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?
Motivation (2)

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

Maximum users by selecting popular hosts
Problem Setting

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

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

$X = \{1, 2, \ldots, N\}$ 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 ?
Maximize the donors

Advertiser

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* ⊆ S is a max k-cover if |S*| = k and the coverage of S* is maximized over all subsets of S of size k.

Finding the optimal solution is NP-hard.

So we focus on approximation algorithms!
\( \alpha \) Approximation Algorithm

- Polynomial time, guaranteed to find “near optimal” solutions for every input.
- Suppose, I have an input set of 100 elements.
  - Optimal solution contains 80 elements
  - Let \( \alpha = 0.5 \)
  - Approximate solution says…
  - Approx \( \geq \alpha \) .optimal
  - In this case, approx : more than 40 elements.
\(\alpha\)-Approximate \(k\)-Cover

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

Looking for an 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, \(1-1/e \approx .63\).
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: $S_1; \ldots, S_m$, and an integer $k$
1: while $k > 0$ do
2: Let $S$ be a set of maximum cardinality
3: Output $S$
4: Remove $S$ and all elements of $S$ from other remaining sets
5: $k = k - 1$
Greedy Algorithm

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 $S_1, \ldots, S_m$, 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 very large.
  • For disk resident datasets, Greedy is not a scalable 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 \((k, v) \rightarrow \text{list}(k, v)\)

- **reduce**
  - takes as input a key and a list of values for that key
  - maps the input to a list of values
  - reduce \((k, \text{list}(v)) \rightarrow \text{list}(v)\)
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: 1,3
S2: 1,2,4
S3: 1,2,3
S4: 1,3
S5: 4
S6: 4

REDUCE

18
**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)
M4Greedy Algorithm

Algorithm 2 The M4GREEDY algorithm.

Require: A ground set $X$, a set system $S \subseteq 2^X$.
1: Let $C$ be an empty list
2: for $i = \lceil \log_{1+\epsilon^2} |X| \rceil$ downto 1 do
3: \hspace{1em} Let $S_w = \{S \mid S \in S \land |S| \geq (1+\epsilon^2)^{i-1}\}$
4: \hspace{1em} for $j = \lceil \log_{1+\epsilon^2} \Delta \rceil$ downto 1 do
5: \hspace{2em} Let $X' = \{x \mid x \in X \land \deg_{S_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
6: \hspace{2em} while $X' \neq \emptyset$ do
7: \hspace{3em} if there exists $S \in S_w$ such that $|S \cap X'| \geq \frac{\epsilon^2}{1+\epsilon^2}$ then
8: \hspace{4em} Append $S$ to the end of $C$
9: \hspace{3em} else
10: \hspace{4em} Let $S_p$ be any contained set of $S_w$ by including bad elements
11: \hspace{4em} Remove all the bad sets of $S_p$ to the end of $C$ in any order
12: \hspace{3em} A set $S \in S_p$ is bad if it contains bad elements of total weight more than $4\epsilon \cdot (1+\epsilon^2)^i$
13: \hspace{4em} Append all the sets of $S_p$ that are not bad to the end of $C$ in any order
14: \hspace{3em} End while
15: \hspace{1em} End for
16: Remove all the sets in $C$ from $S$
17: Remove all the elements in $\bigcup_{S \in C} S$ from $X$ and from the sets in $S$
18: Let $S_w = \{S \mid S \in S \land |S| \geq (1+\epsilon^2)^{i-1}\}$
19: Let $X' = \{x \mid x \in X \land \deg_{S_w}(x) \geq (1+\epsilon^2)^{j-1}\}$
20: Return the list $C$
Algorithm 2 The MrGREEDY algorithm.

Require: A ground set \( X \), a set system \( S \subseteq 2^X \).
1: Let \( C \) be an empty list
2: for \( i = \lceil \log_{1+\varepsilon^2} |X| \rceil \) downto 1 do
3:   Let \( S_w = \{ S \mid S \in S \land |S| \geq (1 + \varepsilon^2)^{i-1} \} \)
4:   for \( j = \lceil \log_{1+\varepsilon^2} \Delta \rceil \) downto 1 do
5:     Let \( X' = \{ x \mid x \in X \land \deg_{S_w}(x) \geq (1 + \varepsilon^2)^{j-1} \} \)
6:     while \( X' \neq \emptyset \) do
7:        if there exists \( S \in S_w \) such that \( |S \cap X'| \geq \frac{\varepsilon \cdot |X'|}{1+\varepsilon^2} \) then
8:           Append \( S \) to \( C \)
9:        else
10:           Let \( X'' = \{ x \mid x \in X \setminus \bigcup_{S \in C} S \} \)
11:           if there exists \( S' \in S_w \) such that \( |S' \cap X''| \geq \frac{\varepsilon \cdot |X''|}{1+\varepsilon^2} \) then
12:              Append \( S' \) to \( C \)
13:       end if
14:    end while
15: end for
16: end for
17: Remove all the sets in \( C \) from \( S \)
18: Remove all the elements in \( \bigcup_{S \in C} S \) from \( X \) and from the sets in \( S \)
19: Let \( S_w = \{ S \mid S \in S \land |S| \geq (1 + \varepsilon^2)^{i-1} \} \)
20: Let \( X' = \{ x \mid x \in X \land \deg_{S_w}(x) \geq (1 + \varepsilon^2)^{j-1} \} \)
21: Return the list \( C \)

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 \( X = \{1, 2, \ldots, n\} \), so total number of sets possible \( 2^x \)

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:

\[
\text{Require: A ground set } X, \text{ a set system } S \subseteq 2^X.
\]
\[
1: \text{Let } \mathcal{C} \text{ be an empty list}
\]
\[
2: \text{for } i = \lceil \log_{1+\epsilon^2} |X| \rceil \text{ downto } 1 \text{ do}
\]
\[
3: \quad \text{Let } S_w = \{S \mid S \in S \land |S| \geq (1+\epsilon^2)^{i-1}\}
\]
\[
4: \text{for } j = \lceil \log_{1+\epsilon^2} \Delta \rceil \text{ downto } 1 \text{ do}
\]
\[
5: \quad \text{Let } X' = \{x \mid x \in X \land \deg_{S_w}(x) \geq (1+\epsilon^2)^{j-1}\}
\]

\(\Delta\) 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 S ⊆ 2^X.
1: Let C be an empty list
2: for i = ⌈log_{1+ε^2} |X|⌉ downto 1 do
3:  Let S_w = \{S \mid S \in S \land |S| \geq (1 + ε^2)^{i-1}\}
4:  for j = ⌈log_{1+ε^2} \Delta⌉ downto 1 do
5:  Let X' = \{x \mid x \in X \land \deg_{S_w}(x) \geq (1 + ε^2)^{j-1}\}
6:  while X' ≠ ∅ do
7:     if there exists S \in S_w such that |S \cap X'| \geq \frac{ε^6}{1+ε^2}.
       then
          Append S to the end of C
    end
8: end
```

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:

6: while $X' \neq \emptyset$ do
7: 
8: if there exists $S \in S_w$ such that $|S \cap X'| \geq \frac{\epsilon^6}{1+\epsilon^2} \cdot |X'|$ then
9: 
10: Append $S$ to the end of $\mathcal{C}$
11: 
12: else
13: Let $S_p$ be a random subset of $S_w$ chosen by including each set in $S_w$ independently with probability $p = \frac{\epsilon^6}{(1+\epsilon^2)^j}$
14: 
15: if $\left| \bigcup_{S \in S_p} S \right| \geq |S_p| \cdot (1 + \epsilon^2)^i \cdot (1 - 8\epsilon^2)$ then
16: 
17: We say that an element $x$ is bad if it is contained in more than one set of $S_p$
18: 
19: A set $S \in S_p$ is bad if it contains bad elements of total weight more than $4\epsilon \cdot (1 + \epsilon^2)^i$
20: 
21: Append all the sets of $S_p$ that are not bad to the end of $\mathcal{C}$ in any order
22: 
23: Append the bad sets of $S_p$ 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 $S_w = \{ S \mid S \in S \land |S| \geq (1 + \epsilon^2)^i - 1 \}$

1. Map
   1. ForEach $e_i$
   2. Iterate through List (S)
   3. Emit ($S_j$, $e_i$) where $S_j \in S$

2. Reduce ($S_j$, List(e)) where e are elements
   1. If sizeof(List(e)) $\geq$ constant
   2. Emit ($S_j$)
Realization in MapReduce

Input set
1: S1, S2, S3, S4
2: S2, S3

MAP

S1: 1
S1: 3

REDUCE

S2: 1
S2: 2, 4

REDUCE

Output S2

IF COUNT IS MORE THAN OR EQUAL TO 3
Some facts about $M_R$ Greedy

- The approximation guarantee of $M_R$Greedy is $1-1/e-O(\epsilon)$.

- Running time $O(poly(\epsilon) \cdot \log^3 nm)$
  - $n$ 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
1. The coverage of $M_R$Greedy 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 $M_R$Greedy 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
   \[ m=5.64 \text{ M} \]
   \[ n=2.96 \text{ M} \]
   \[ E=72.8 \text{ M} \]
   \[ \Delta=2115 \]
   \[ w^*(S)=1.19 \text{ M} \]

2. Query – Hosts
   CP- k hosts addresses maximum queries
   \[ m=625 \text{ K} \]
   \[ n=239 \text{ K} \]
   \[ E=2.8 \text{ M} \]
   \[ \Delta=10 \]
   \[ w^*(S)=164 \text{ K} \]
Experiments: 5 Data Sets (2)

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

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

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 indistinguishable
Relative Performances
User – Hosts (1)

Spikes shows book keeping phases

Red line is very close to blue line
Query-Hosts (2)

Relative performance on Query-hosts

Getting worse
m=89 K
n=704 K
E=2.7 M
$\Delta=145$
w*(S)=54.3 K

Getting worse
Page-Ads (4)
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 $\epsilon$ as 0.75
  • Non sensical choice as large value of $\epsilon$ 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.
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 $w : X \to \mathbb{R}^+$. For $X' \subseteq X$, let $w(X') = \sum_{x \in X'} w(x)$. For $S' \subseteq S$, let

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

- 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 $1-1/\sqrt{e}$.
- For $M_R$ Greedy, approximation will be $(1-1/\sqrt{e} - O(\epsilon))$.
- And parallel running time will be polylogarithmic.