Max-Cover in Map-Reduce

F. Chierichetti, R. Kumar, A. Tomkins

Marco Cianfriglia

Laurea Magistrale in Informatica
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Università Sapienza, Roma

28 maggio 2012

SAPIENZA
UNIVERSITÀ DI ROMA

Agenda

Table of Contents

Il problema : Max-k-Cover

The NP-hard MAX-k-COVER problem requires selecting k sets from a collection so as to maximize the size of the union.

Applicazioni

- Selecting popular hosts to maximize users
- Selecting web pages to maximize visibility of banner ads
- Selecting the best places to put a set of charity dropboxes
- ecc..

In general

Every time we have a collection of sets and we want to pick some sets and maximize the elements picked in respect to some property

Table of Contents

Formal Definition

Coverage

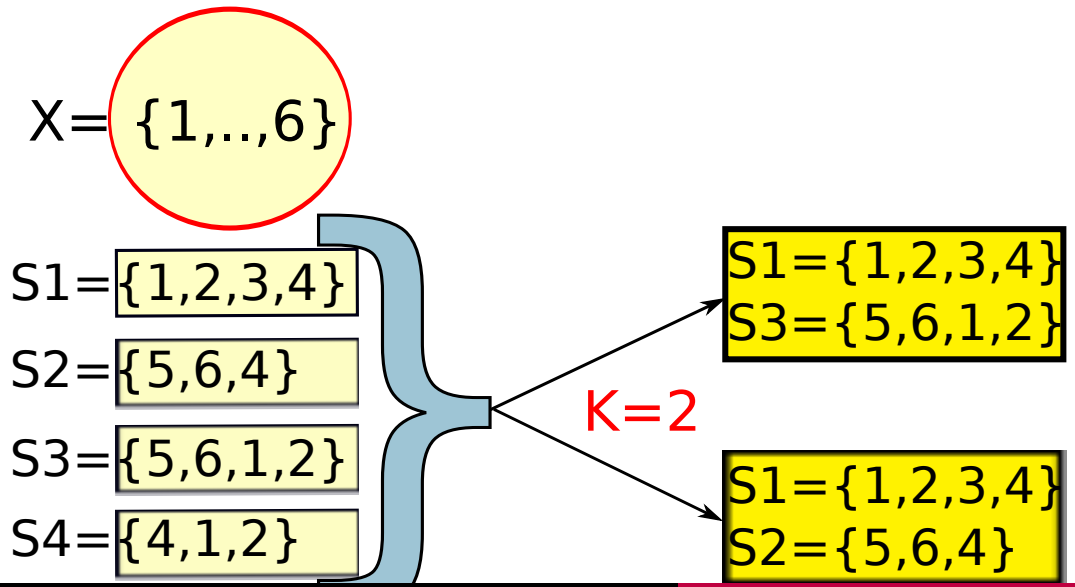
Let $X = \{1, ..., n\}$ be a universe of n elements. Let S, $\mid S \mid = m$ be a family of non-empty subsets of X. Given $S' \subseteq S$, the coverage $cov(S')$ of S' is simply

$$cov(S') = \cup_{S \in S'} S$$

Formal Definition

Max-k-Cover

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



Formal Definition

MAX-k-COVER è un problema NP-hard quindi ci concentreremo su algoritmi di approssimazione

α -APPROXIMATE k-COVER

For $\alpha > 0$, a set $S' \subseteq S$, $\mid S' \mid \leq k$, is an α **-approximate max k-cover** if for any max k-cover S^* , $cov(S') \geqslant \alpha \cdot cov(S^*)$

Formal Definition

Degree

We define the **degree** $deg(x)$ of an element $x \in X$ to be the number of sets cointaining x i.e, $deg(x) =\mid \{S \in S \mid S \ni x\} \mid$

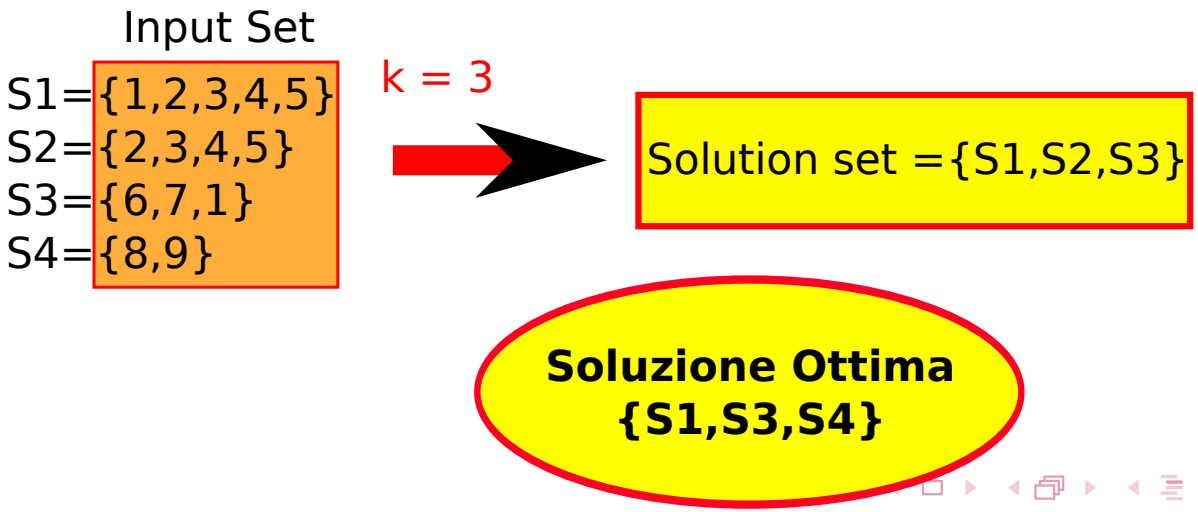
Maximum degree

We define the **maximum degree** Δ of an instance as $\Delta = max_{x \in X} deg(x)$.
We also define the degree of an element $x \in X$ with respect to a subfamily $S' \subseteq S$ as $deg_{S'}(x) =\mid \{S \in S' \mid S \ni x\} \mid$

Table of Contents

Soluzione Naive

- Si ordinano gli insieme rispetto alla cardinalità in ordine decrescente
  - si selezionano i primi k
- E' facile trovare un esempio in cui il Naive "sbaglia"



Classic GREEDY

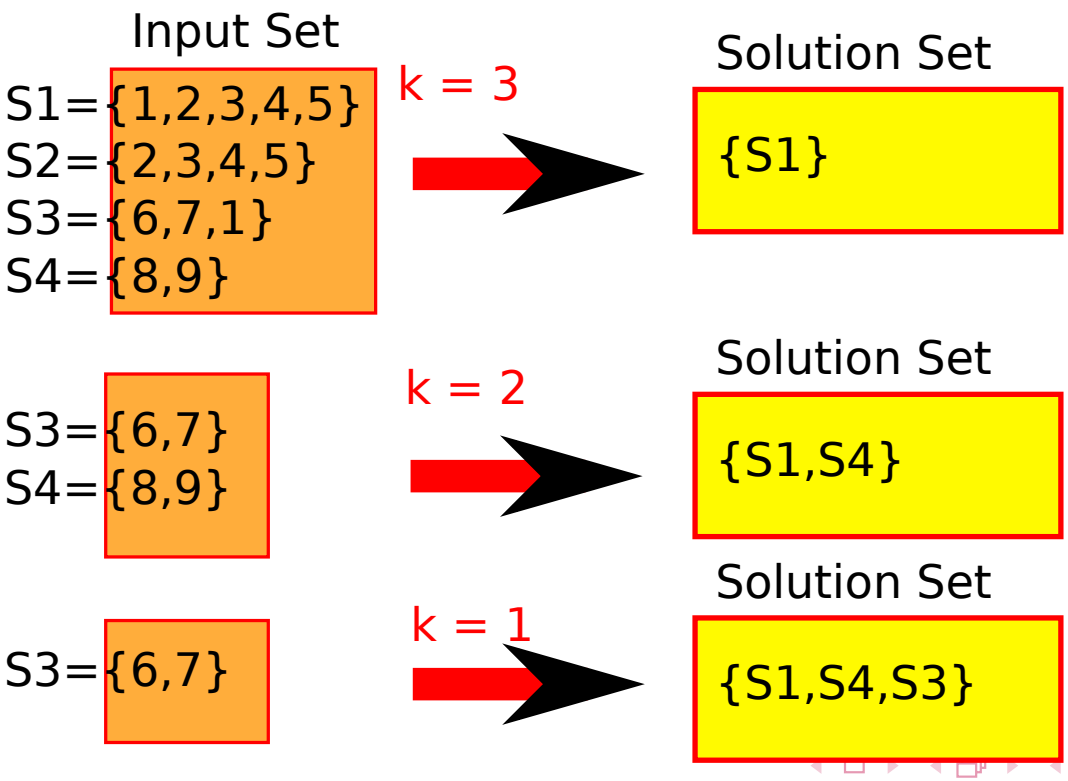
A greedy algorithm follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum

Implementation

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 its elements from other remaning sets
        $k = k - 1$

Classic Greedy: esempio

Un piccolo esempio di esecuzione del Greedy



Classic Greedy Algorithm

Analizziamo le caratteristiche :

- Semplice ed intuitivo
- Sequenziale
- Buona approssimazione ($\alpha = (1 - \frac{1}{e}) \approx 0.63$)

Classic Greedy Algorithm

Ma se dobbiamo ordinare ?

Classic Greedy Algorithm

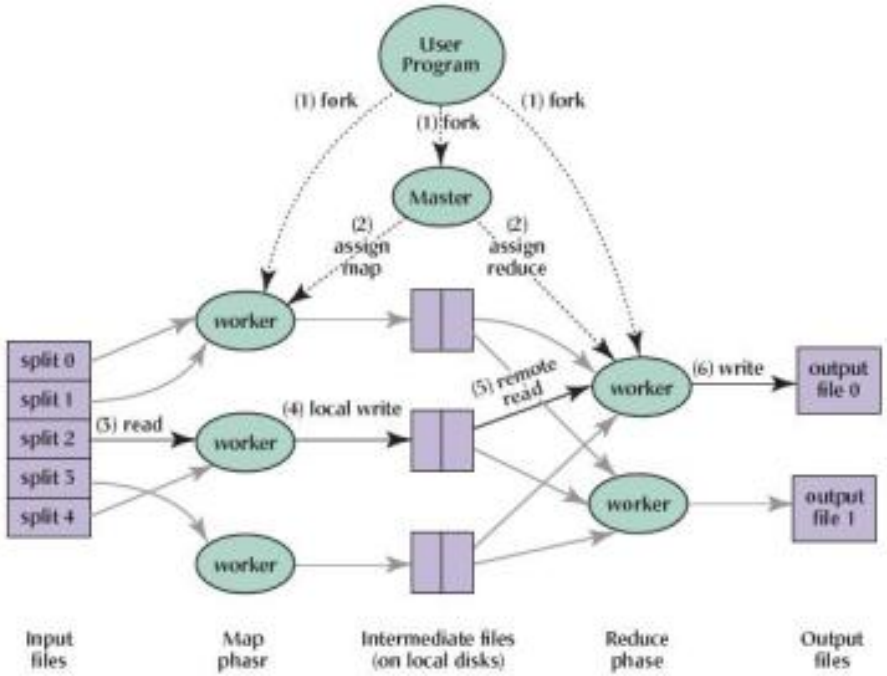
Alcuni fattori da considerare:

- Dopo aver selezionato l'insieme migliore tra i rimanenti, update rimuove gli elementi dell'insieme scelto dagli insiemi rimanenti
- Il numero di insiemi coinvolto nell'update può essere molto grande
- L'operazione di update deve essere ripetuta dopo la scelta di ogni insieme da inserire nella soluzione
- Approccio non scalabile
- Di fatto impraticabile su disk-resident datasets

Table of Contents

Map Reduce Framework

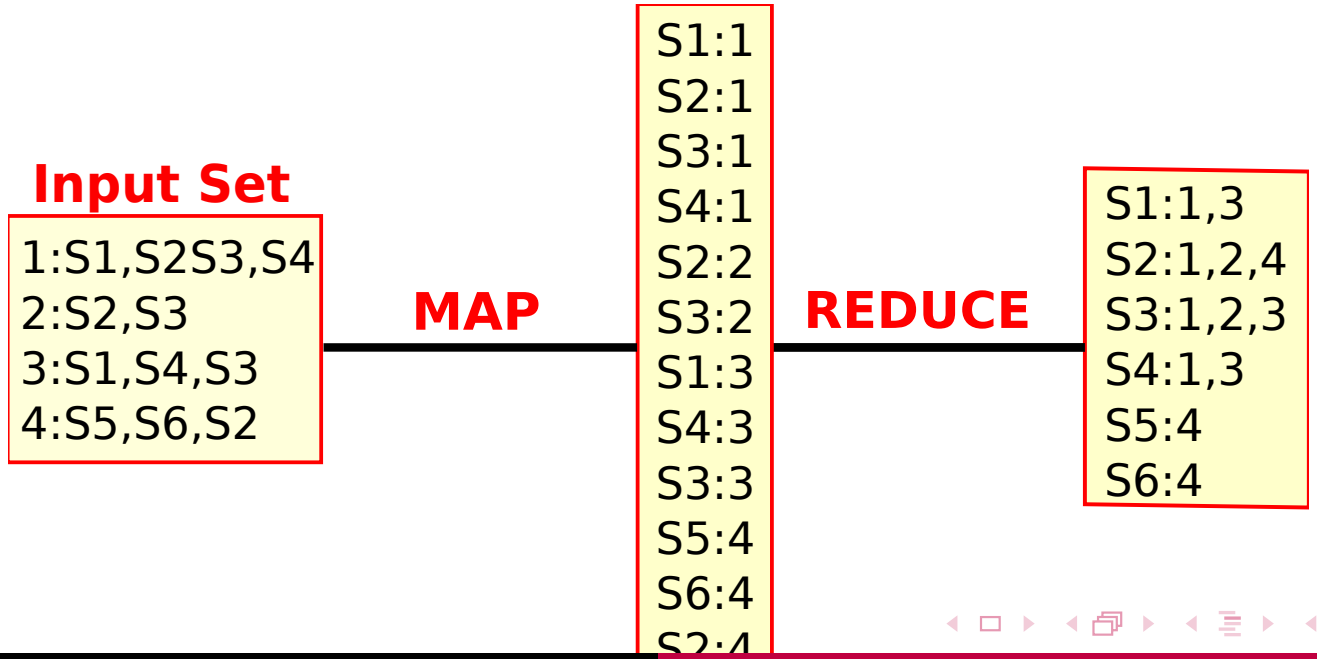
- Framework ideato da Google per computazione paralella, distibuita e sequenziale di grande quantità di dati
- Basato sulle funzioni **map** e **reduce** del Lisp



Map-Reduce Transpose

Vediamo ora un esempio di utilizzo di Map-reduce

- Transposing of an adjacency list
- key-element
- Value-set



Map-Reduce Transpose

- **MAP**: L'operazione di Map prende in input una lista di adiacenza x: Sk, ... Si,...Sy (Es. 1:S1,S2,S3,S4) ed emette le coppie (chiave, valore), dove chiave = Si e valore = x
- Le coppie con la stessa chiave vengono inviate allo stesso reducer
- **REDUCE**: L'operazione Reduce raggruppa tutti i valori per una stessa chiave in un unica lista associata alla chiave (Es: Si:x,y,...,z).

Map-Reduce Transpose

Implementazione delle funzioni **Map** e **Reduce** per l'operazione transpose

def **Map**($list_i$):
    foreach set in $list_i$:
        output(set,i)

def **Reducer**(key,value):
    foreach value for the same key
        output (key, list of values)

Table of Contents

MrGreedy

Idea

Scegliere diversi insiemi da aggiungere simultaneamente alla soluzione.
Scegliere quando è un punto cruciale dell'algoritmo.

- Algoritmo per MAX-k-COVER basato su Map-Reduce
- Fattore di approssimazione di $\alpha = (1 - \frac{1}{e-\epsilon})$ (Greedy $\alpha = (1 - \frac{1}{e})$)
- Tempo $O(poly(\epsilon)\log^3(nm))$ round Map-Reduce
  - n numero di elementi
  - m numero di insiemi

MrGreedy: valore α

Il fattore di approssimazione è dimostrato essere $\alpha = (1 - \frac{1}{e-\epsilon})$

Idea dimostrazione

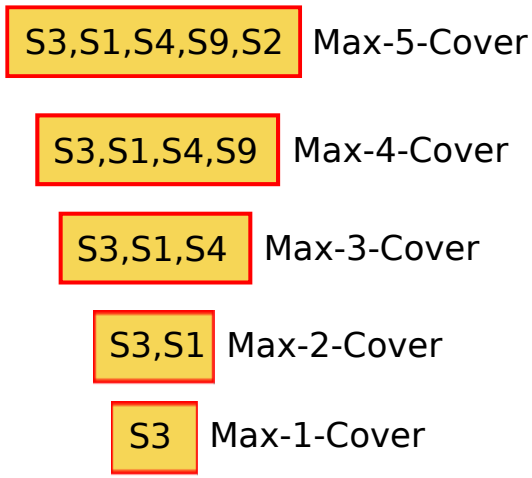
Calcolare rapporto di approssimazione tra MrGreedy e Greedy "classico" di cui si conosce, ed è dimostrato, il valore di approssimazione.

Unendo le due informazioni si ottiene il valore di approssimazione per MrGreedy e la dimostrazione di correttezza.

MrGreedy: Caratteristiche

Una caratteristica interessante di MrGreedy è:

- **Prefix-optimality**: $\forall k$ il prefisso di lunghezza k nell'output ordinato è un $(1 - \frac{1}{e-\epsilon})$-approssimazione per il corrispondente Max-k-Cover



MrGreedy: Requisiti

- Requisiti imposti all'algoritmo così da renderlo **utilizzabile in pratica** e **realmente scalabile**
  1. Numero delle iterazioni è al massimo *polilogaritmico* nella dimensione dell'input
  2. La dimensione dell'output rimanga lineare alla dimensione dell'input
  3. Le operazioni di **map** e **reduce**:
     - lavorino in tempo lineare rispetto alla dimensione del loro input
     - utilizzino una quantità di memoria costante o logaritmica rispetto alla dimensione del loro input.

MrGreedy Algorithm

Require: A ground set X, a set system $S \subseteq 2^X$
Let C be an empty list
for $i = \lceil \log_{1+\epsilon^2} \mid X \mid \rceil$ **downto** 1 **do**
Let $S_w = \{S \mid S \in S \wedge \mid S \mid \geqslant (1+\epsilon^2)^{i-1}\}$
for $j = \lceil \log_{1+\epsilon^2} \Delta \rceil$ **downto 1 do**
Let $X' = \{x \mid x \in X \wedge deg_{S_w}(x) \geqslant (1+\epsilon^2)^{j-1}\}$
while $X' \neq \oslash$ **do**
if there exists $S \in S_w suchthat \mid S \cap X' \mid \geqslant \frac{\epsilon^6}{1+\epsilon^2} \cdot \mid X' \mid$ **then**
Append S to the end of C
else
Let S_p be a random subset of S_w chosen by including each set in S_w indepentently with probability $p = \frac{\epsilon}{(1+\epsilon^2)^j}$
if $\mid \cup_{S \in S_p} S \mid \geqslant \mid S_p \mid \cdot (1+\epsilon^2)^i \cdot (1-8\epsilon^2)$ **then**
We say that an element x is **bad** if it is contained in more than one set of S_p
A set $S \in S_p$ in bad if it contains bad elements of total weight more than $4\epsilon \cdot (1+\epsilon^2)^i$
Append all the sets of S_p that are not bad to the end of C in any order
Append the bad sets of S_p to the end of C in any order
Remove all the sets in C from S
Remove all the elements in $\cup_{S \in C} S$ from X and from the sets in S
Let $S_w = \{S \mid S \in S \wedge \mid S \mid \geqslant (1+\epsilon^2)^{i-1}\}$
Let $X' = \{x \mid x \in X \wedge deg_{S_w}(x) \geqslant (1+\epsilon^2)^{j-1}\}$
Return the list C

MrGreedy: analisi step-by-step 1/4

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 $S \subseteq 2^X$
Let C be an empty list
for $i = \lceil \log_{1+\epsilon^2} \mid X \mid \rceil$ **downto** 1 **do**
Let $S_w = \{S \mid S \in S \wedge \mid S \mid \geqslant (1+\epsilon^2)^{i-1}\}$

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

MrGreedy: analisi step-by-step 2/4

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

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

MrGreedy: analisi step-by-step 3/4

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

- 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

MrGreedy: analisi step-by-step 4/4

else
Let S_p be a random subset of S_w chosen by including each set in S_w indepentently with probability $p = \frac{\epsilon}{(1+\epsilon^2)^j}$
if $| \cup_{S \in S_p} S | \geqslant | S_p | \cdot (1+\epsilon^2)^j \cdot (1-8\epsilon^2)$ **then**
We say that an element x is **bad** if it is contained in more than one set of S_p
A set $S \in S_p$ in bad if it contains bad elements of total weight more than $4\epsilon \cdot (1+\epsilon^2)^j$
Append all the sets of S_p that are not bad to the end of C in any order
Append the bad sets of S_p to the end of C in any order

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

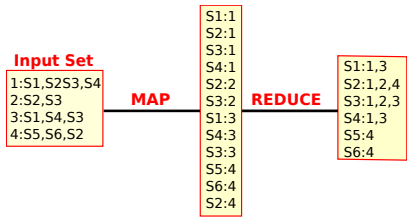
MrGreedy: MapReduce

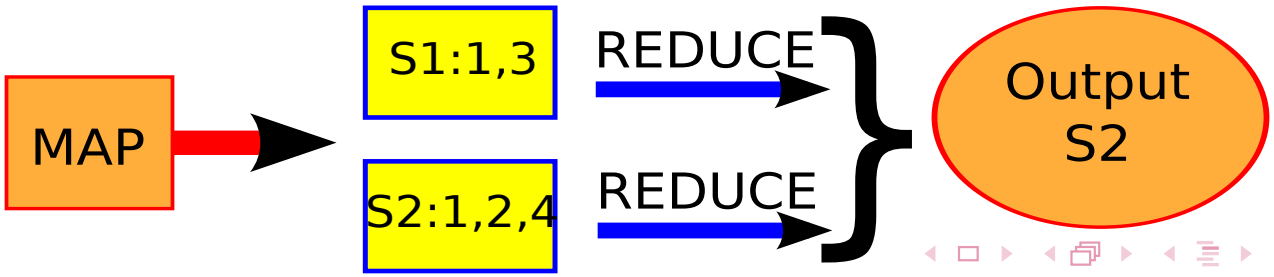
Vediamo un semplice snapshot di un singolo round map-reduce

Input Set

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

La transpose





MrGreedy: considerazioni

Alcune considerazioni su MrGreedy:

- E' dimostrato che il valore di approssimazione di MrGreedy è $\alpha = 1 - \frac{1}{e - O(\epsilon)}$
- Il tempo di esecuzione è $O(poly(\epsilon) \cdot log^3 nm)$ dove
  - n è il numero degli elementi
  - m è il numero degli insiemi
- Risolve alcuni problemi del Greedy classico nell'analisi di grandi dataset utilizzando il framework Map-Reduce ed inoltre ha un'approssimazione molto simile a quella del Greedy ($1 - \frac{1}{e}$)

MrGreedy: Map-Reduce

Utilizzo di Map-Reduce in MrGreedy

- Selezione S_W e di X' può essere realizzata come combinazione di **map**, **transpose** e **reduce** (riga 3 e 5)
- Calcolo dimensione dell'intersezione : **map** e **reduce** (riga 7)
- ecc.. (righe 10, 11, 13-17)

Transpose

L'operazione *transpose* è utilizzata per mantenere aggiornate le relazioni di appartenenza tra gli elementi e gli insiemi.

Table of Contents

Valutazione: Obiettivi

Gli esperimenti effettuati hanno avuto come obiettivi

1. Dimostrare la non-distinguibilità di MrGreedy dal classico Greedy
2. Dimostrare che MrGreedy è realmente parallelizzabile
3. Provare la fattibilità di implementare MrGreedy su Map-Reduce

Data sets utilizzati

- **User-hosts**
- **Query-hosts**
- **Photos-tags**
- **Page-ads**
- **User-queries**

Data set	m	n	E	Δ	$w*(S)$
User-hosts	5.64M	2.96M	72.8M	2,115	1.19M
Query-host	625K	239K	2.8M	10	164K
Photo-tags	89k	704K	2.7M	145	54.3K
Page-ads	321K	357K	9.1M	24,825	164K
User-queries	14.2M	100K	72M	5,369	21.4K

Tabella: Details of the data sets

- **m**: the number of sets
- **n**: the number of elements
- **E**: the number of element-set membership
- Δ: the maximum degree of an element
- $w*(S)$: the maximum cardinality of a set

Data set 1

User-hosts

The problem is to determine the hosts that are visister by as many users as possible

- Users are elements
- Hosts are sets

The set for a given host consists of all the users who visited some web page on the host during the browsing.
Data from Yahoo! toolbar logs. The istance is a subset of data during the week July 18-25 2009

Data set 2

Query-hosts

We seek the hosts that together cover as many unique queries as possible

- Queries are elements
- Hosts are sets

The set for a given host contains all queries for which the host appeared in the top ten search results.
The queries were sub-sampled from Yahoo! query logs on July 1 2009

Data set 3

Photos-tags

The problem is to find those tags that together cover as many photos as possible

- Photographs uploaded to the Flickr web service are elements
- Tags are sets

The set for a given tag contains the photos to which that tag has been applied

Data set 4

Page-ads

The problem is to determine a collection of ads that covers as many pages as possible

- Web pages are elements
- The ads are sets

The set for a given ads contains the web pages in which the ads appeared.
These data are derived from Yahoo!'s content match advertising system.

Data set 5

User-queries

The problem is to determine the queries that cover as many users as possible

- Anonymized ids are elements
- Queries are sets

The set for a particular query contains all the anonymized userids that issued the query in our sample.
The queries are derived from Yahoo! logs

Esperimento 1: descrizione

Obiettivo

Mostrare la qualità della soluzione che si ottiene con MrGreddy confrontando il suo grado di approssimazione con quello di Greedy e Naive

Esperimento 1: grafico 1

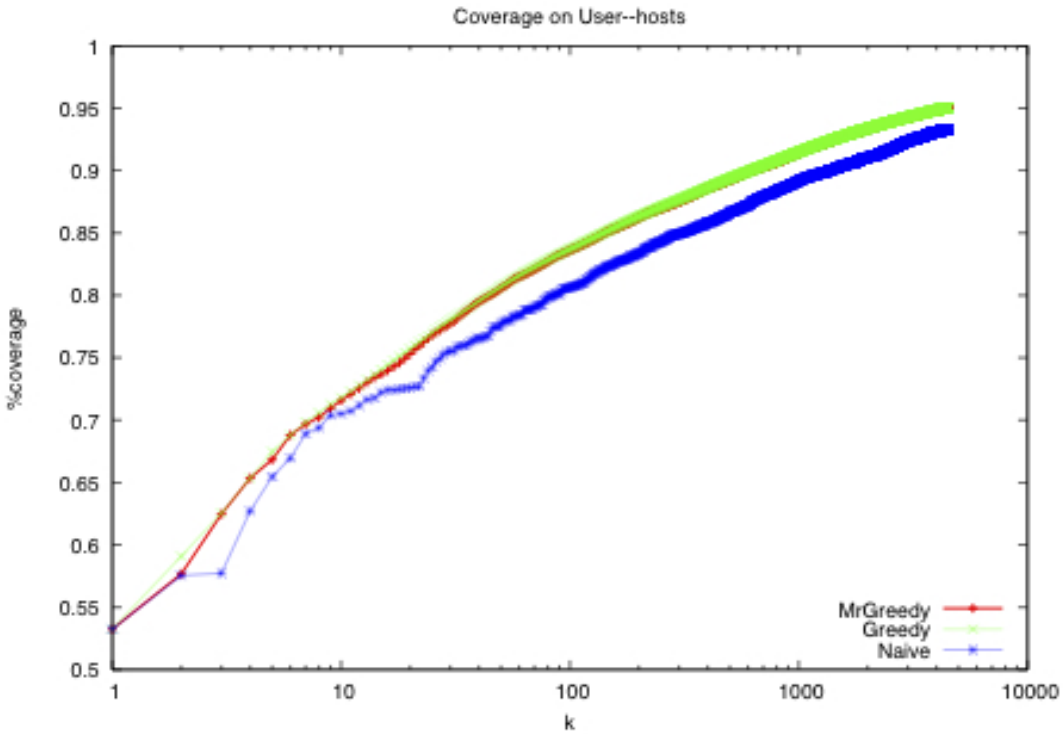


Figura: Performance of MrGreedy, Greedy and Naive on User-hosts

Esperimento 1: grafico 2

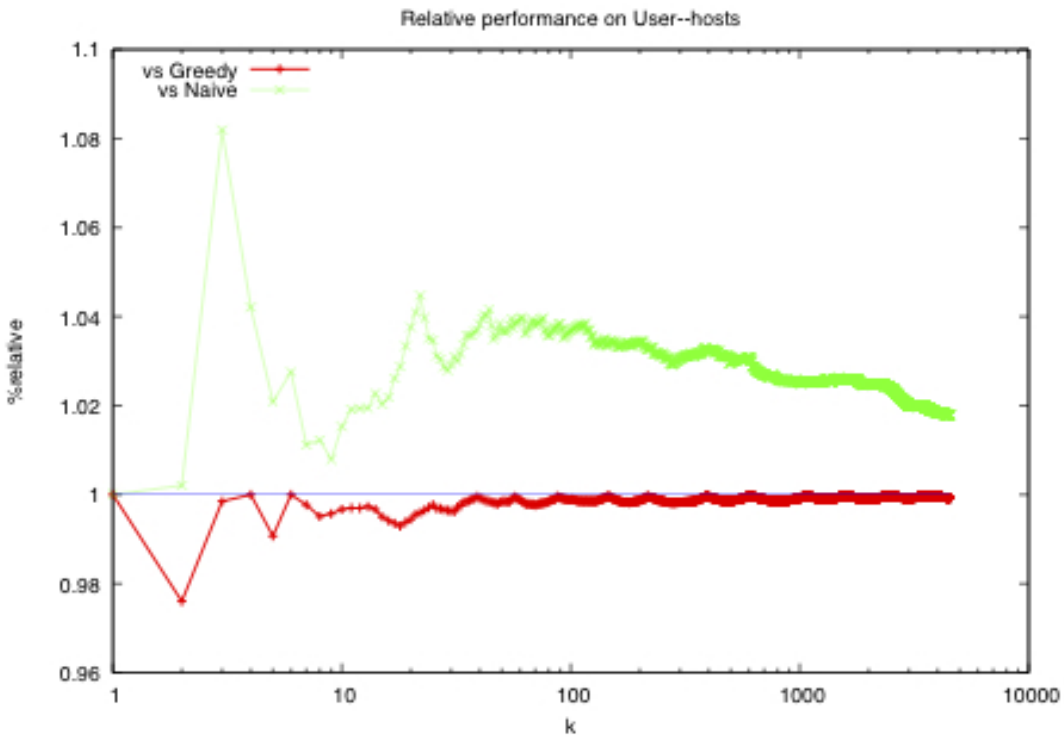


Figura: Relative performance on User-hosts

Esperimento 1: grafico 3

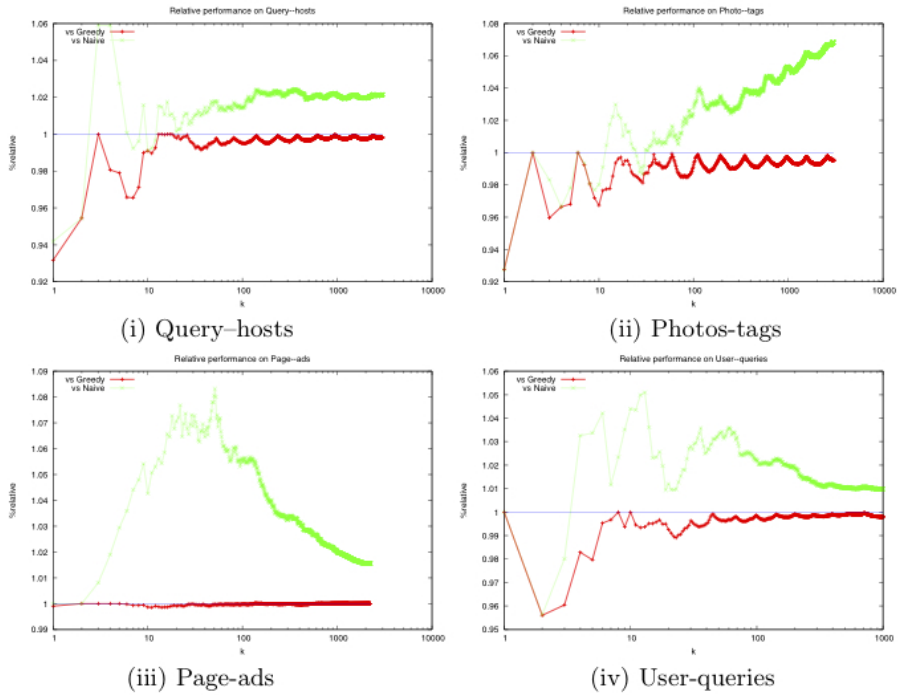


Figura: Relative performance on other four data sets

Esperimento 2: descrizione

Obiettivo

Studiare l'effetto del parametro ϵ e come il variare di quest'ultimo influisca su

- qualità della coverage
- tempo di esecuzione

Esperimento 2: grafico

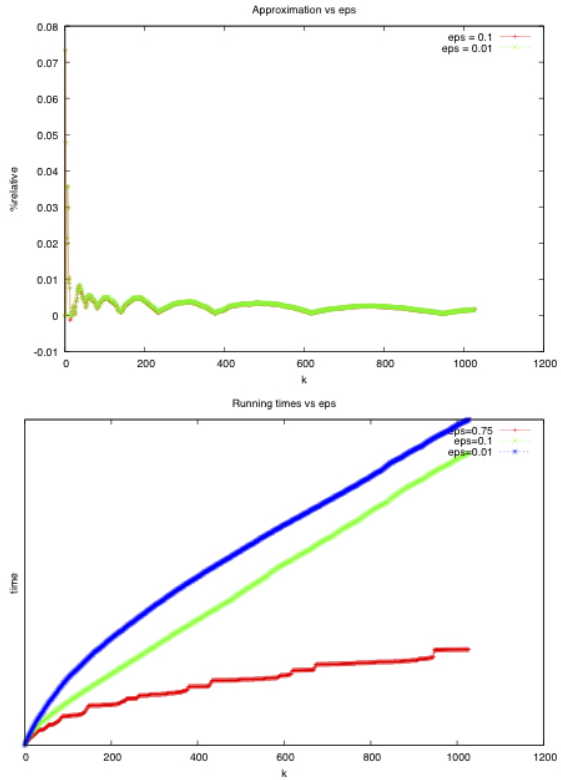


Figura: Effect of ϵ

Esperimento 3: descrizione

Obiettivo

Studiare il tempo di esecuzione di MrGreedy e metterlo a confronto con quello di Greedy.
Sono stati analizzati due aspetti:

- Wall-clock time
- Numero di esecuzioni parallele possibili con MrGreedy (in funzione di k)
  - In particolare hanno studiato il numero di volte che le righe 14 e 15 sono state invocate. Il numero di invocazione è di circa 90 e in quello step sono stati aggiunti alla soluzione piu' di 2400 insiemi.

Esperimento 3: grafico 1

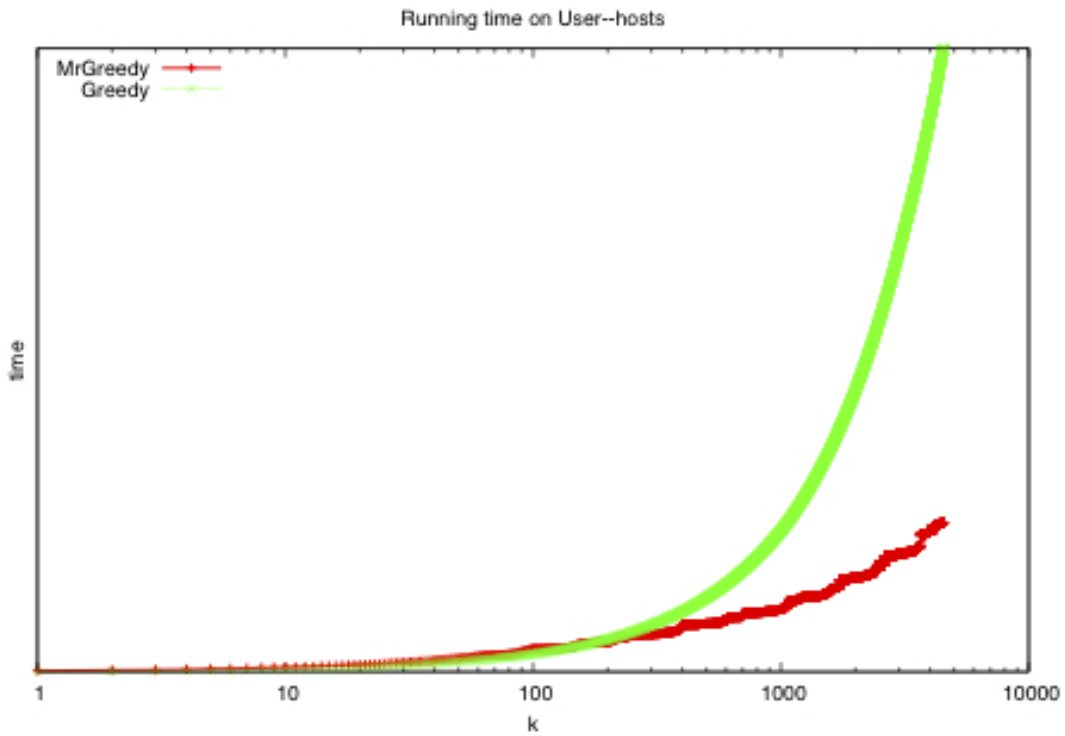


Figura: Running times for MrGreedy vs Greedy on User-hosts, on a semi-log scale (time vs log-k)

Esperimento 3: grafico 2

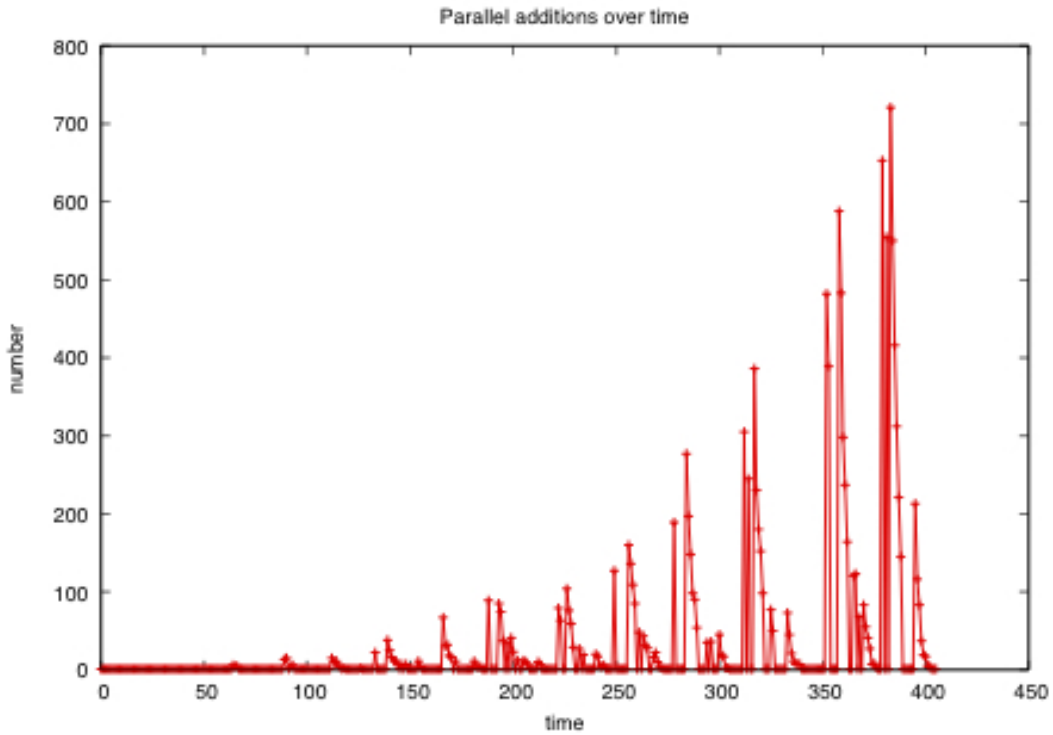


Figura: Number of batch additions over time for MrGreedy on User-hosts

Conclusioni

L'algoritmo presentato MrGreedy sfrutta il framework Map-Reduce per trovare una soluzione approssimata al Max-k-Cover che sia scalabile al crescere dell'input.

- fattore approssimazione (quasi) identico a quello Greedy
- realmente parallelizzabile
- nella pratica molto veloce (cerca di non effettuare k scelte sequenziali)

Ringraziamenti

Grazie a tutti per l'attenzione.

Domande

Domande

Bibliografia

1. Max-Cover in Map-Reduce (Article, Flavio Chierichetti, Ravi Kumar, Andrew Tomkins)
2. Max-Cover in Map-Reduce (Presentation, Presented by Isha Khosla, Advisor Klaus Berberich)
3. MapReduce: Simplified Data Processing on Large Clusters (Article, Jeffrey Dean, Sanjay Ghemawat.)
4. Map-Reduce - What is it, and why is it so popular (Presentation, Luigi Laura)