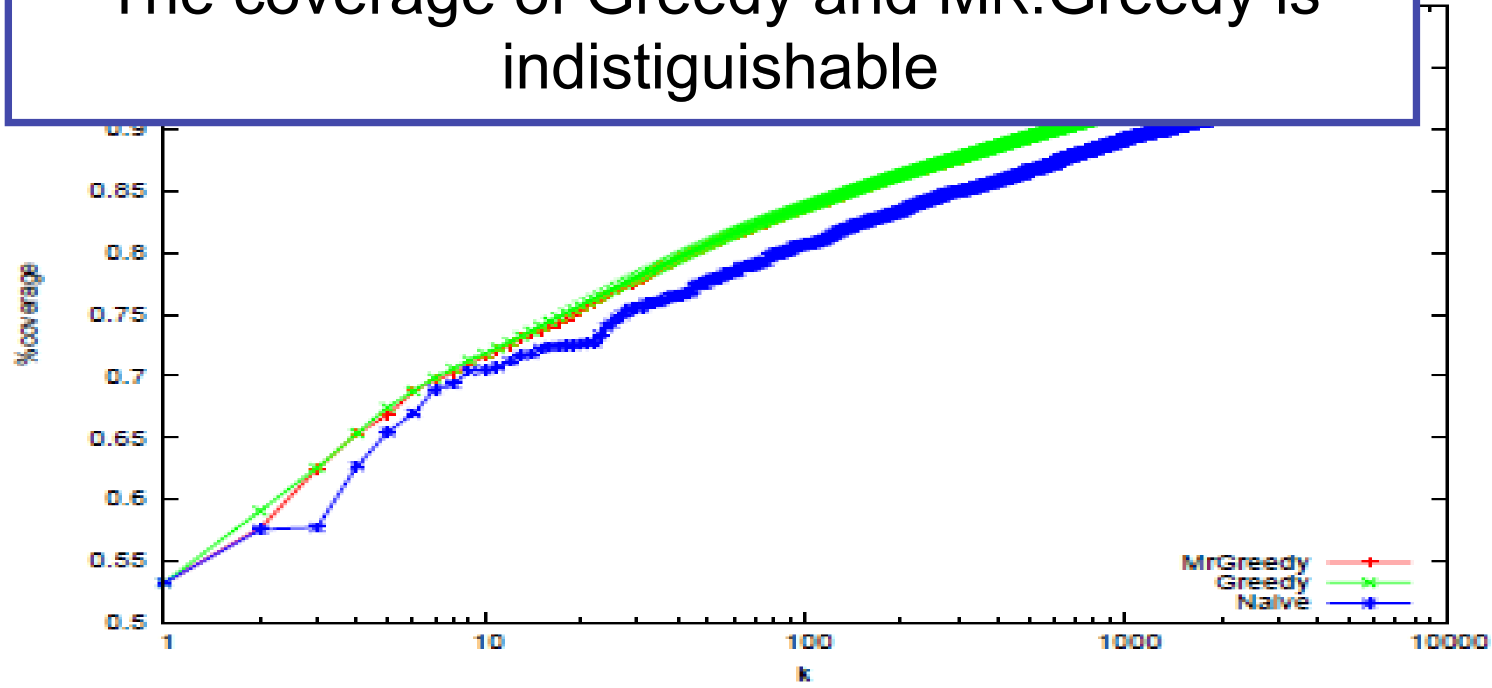
User-queries

m=14.2 M
n=100 K
E=72 M
Δ=5369
w*(S)=21.4 K

Coverage of MRGreedy, Greedy, and Naive on User hosts.

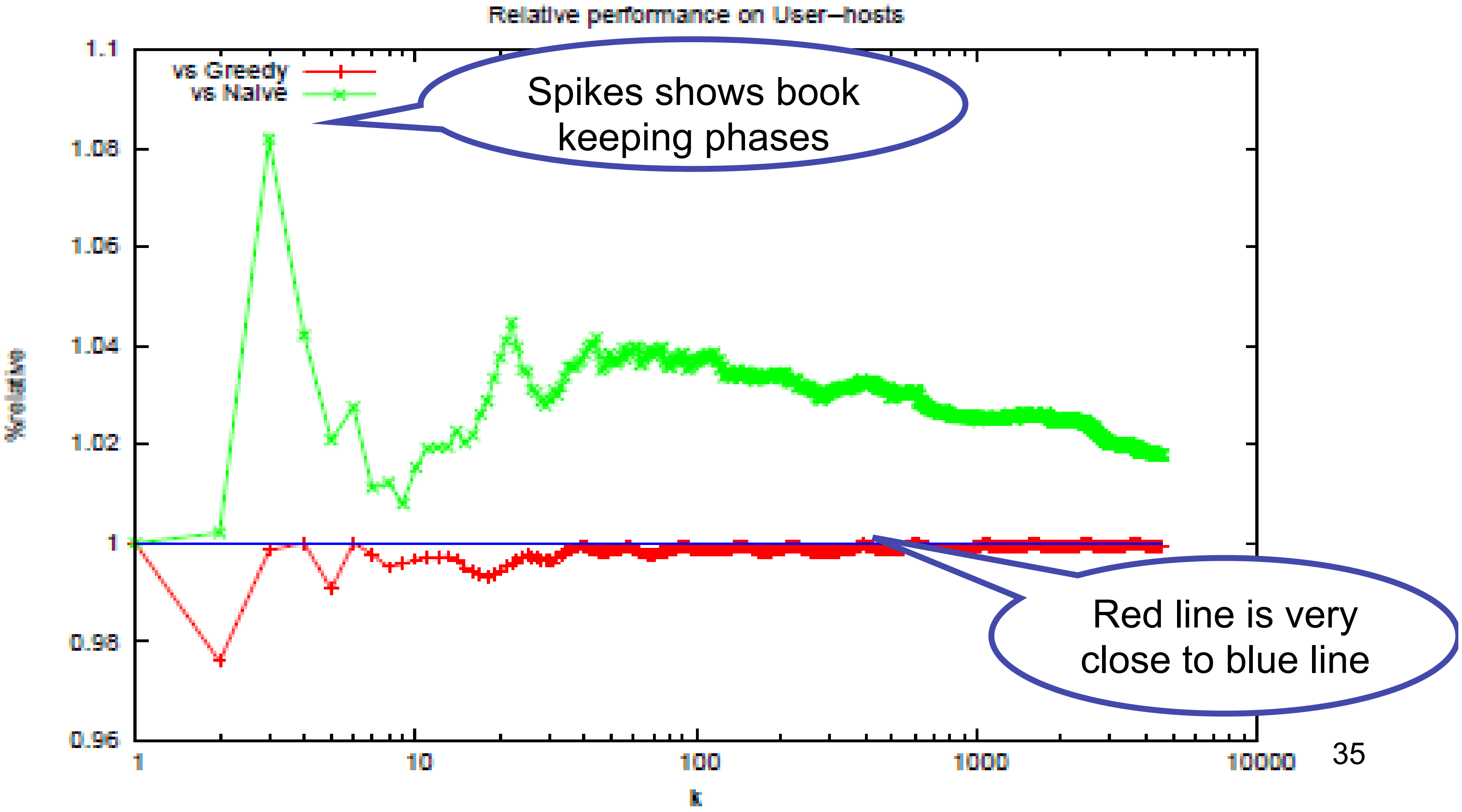
X axis specifies k

Y axis species the fraction of elements covered by prefix of length k



The coverage of Greedy and MR.Greedy is indistiguishable

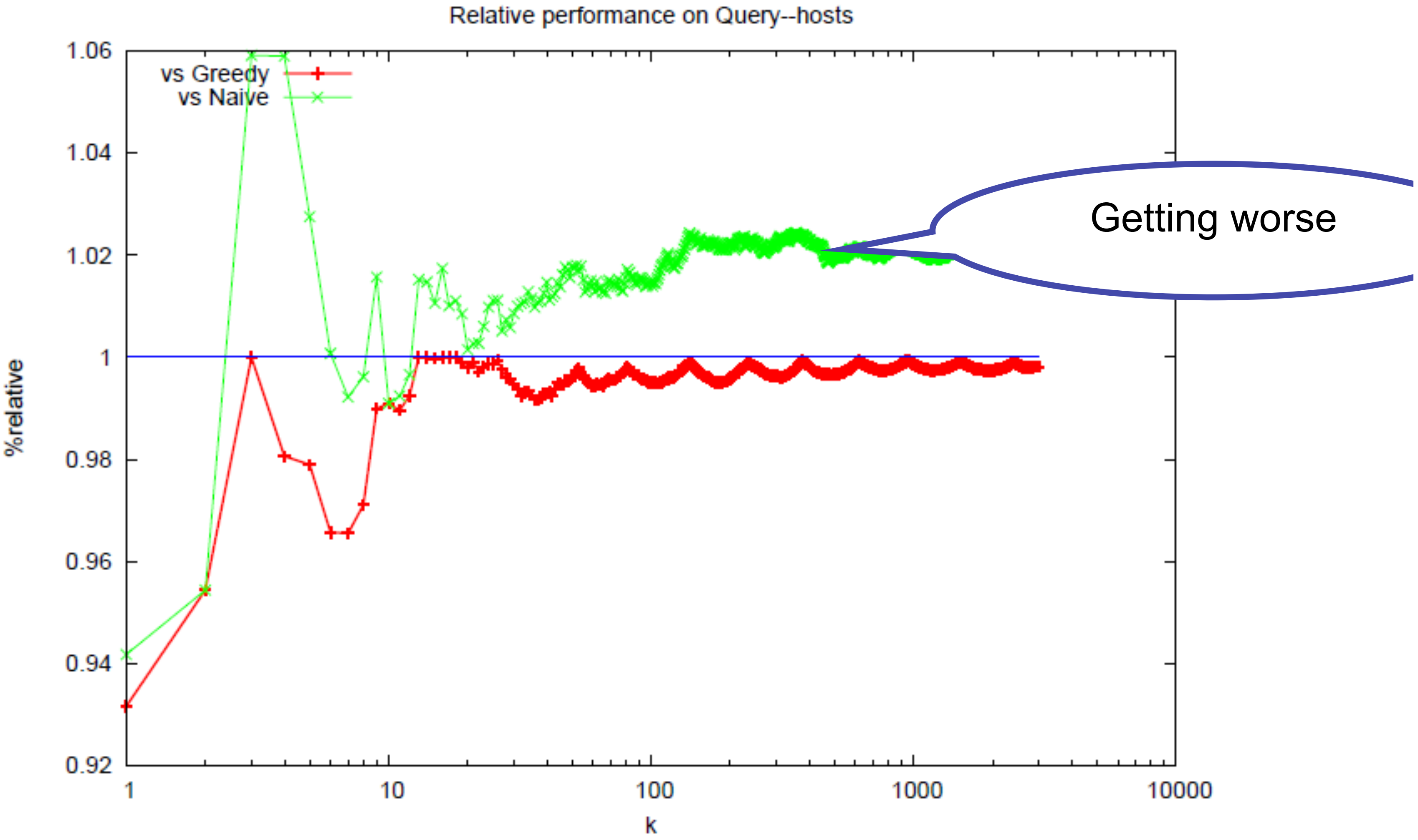
Relative Performances User – Hosts (1)



Spikes shows book keeping phases

Red line is very close to blue line

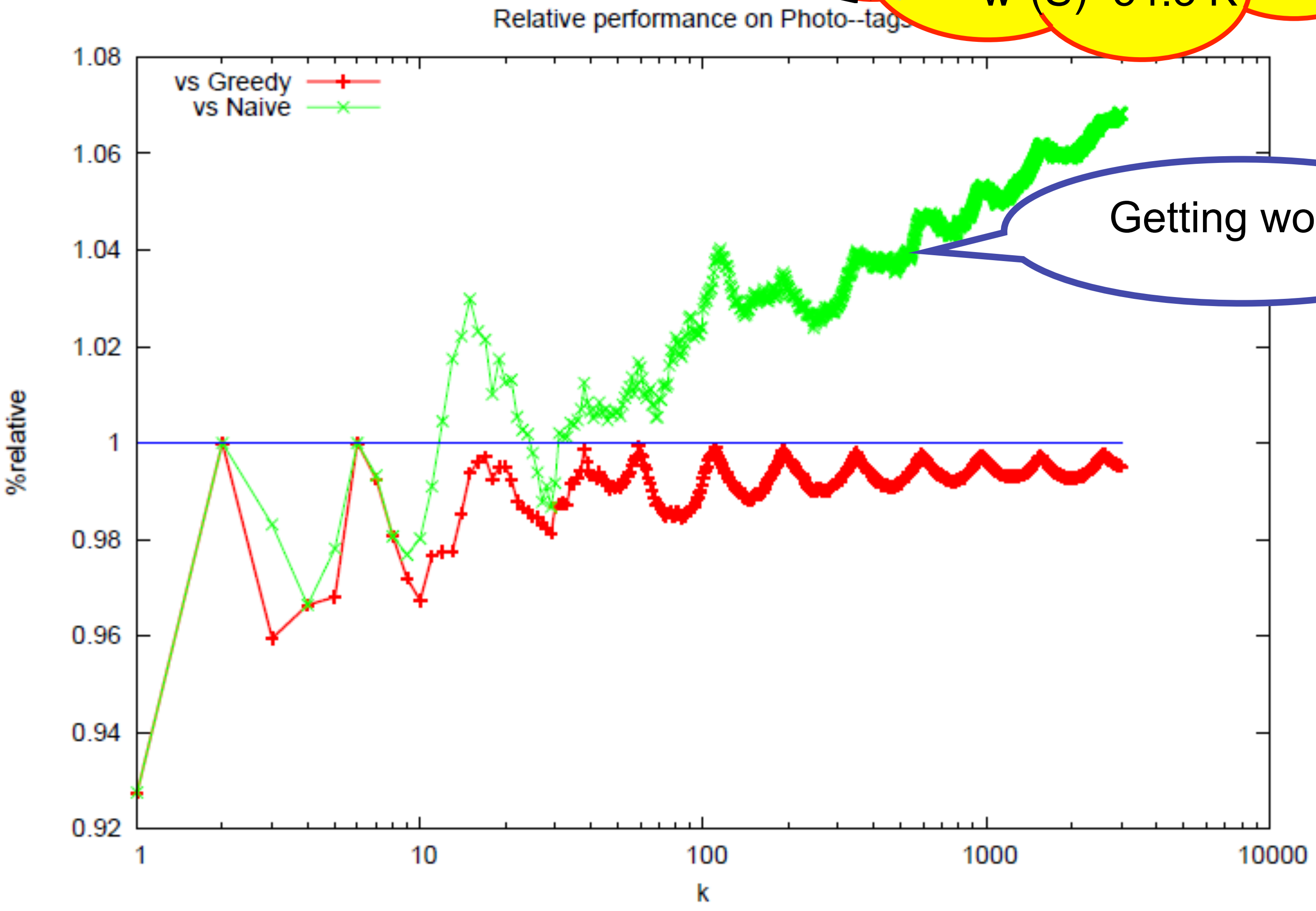
Query-Hosts (2)



Getting worse

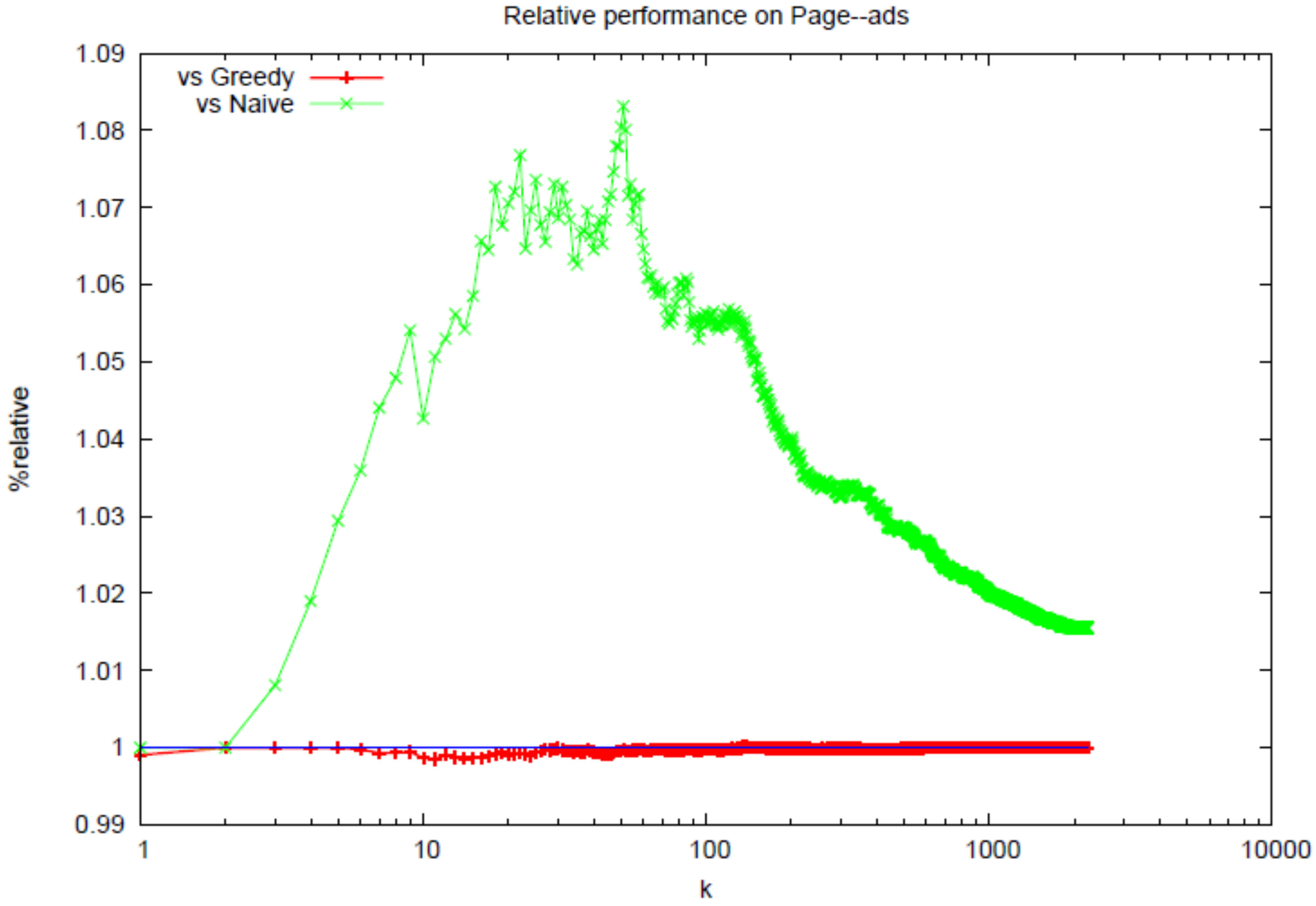
Photo-Tag

m=89 K
n=704 K
E=2.7 M
Δ=145
w*(S)=54.3 K

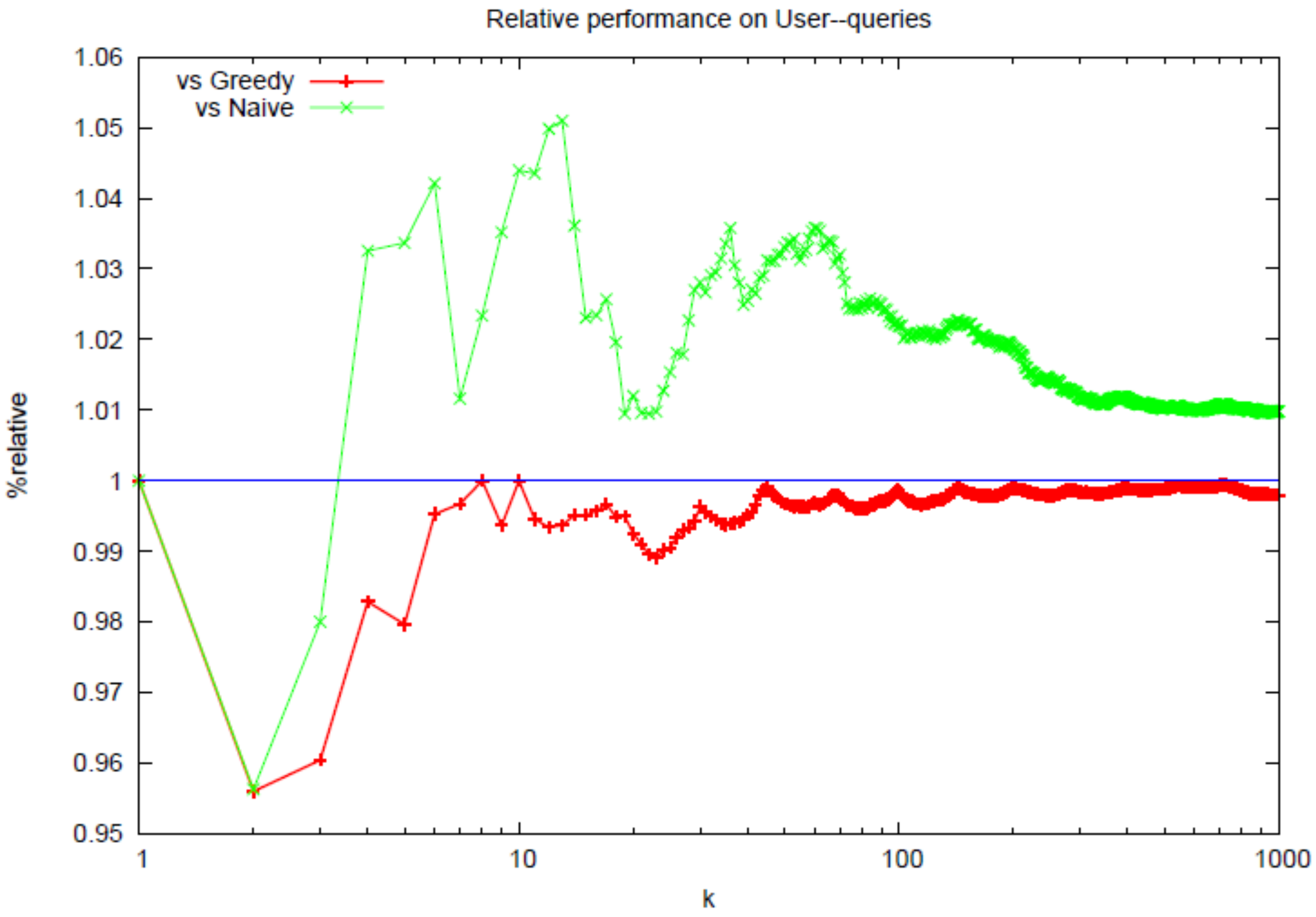


Getting worse

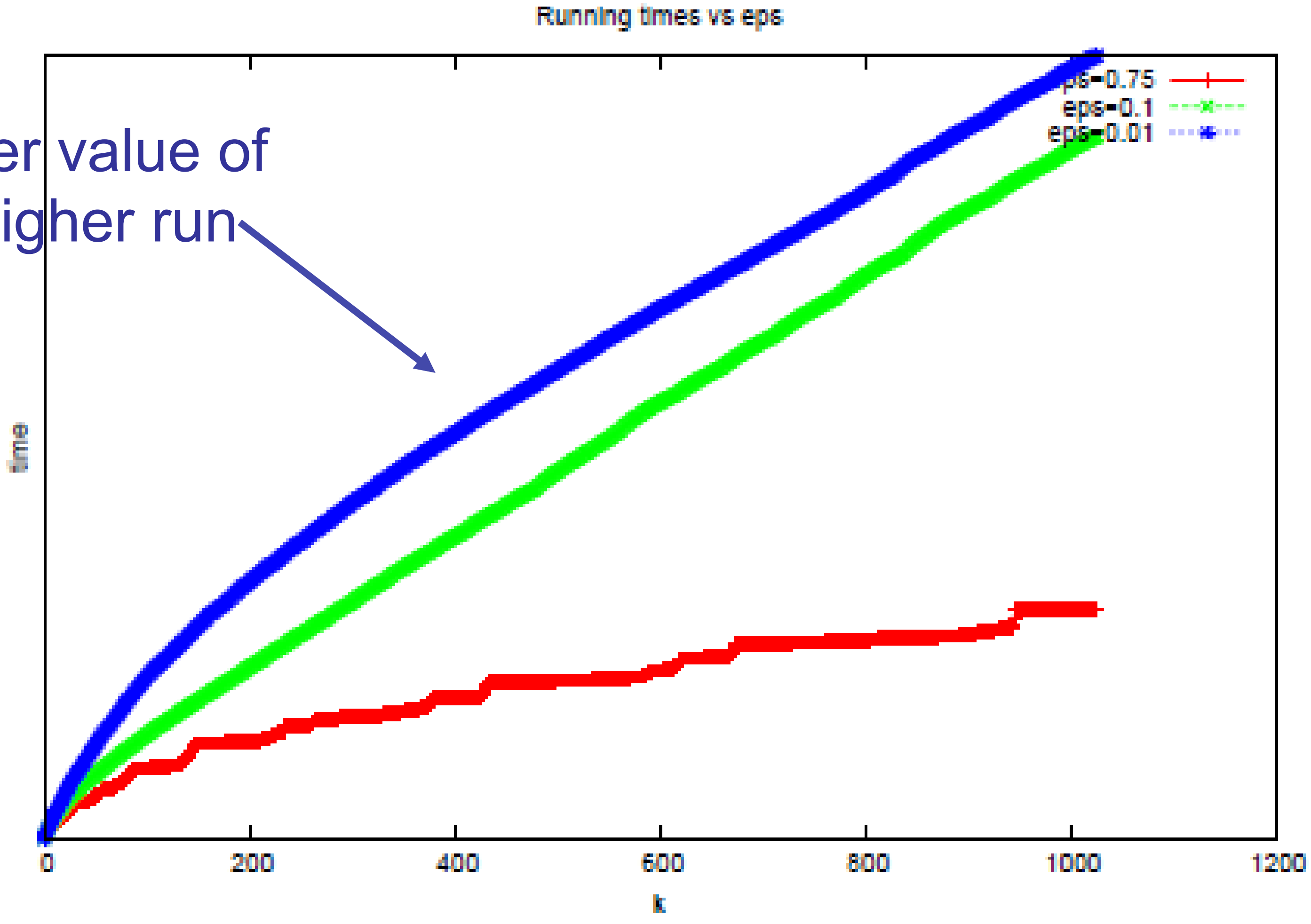
Page-Ads (4)


Relative performance on Page--ads
vs Greedy
vs Naive
%relative
1.09
1.08
1.07
1.06
1.05
1.04
1.03
1.02
1.01
1
0.99
1
10
100
1000
10000
k

User Queries (5)



Effect of epsilon



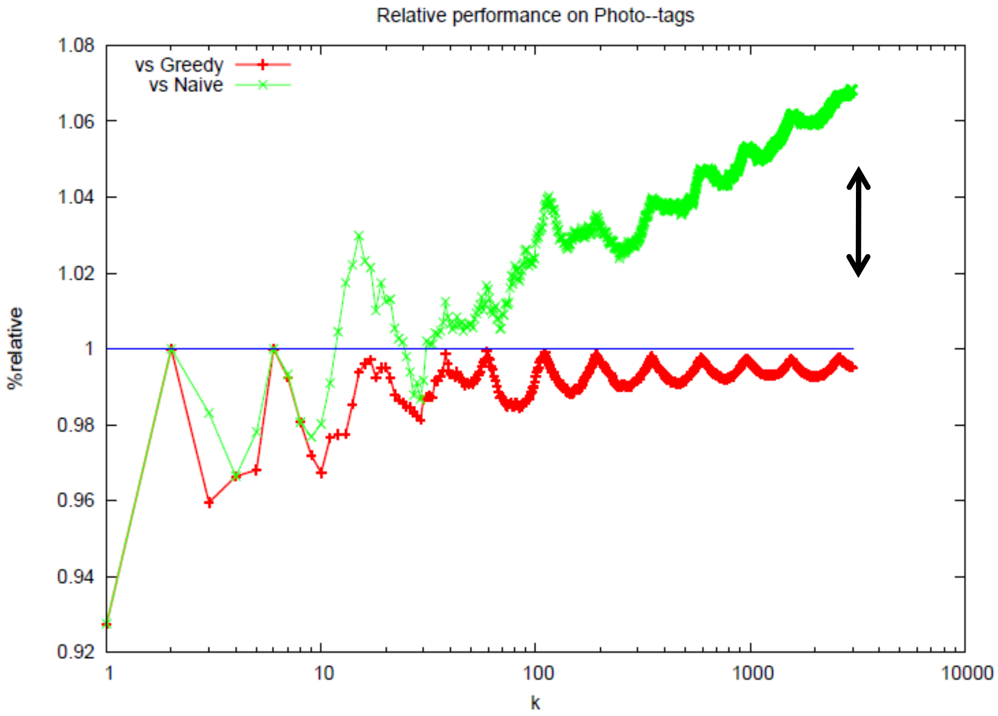
Smaller value of eps, higher run time

Outline

- Motivation
- Introduction
- Classical Approach : Greedy
- Proposed Algorithm: M_R Greedy
- Experiments
- Weaknesses
- Conclusion

Weaknesses (1)

- relative performance between Mr-Greedy and Naive. Even in the worst dataset (Photos-Tags) the naïve algorithm is less than 10% worse than MrGreedy.



Weaknesses (2)

- Choice of Є as 0.75
    - Non sensical choice as large value of Є provides no theoretical approx.

- Discussion about other methods of approximating the Max Cover problem, besides the greedy approach. For example algorithms based on linear programming relaxations.

Conclusion

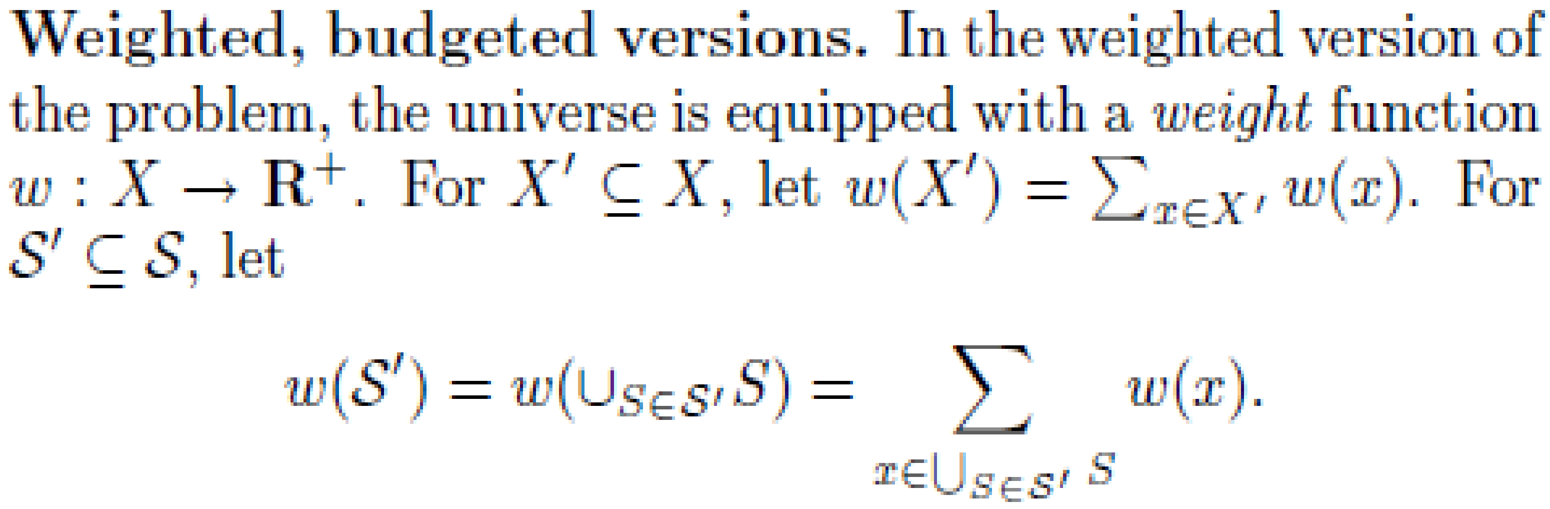
Obtained an algorithm that provides almost the same approximation as Greedy, and can be implemented in the scalable and widely-used Map-Reduce framework.

Thanks

&

Questions

Weighed budgeted version



Weighted, budgeted versions. In the weighted version of the problem, the universe is equipped with a *weight* function $w : X \to \mathbf{R}^+$. For $X' \subseteq X$, let $w(X') = \sum_{x \in X'} w(x)$. For $\mathcal{S}' \subseteq \mathcal{S}$, let

$$w(\mathcal{S}') = w(\cup_{S \in \mathcal{S}'} S) = \sum_{x \in \cup_{S \in \mathcal{S}'} S} w(x).$$

- Replace x (in all the sets that contain it) with w(x) unweighted copies of x.
- It is not strongly polynomial and it requires each element weight to be integral.
- To overcome that, we will multiply it with some positive number.

Weighed budgeted version

- Budgeted version of greedy provides an approximation of (1-1/√e).
- For MR Greedy, approximation will be (1-1/√e –O(ε)).
- And parallel running time will be polylogarithmic.