Max-Cover in Map-Reduce

Flavio Chierichetti*
Dipartimento di Informatica
La Sapienza Univ. of Rome
Roma 00198, Italy.
chierichetti@di.uniroma1.it

Ravi Kumar
Yahoo! Research
701 First Avenue
Sunnyvale, CA 94089.
ravikumar@yahoo-inc.com

Andrew Tomkins*
Google, Inc.
1600 Amphitheater Parkway
Mountain View, CA 94043.
atomkins@gmail.com

ABSTRACT

The NP-hard Max-k-Cover problem requires selecting k sets from a collection so as to maximize the size of the union. This classic problem occurs commonly in many settings in web search and advertising. For moderately-sized instances, a greedy algorithm gives an approximation of $(1 - 1/e)$. However, the greedy algorithm requires updating scores of arbitrary elements after each step, and hence becomes intractable for large datasets.

We give the first max cover algorithm designed for today's large-scale commodity clusters. Our algorithm has provably almost the same approximation as greedy, but runs much faster. Furthermore, it can be easily expressed in the Map-Reduce programming paradigm, and requires only polylogarithmically many passes over the data. Our experiments on five large problem instances show that our algorithm is practical and can achieve good speedups compared to the sequential greedy algorithm.

Categories and Subject Descriptors. F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms. Algorithms, Experimentation, Theory

Keywords. Maximum cover, Greedy algorithm, Map-reduce

1. INTRODUCTION

Say a search engine wishes to focus its attention on one thousand queries such that as many users as possible will see one of them. Selecting these queries is a *maximum coverage* (or *max cover*) problem. These problems arise whenever we seek the "best" collection of items, but the items may partially overlap in the value they provide, and should not be double-counted. All of the following are instances of max cover problems: how should one select 100 movie stars that reach as many people on the planet as possible? If a web search engine has resources to change the appearance of results from 500 websites, how should the websites be chosen to touch as many queries as possible? What collection of five vitamins wards off the most ailments? Where should a set of charity dropboxes be placed to be available to as many people as possible?

In fact, all these examples are instances of the canonical problem of this type, Max-k-Cover, which asks to select k sets from a family of subsets of a universe so that their union is as large as possible. The problem is NP-hard, but can be approximated by a simple greedy algorithm (called Greedy) to within a factor of $(1 - 1/e)$ of the best possible. However, after selecting the best remaining set, Greedy must remove all elements of that set from other sets. The number of sets that must be touched to perform this update could be enormous, and the operation must be repeated for each set to be output. For disk-resident datasets, Greedy is not a scalable approach.

Nonetheless, max cover problems arise frequently at large scale. Dasgupta et al. [8], for instance, employ this formulation in pursuit of discovering fresh content for web crawler scheduling policies: finding as much new content as possible by crawling at most k webpages. Saha and Getoor [34] used the max cover formulation for a multi-topic blog-watch application: finding at most k blogs to read interesting articles on a list of topics.

Our motivation for studying max cover problems arose in one of the examples described above: we wished to find a large but bounded number of web hosts that appeared within the top-three results for as many web queries as possible. As the analysis was to take place over weeks or months of data, the scale of the problem quickly reached tens of billions of queries, making the performance of Greedy unacceptable. Thus, we began to pursue a version of this classical algorithm that could be implemented on the massively parallel large-scale data processing clusters based on the Map-Reduce paradigm [9] that are increasingly available today. These clusters are characterized by a focus on sequential data processing, another challenge in the context of the existing algorithm.

Max cover problems arise frequently in the context of web search and advertising, and when they do, they are often at daunting data scales. In addition to the host analysis problem described above, consider an advertiser interested in placing banner ads at various points of the Yahoo! network. The advertiser pays a fixed amount for each impression, independent of the location, and wishes to reach as many users as possible. Or consider an experiment in which we wish to deploy a manually-generated result in response to a small set of queries that will reach users from as many US zip codes as possible. Algorithmic tools for these natural questions are not generally available using the programming paradigms at our disposal.

*Part of this work was done while the author was at Yahoo! Research.

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

Our contributions. In this paper we develop a Map-Reduce-based algorithm for Max-k-Cover. Specifically, our algorithm obtains a $(1 - 1/e - \epsilon)$-approximation and can be implemented to run in $O(\text{poly}(\epsilon)\log^3(nm))$ Map-Reduce steps over the input instance, where n is the number of elements and m is the number of sets. Thus, we end up in the best of two worlds: nearly matching the performance of Greedy, while obtaining an algorithm that can be implemented in the scalable Map-Reduce framework. This is one of the few instances of a non-trivial Map-Reduce realization of a classical algorithm with provable guarantees of performance (more on this point in Section 3).

The main idea behind our algorithm, which we call MrGreedy, is to simultaneously choose many sets to include in the solution. This objective cannot be met at every step — identifying when to perform the parallel choice is the crux of our algorithm. In this regard, we derive inspiration from the parallel algorithm of Berger, Rompel, and Shor [2] for Set-Cover. Our problem, however, requires new ideas to carry out the analysis; see Section 4. To this end, we consider a weaker sequential version of Greedy, called MrGreedy, and show that its approximation ratio is essentially equal to that of Greedy. Then, we show that MrGreedy (whose parallel running time we can bound to be polylogarithmic) has an approximation ratio at least as good as the weaker sequential algorithm. In addition, MrGreedy has a strong output guarantee that we call *prefix-optimality*: for each k, the prefix of length k in the ordering output by MrGreedy is a $(1 - 1/e - \epsilon)$-approximation to the corresponding Max-k-Cover.

We then show how to implement our algorithm in the Map-Reduce setting, where the parallel running time translates to the number of Map-Reduce steps over the input. A crucial feature of MrGreedy is that it does not rely on main memory to store the elements of a set or the sets to which an element belongs. (Making such assumptions can be unrealistic — a common website may appear in the top results for tens of millions of queries.) We conduct experiments on five large real-world instances. Our experiments show that it is feasible to implement MrGreedy in practice, and obtain reasonable speedup over Greedy. Moreover, the approximation performance of MrGreedy is virtually indistinguishable from that of Greedy.

Since the set system can be alternatively viewed as a bipartite graph (elements and sets on each side and element–set membership determines the edges), the number of elements coverable with k sets can be viewed as a purely structural property of bipartite graphs. In this aspect, our work paves the way for the development of better understanding of bipartite graphs such as how much duplication exists in the covering of elements by sets; this is along the lines of [23].

Organization of the paper. The rest of the paper is organized as follows. Section 2 contains the necessary background material on Max-k-Cover, Greedy, and Map-Reduce. Section 3 discusses related work on both Max-k-Cover and Map-Reduce. Section 4 presents our main result: MrGreedy, a parallel algorithm for Max-k-Cover, and its Map-Reduce realization. Section 5 discusses the preliminary experimental results on the implementation of this algorithm. Section 6 contains the concluding thoughts and directions for future work.

2. PRELIMINARIES

In this section we set up the basic notation and provide the necessary background for the problem and the computational model.

2.1 The Max-k-Cover problem

Let $X = \{1, \ldots, n\}$ be a universe of n elements. Let $\mathcal{S}, |\mathcal{S}| = m$ be a family of non-empty subsets of X.

Given $\mathcal{S}' \subseteq \mathcal{S}$, the *coverage* $\text{cov}(\mathcal{S}')$ of $\mathcal{S}'$ is simply

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

Without loss of generality, we can assume that $\text{cov}(\mathcal{S}) = X$.

Definition 1 (Max k-cover). *Given an integer $k > 0$, $\mathcal{S}^* \subseteq \mathcal{S}$ is a* max k-cover *if $|\mathcal{S}^*| = k$ and the coverage of $\mathcal{S}^*$ is maximized over all subsets of $\mathcal{S}$ of size k.*

Since the Max-k-Cover problem is known to be NP-hard, we will focus on provably good approximation algorithms.

Definition 2 (α-approximate k-cover). *For $\alpha > 0$, a set $\mathcal{S}' \subseteq \mathcal{S}$, $|\mathcal{S}'| \leq k$, is an* α-approximate max k-cover *if for any max k-cover $\mathcal{S}^*$, $\text{cov}(\mathcal{S}') \geq \alpha \cdot \text{cov}(\mathcal{S}^*)$.*

A polynomial-time algorithm producing an α-approximate max k-cover for every instance of Max-k-Cover is said to be an *α-approximation algorithm* for the problem.

We define the *degree* $\deg(x)$ of an element $x \in X$ to be the number of sets containing x, i.e., $\deg(x) = |\{S \in \mathcal{S} \mid S \ni x\}|$. 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 $\mathcal{S}' \subseteq \mathcal{S}$ as $\deg_{\mathcal{S}'}(x) = |\{S \in \mathcal{S}' \mid S \ni x\}|$.

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

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

The goal is find $\mathcal{S}^* \subseteq \mathcal{S}$ such that $|\mathcal{S}^*| = k$ and has maximum weight $w(\mathcal{S}^*)$.

In the budgeted version, each set $S \in \mathcal{S}$ is equipped with a *cost* $c : \mathcal{S} \to \mathbf{R}^+$. The cost of $\mathcal{S}' \subseteq \mathcal{S}$ is given by $c(\mathcal{S}') = \sum_{S \in \mathcal{S}'} c(S)$. Given a budget B, the goal now is to output $\mathcal{S}^* \subseteq \mathcal{S}$ whose cost is at most B and the weight of the elements it covers is maximized.

2.2 The Map-Reduce model

We focus on the Map-Reduce model of computation [9]. In the Map-Reduce model, computations are distributed across several processors, split as a sequence of map and reduce steps. The map step consumes a stream of key–value tuples and outputs a set of (possibly different or amended) key–value tuples. In the reduce step, all tuples with same key are brought and processed together. For example, transposing an adjacency list — a crucial component in our algorithm — can be done effortlessly in Map-Reduce: given tuples of the form $\langle i; S_1, \ldots, S_{k_i}\rangle$, meaning that the element i (key) is present in sets $S_1, \ldots, S_{k_i}$, the *transpose* operation will produce an output of the form $\langle S; i_1, \ldots, i_{\ell_S}\rangle$, where

$i_1, \ldots, i_{\ell_S}$ are the elements of the set S (key). Realizing this in Map-Reduce is easy: the map step outputs tuples of the form $\langle S; i \rangle$ while the reduce step groups together all the elements that belong to the set S, the key.

Map-Reduce is a powerful computational model that has proved successful in enabling large-scale web data mining. For example, many matrix-based algorithms, such as PageRank computation [31], have been implemented successfully in the Map-Reduce model and used to process gargantuan snapshots of the web graph. While the general Map-Reduce model allows for greater flexibility, to make our algorithms practical and truly scalable, we list three requirements on their efficiency.

- Many Map-Reduce algorithms can be iterative; PageRank for instance is such an algorithm. In most scenarios, the number of iterations can be a pre-determined constant, albeit large. In our case, we require that the number of iterations is at most polylogarithmic in the input size.
- While in principle the output of the map or reduce step can be much bigger than the input, in our case, we require that it still remain linear in the input size. Also, we require the map and reduce steps to run in time linear in their input sizes.
- We require that the map/reduce steps use constant or logarithmic amount of memory. In particular, we do not assume that it is possible to store the elements of a set or all the sets to which an element belongs, in memory. In other words, we look for an algorithm that works in a truly streaming fashion, without relying on the main memory.

2.3 The classical Greedy algorithm

A classical *greedy algorithm* achieves a constant-factor approximation to Max-k-Cover. The factor of approximation is $1 - 1/e \approx .63$. We first recall this algorithm.

Algorithm 1 The Greedy algorithm.

```
Require: $S_1, \ldots, S_m$, 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
    $k = k - 1$
```

Note that this algorithm is blatantly sequential: after a set is picked to be included in the solution, bookkeeping is necessary to keep the remaining sets and elements up-to-date. This may not be expensive if the data structures to maintain the element–set memberships can be held in memory and if k passes over the data is acceptable; however, neither of these is feasible with massive data. As we will see, MrGreedy also has an overall greedy and an iterative flavor, but if an opportunity permits, it will try to pick many sets in parallel to be included in the solution. Doing this, while simultaneously preserving the approximation guarantee is the balance our algorithm will strive to achieve. Since our algorithm is iterative, it does not need to keep the data structures in main memory. We rely on the Map-Reduce framework and the transpose operation in order to keep the element–set memberships up-to-date.

Note also that the 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; we call this the *prefix-optimality property.* In fact, our algorithm will also enjoy this prefix-optimality property.

3. RELATED WORK

There are two principal lines of work that we need to address. The first is the extensive literature on the Max-k-Cover and related combinatorial optimization problems in sequential and parallel settings. The second is the burgeoning body of work on Map-Reduce-based algorithms for large web mining problems.

The complexity of Max-k-Cover**.** The study of Max-k-Cover and the related Set-Cover problem is classical. As we mentioned earlier, Max-k-Cover is NP-hard [15]. Hochbaum and Pathria [17] (see also [16]) present an elegant analysis of the Greedy algorithm for Max-k-Cover, proving its $(1 - 1/e)$-approximation guarantee. Khuller, Moss, and Naor [21] extended the Greedy algorithm to the budgeted case, maintaining the same approximation guarantee. Gandhi, Khuller, and Srinivasan [14] consider the related problem of choosing the minimum number of sets to cover at least k elements, which is harder to approximate than Set-Cover and therefore Max-k-Cover. It is also known that Max-k-Cover cannot be approximated to $1 - 1/e - \epsilon$, unless NP $\subseteq$ DTIME($n^{O(\log \log n)}$) [12, 21]. Thus the sequential complexity of Max-k-Cover is understood well. To the best of our knowledge, the parallel or distributed complexity of Max-k-Cover has not been examined yet.

The related Set-Cover problem has been studied in a parallel setting. Berger, Rompel, and Shor [2] give an NC algorithm that approximates Set-Cover to $O(\log n)$, which is optimal [1, 12]. The algorithm of Berger et al. uses a number of processors linear in the number of sets, and runs in $O(\text{polylog}(nm))$ rounds. Even though our Max-k-Cover algorithm is inspired by theirs, our analysis needs new ideas since our setting is different from theirs.

Max-k-Cover-based framework has been used a lot in many data mining applications; earliest ones include identifying the most influential users in social networks, under a model of influence propagation [20, 24, 5].

Algorithms in the Map-Reduce **model.** With an increasing demand to mine large amounts of web-originated data, the Map-Reduce paradigm of computation [9] and Map-Reduce-based algorithms have come to the rescue. For example, PageRank, and in fact most matrix-vector based iterative algorithms, have efficient Map-Reduce implementations as long as the matrix is reasonably sparse. Map-Reduce-based realizations of existing algorithms have been extensively developed in machine learning, including algorithms for regression, naive Bayes, k-means clustering, principal and independent component analysis, EM, and SVM [6]; see also [22, 10]. Likewise, in text and natural language processing, Map-Reduce-based algorithms are constantly being developed. For example, algorithms have been developed for pair-wise document similarity [11], word co-occurrences [27], language modeling [3], and indexing [28]; for more details, see the essay [25], the recent tutorial [26], and the website `http://cluster-fork.info`. There have been some attempts to abstractly model Map-Reduce computations; see [13, 19].

There has been some work on developed Map-Reduce-based algorithms and heuristics for large scale graph mining problems. Tsourakakis et al. [18] propose heuristics for diameter estimation of large graphs. The problem of triangle-counting was considered by Tsourakakis [35] and Tsourakakis et al. [36]. Papadimitriou and Sun [32] develop algorithms for co-clustering. Rao and Yarowsky [33] obtain simple algorithms for ranking and semi-supervised classification on graphs. Very recently, Karloff, Suri, and Vassilvitskii [19] obtain an algorithm for finding the minimum spanning tree. The application of Map-Reduce model to graph algorithms has been somewhat limited (and to a certain extent, disappointing) so far. See the wishful thinking article by Cohen [7] and the inspiration and frustration expressed by Muthukrishnan [30].

There is a large and growing body of work on obtaining graph algorithms in the streaming and semi-streaming models; we refer to [29] for an overview.

4. THE MrGreedy ALGORITHM

In this section we develop the MrGreedy algorithm that tries to emulate the sequential Greedy algorithm for Max-k-Cover, *without making k sequential choices*. As mentioned earlier, the main idea is to add multiple sets to the solution in parallel, as long as it is appropriate. We will prove that MrGreedy in fact outputs a total ordering of the sets with the prefix-optimality property, i.e., for any $k' \leq k$, the first k' sets are an approximate solution to the Max-k'-Cover instance. For simplicity, we focus on the unweighted, unbudgeted version of the problem.

The MrGreedy algorithm is inspired by the Set-Cover algorithm of Berger, Rompel, and Shor [2]. Unfortunately, their algorithm and analysis cannot be used "as is" for two main reasons. First, they are tailored for Set-Cover and not Max-k-Cover. Second, their algorithm does not return a total ordering of the sets and hence does not enjoy the prefix-optimality property. Therefore, we need to modify their algorithm to work for Max-k-Cover, return an ordering on the sets, and require the ordering to be prefix-optimal.

To do these, we incorporate two main ideas. First, we show that it suffices to work with a version of Greedy that guarantees something weaker than the usual Greedy. Next, we show that a modification of the algorithm of [2] can mimic the weaker Greedy. We also incorporate the prefix-optimality requirement in this modification by imposing an ordering of the sets. At the end, we will prove that the chosen ordering is good enough for the algorithm to return the stated approximation for each k.

Suppose we fix a prefix of the first $i-1$ sets chosen by some approximate greedy algorithm $\mathcal{A}$ and Greedy. Now, let α_i be the ratio of the new elements covered by $\mathcal{A}$ to those covered by Greedy with their respective choice of the ith set. Let $\overline{\alpha}_i$ be the average of $\alpha_1, \ldots, \alpha_i$: $\overline{\alpha}_i = (1/i)\sum_{j=1}^{i} \alpha_j$. We show a simple fact about the $\overline{\alpha}_i$'s.

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 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
            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 $\mathcal{C}$
```

Lemma 3. *For each $i = 1, \ldots, k-1$, we have*

$$\left(1 - \frac{\overline{\alpha}_i}{k}\right)^i \cdot \left(1 - \frac{\alpha_{i+1}}{k}\right) \leq \left(1 - \frac{\overline{\alpha}_{i+1}}{k}\right)^{i+1}.$$

Proof. By the AM–GM inequality we have

$$\begin{aligned} &\left(1 - \frac{\overline{\alpha}_i}{k}\right)^i \cdot \left(1 - \frac{\alpha_{i+1}}{k}\right) \\ \leq\ & \left(\frac{i \cdot \left(1 - \frac{\overline{\alpha}_i}{k}\right) + \left(1 - \frac{\alpha_{i+1}}{k}\right)}{i+1}\right)^{i+1} \\ =\ & \left(1 - \frac{i \cdot \overline{\alpha}_i + \alpha_{i+1}}{(i+1)\cdot k}\right)^{i+1} \\ =\ & \left(1 - \frac{\overline{\alpha}_{i+1}}{k}\right)^{i+1}. \end{aligned}$$

□

The following lemma lower bounds the approximation ratio of $\mathcal{A}$ in terms of its $\overline{\alpha_i}$'s.

Lemma 4. *For each $k \geq 1$ it holds that the first k sets in the list $\mathcal{C}$ produced by $\mathcal{A}$ is a $1 - \exp(-\overline{\alpha}_k)$ approximate solution of the* Max-k-Cover *problem.*

Proof. Fix some $k \geq 1$ as the length of the prefix. We will mimic the well-known proof for sequential max cover, changing the inductive step proof, since our algorithm has a weaker guarantee than the usual sequential greedy for its selection step. Let $S'_{(1)}, \ldots, S'_{(k)}$ be the first k sets picked by

$\mathcal{A}$, $\mathcal{S}' = \{S'_{(1)}, \ldots, S'_{(k)}\}$, and let $S^*_1, \ldots, S^*_k$ be the k sets in the optimal solution (sorted arbitrarily), $\mathcal{S}^* = \{S^*_1, \ldots, S^*_k\}$.

Let ω_i be the number of elements that (i) are covered by set $S'_{(i)}$ and (ii) are *not* covered by sets $S'_{(1)}, \ldots, S'_{(i-1)}$. Then the coverage of the algorithm solution $S'_{(1)}, \ldots, S'_{(k)}$ is $\mathrm{cov}(\mathcal{S}') = \sum_{i=1}^{k} \omega_i$. We want to lower bound $\mathrm{cov}(\mathcal{S}')/\mathrm{cov}(\mathcal{S}^*)$.

Fix $i \geq 1$, and consider the first $i-1$ sets chosen by the algorithm; the total number of elements covered by $\mathcal{S}^*$, that are not covered by the sets $S'_{(1)}, \ldots, S'_{(i-1)}$, is at least $\mathrm{cov}(\mathcal{S}^*) - \sum_{j=1}^{i-1} \omega_j$. This implies that there must exist some set S^* in the optimal solution, different from $S'_{(1)}, \ldots, S'_{(i-1)}$, that covers a subset of those elements having cardinality at least $(1/k)(\mathrm{cov}(\mathcal{S}^*) - \sum_{j=1}^{i-1} \omega_j)$. Suppose that the algorithm will choose as the ith set an α_i approximation of the "best" set, i.e.,

$$\omega_i \geq \alpha_i \cdot \max_{S \in \mathcal{S}} w\left(S \setminus \bigcup_{j=1}^{i-1} S_{(j)}\right). \qquad (1)$$

Lower bounding the max in (1) with its value at S^*, we obtain

$$\omega_i \geq \alpha_i \cdot \frac{\mathrm{cov}(\mathcal{S}^*) - \sum_{j=1}^{i-1} \omega_j}{k}.$$

We show by induction that

$$\sum_{j=1}^{i} \omega_j \geq \left(1 - \left(1 - \frac{\overline{\alpha}_i}{k}\right)^i\right) \cdot \mathrm{cov}(\mathcal{S}^*),$$

for each $i = 1, \ldots, k$. This directly implies the main statement since $(1 - x/k)^k \leq \exp(-x)$, for any $k \geq 1, x \geq 0$.

The base case is trivial, since

$$\omega_1 \geq \alpha_1 \cdot \frac{\mathrm{cov}(\mathcal{S}^*)}{k} = \overline{\alpha}_1 \cdot \frac{\mathrm{cov}(\mathcal{S}^*)}{k}.$$

Now, suppose the property holds for $i \geq 1$, then

$$\begin{aligned}
\sum_{j=1}^{i+1} \omega_j &= \sum_{j=1}^{i} \omega_j + \omega_{i+1} \\
&\geq \sum_{j=1}^{i} \omega_j + \alpha_{i+1} \cdot \frac{\mathrm{cov}(\mathcal{S}^*) - \sum_{j=1}^{i} \omega_j}{k} \\
&= \left(1 - \frac{\alpha_{i+1}}{k}\right) \sum_{j=1}^{i} \omega_j + \alpha_{i+1} \cdot \frac{\mathrm{cov}(\mathcal{S}^*)}{k} \\
&\geq \left(1 - \frac{\alpha_{i+1}}{k}\right) \cdot \left(1 - \left(1 - \frac{\overline{\alpha}_i}{k}\right)^i\right) \cdot \mathrm{cov}(\mathcal{S}^*) \\
&\quad + \alpha_{i+1} \cdot \frac{\mathrm{cov}(\mathcal{S}^*)}{k} \\
&\geq \left(1 - \frac{\alpha_{i+1}}{k}\right) \cdot \mathrm{cov}(\mathcal{S}^*) - \left(1 - \frac{\overline{\alpha}_{i+1}}{k}\right)^{i+1} \cdot \mathrm{cov}(\mathcal{S}^*) \\
&\quad + \alpha_{i+1} \cdot \frac{\mathrm{cov}(\mathcal{S}^*)}{k} \\
&= \left(1 - \left(1 - \frac{\overline{\alpha}_{i+1}}{k}\right)^{i+1}\right) \cdot \mathrm{cov}(\mathcal{S}^*),
\end{aligned}$$

where the first inequality follows from (1), the second inequality follows from the induction hypothesis, and the third inequality follows from Lemma 3.

The proof is complete. □

Now that we have determined the approximation ratio of $\mathcal{A}$ in terms of its α_i's, we turn to analyze MrGreedy. We start by showing that the number of bad sets in line 13 is small — this property will be crucial in our main approximation theorem.

Lemma 5. *At line 13, the number of bad sets is at most* $4\epsilon \cdot |\mathcal{S}_p|$.

Proof. Observe that, at any execution of line 12, each set in $\mathcal{S}_w$ has weight upper bounded by $(1+\epsilon^2)^i$. Indeed, this is trivially true when $i = \lceil \log_{1+\epsilon^2} |X| \rceil$, since in this case the upper bound is greater than the size of the ground set, $(1+\epsilon^2)^i \geq |X|$. Now, observe that for i to be decreased, the loop on j must be exhausted. Further, when $j = 1$, the only way of exiting the loop at line 6, i.e., the only way of letting $X' = \emptyset$ is to remove all sets of weight at least $(1+\epsilon^2)^{i-1}$ (lines 18, 19). Thus, whenever we execute line 3 and line 12, no set of weight more than $(1+\epsilon^2)^i$ exists.

Now, at any execution of line 11, we have $\mathcal{S}_p \subseteq \mathcal{S}_w$, and thus $\forall S \in \mathcal{S}_p$ we have $(1+\epsilon^2)^{i-1} \leq |S| < (1+\epsilon^2)^i$. Then, $\sum_{S \in \mathcal{S}_p} |S| \leq |\mathcal{S}_p| \cdot (1+\epsilon^2)^i$. If the test at line 11 is negative, we just cycle all the way through line 10 without changing the state of the algorithm. When we do get to line 12, we have $\left|\bigcup_{S \in \mathcal{S}_p} S\right| \geq |\mathcal{S}_p| \cdot (1+\epsilon^2)^i \cdot (1 - 8\epsilon^2)$. For each element that occurs in more than one set in $\mathcal{S}_p$, mark all the occurrences of that element. Observe that the bad elements of line 12 are exactly those whose occurrences are marked. The total number of the marked occurrences is

$$T \leq 2 \cdot \left(\sum_{S \in \mathcal{S}_p} |S| - \left|\bigcup_{S \in \mathcal{S}_p} S\right|\right) \leq |\mathcal{S}_p| \cdot (1+\epsilon^2)^i \cdot 16\epsilon^2.$$

Observe that no more than $4\epsilon \cdot |\mathcal{S}_p|$ sets in $\mathcal{S}_p$ can contain more than $4\epsilon \cdot (1+\epsilon)^i$ marked elements, since otherwise we would have a contradiction with the upper bound on T. That is, no more than $4\epsilon \cdot |\mathcal{S}_p|$ sets can be bad. □

To fill the final missing step, we now prove a lower bound on $\overline{\alpha}_k$'s of MrGreedy.

Lemma 6. *For each* $k \geq 1$*, it holds that* $\overline{\alpha}_k \geq 1 - O(\epsilon)$.

Proof. Fix an arbitrary $1 \leq t \leq k$. If we add the tth set to $\mathcal{C}$ via line 8, we will have $\alpha_t \geq \frac{1}{1+\epsilon^2} \geq 1 - O(\epsilon)$. Indeed, at the beginning of the loop at line 6, the largest uncovered weight of the remaining sets will be $\leq (1+\epsilon^2)^i$, and each set in $\mathcal{S}_w$, thus in particular the one being added to $\mathcal{C}$ via line 8, has uncovered weight at least $(1+\epsilon^2)^{i-1}$; thus $\alpha_t \geq 1 - O(\epsilon)$.

On the other hand, whenever we add two batches of sets via lines 14, 15, the following will happen. The first batch (line 14) will be composed of sets each containing a number of unique elements at least $(1+\epsilon^2)^{i-1} - 4\epsilon \cdot (1+\epsilon^2)^i = (1 - O(\epsilon)) \cdot (1+\epsilon^2)^{i-1}$. Since the largest number of uncovered elements of a yet-to-be-taken set is at most $(1+\epsilon^2)^i$, no matter how we sort the sets in the first batch, each of them will have an $\alpha_t \geq \frac{1-O(\epsilon)}{1+\epsilon^2} = 1 - O(\epsilon)$. As for the second batch, we lower bound the α_t's of its sets with 0.

Applying Lemma 5, we can upper bound the number of sets in the second batch by at most $4\epsilon \cdot |\mathcal{S}_p|$ so that there are at least $|\mathcal{S}_p| \cdot (1 - 4\epsilon) = (1 - O(\epsilon)) \cdot |\mathcal{S}_p|$ sets in the first batch.

Thus, if lines 14 and 15 add sets starting from position t_0 of $\mathcal{C}$, we have $\alpha_{t_0}, \ldots, \alpha_{t_0 + (1-O(\epsilon)) \cdot |\mathcal{S}_p| - 1} \geq 1 - O(\epsilon)$, and

$\alpha_{t_0+(1-O(\epsilon))\cdot|\mathcal{S}_p|}, \ldots, \alpha_{t_0+|\mathcal{S}_p|-1} \geq 0$. It is easy to see that a lower bound on the average of the α_t's in any prefix of $\alpha_{t_0}, \ldots, \alpha_{t_0+|\mathcal{S}_p|-1}$ is

$$\frac{(1-O(\epsilon)) \cdot |\mathcal{S}_p| \cdot (1-O(\epsilon))}{|\mathcal{S}_p|} = 1 - O(\epsilon).$$

Since (i) we can cut C into the sequences added by line 8, or by lines 14–15, and (ii) each sequence contains sets whose average α_t's are at least $1 - O(\epsilon)$, and (iii) the lower bound on the average α_t of a part holds even if we cut that part to any of its prefix, the lemma is proved: for each k, $\overline{\alpha}_k \geq 1 - O(\epsilon)$. □

Combining Lemmas 4 and 6, with $\mathcal{A}$ set to MrGreedy, gives us the main result.

Theorem 7. *The approximation guarantee of* MrGreedy *is $1 - 1/e - O(\epsilon)$.*

Let us now comment on the number of Map-Reduce passes of MrGreedy.

Lemma 8. *The number of times lines 7–15 is executed is $O(\epsilon^{-10} \cdot \log^2 n \cdot \log \Delta)$, with high probability.*

Proof. To prove this, we first need a fact about the sampling step of MrGreedy.

Lemma 9 ([2]). *With probability at least $1/8$ (over the random sampling), (i) the test at line 11 succeeds and (ii) the elements covered by $\left|\bigcup_{S \in \mathcal{S}_p} S\right| \geq |\mathcal{S}_p|$ contain at least a $\epsilon^2/2$ fraction of X'.*

(In the algorithm of [2], there are two independent parameters ϵ and δ. For simplicity, we set $\epsilon = \delta$ since the approximation guarantee will be maximized when they are equal. Further, we set our ϵ to be the square root of the $\epsilon = \delta$ in [2].)

From Lemma 9, independent of the result of the test at line 7, with probability $\Omega(1)$, an $\Omega(\epsilon^6)$ fraction of X' will be removed from consideration. Since at the beginning $|X'| \leq n$, by a Chernoff bound, with high probability the loop at line 6 will end after $O(\epsilon^{-6} \log n)$ iterations. Observing that the loop at line 4 iterates for $O(\epsilon^{-2} \log \Delta)$ times, and that the loop at line 2 iterates for $O(\epsilon^{-2} \log n)$ times, gives the stated upper bound on the number of executions of lines 7–15. □

4.1 Realization in Map-Reduce

In this section we comment on the implementation of MrGreedy in Map-Reduce. Recall the transpose operation we discussed earlier, which lets us switch between element–set and set–element representations using Map-Reduce.

The steps involving the selection of S_w and X' in the algorithm (lines 3 and 5) can be realized by a combination of map, transpose, and a reduce operation. Line 7 can be realized by computing the size of an intersection, which can be done with a map and a reduce. The random selection in line 10 is a simple map operation and the test in line 11 is equivalent to computing the size of a union, which is once again a transpose, map, followed by a transpose. Lines 13–15 that determine the goodness of sets and elements can be determined using a transpose and a map operation. Finally, the updates in lines 16-17 are once again easy in Map-Reduce using a transpose, map, and reduce.

4.2 Weighted, budgeted versions

We observe how the weighted case can be easily reduced to the unweighted one. For instance, assuming integer weights, we could, for each element x, replace x (in all the sets that contain it) with $w(x)$ unweighted copies of x. Then, for each class $\mathcal{S}' \subseteq \mathcal{S}$ of sets, the weight of $\mathcal{S}'$ in the original instance will equal the total coverage of $\mathcal{S}'$ in its unweighted counterpart. This reduction has two downsides: it is not strongly polynomial and it requires each element weight to be integral.

It is possible to overcome these hindrances using another reduction. First of all, observe that multiplying all the weights by the same positive number only scales the value of each solution by that same number. Further, if $W = \max_{x \in X} w(x)$, then one can easily check that removing all elements of weight less than $(\epsilon \cdot W)/n$ from X and from the sets that contain them in $\mathcal{S}$ changes the value of each solution of value with a constant factor of the optimum by a $1 \pm O(\epsilon)$ factor. Finally, rounding each weight $w(x)$ to $\lceil \frac{w(x)}{\epsilon} \rceil$ changes the value of each solution by a $(1 \pm O(\epsilon))$ factor. So, suppose we rescale all weights so that the maximum weight becomes $\epsilon^{-2} \cdot n$. Then, we delete all elements having new weight less than ϵ^{-1}, and we round each remaining weight w to $\lceil \frac{w}{\epsilon} \rceil$, losing only a $(1 \pm O(\epsilon))$ approximation factor. Observe that the new weights are all integers. Substituting an element x of integral weight $w \in \mathbf{N}$ with w new elements each of weight 1 and each contained in all the sets that contained x, does not change the value of any solution. The reduction is complete. The downside of this reduction is that the number of elements gets squared, which may not be desirable in practice.

Now we comment on the budgeted version of Max-k-Cover. Khuller et al. [21] show how a budgeted version of Greedy, along with a simple external check, gives a $(1 - 1/\sqrt{e})$-approximation to the budgeted, weighted, Max-k-Cover problem (in fact, this approximation is obtained if one returns the best of the solution produced by the algorithm, and the set of maximum weight among those having cost less than or equal to the budget). Using an argument similar to the one we gave for the unbudgeted case, we can show that the budgeted greedy algorithm of Khuller et al. can be parallelized like the unbudgeted one. The approximation guarantee is $(1 - 1/\sqrt{e} - O(\epsilon))$ and the parallel running time remains polylogarithmic. We omit the details.

4.3 Discussions

In this section we comment on other possible approaches for obtaining a parallel approximation algorithm for Max-k-Cover. Suppose one could strengthen our weak version of greedy algorithm to return a set covering a number of new elements within a factor of $1 - \epsilon$ of the maximum. Then, a result of Hochbaum and Patria [16, Chapter 3, Theorem 3.9] could directly be applied to obtain a $1 - e^{-1+\epsilon} = 1 - e^{-1} - O(\epsilon)$ approximation to the Max-k-Cover problem. It is in fact unclear if such an algorithm can exist in the parallel setting. For instance, the algorithm of [2] *can* choose some bad sets (i.e., sets having large intersection with the other chosen sets). Our contribution is to show that (i) such sets are few and (ii) can be positioned in the total ordering in such a way that the approximation guarantee is changed by just a $1 - O(\epsilon)$ factor.

We observe how a partially different approach could have been taken if we did not insist on prefix-optimality. For instance, we could have randomly permuted each equivalence class of sets returned by the algorithm of [2]; this would have ensured that with some $\Omega(1)$ probability few bad sets would have ended in the k-prefix of the ordering — we could have then ignored those few bad sets and considered only the others. This approach would have still required to prove that the number of bad sets is small, and that, being few, they could not have caused havoc.

5. EXPERIMENTS

In this section we detail an experimental study of our new algorithm. For purposes of the experiments, we use five large, real-world instances of set systems. For each of these instances, we ran the standard Greedy algorithm and our MrGreedy algorithm. The goal of the experiments is to demonstrate three things. First, it is feasible to implement MrGreedy in practice. Second, the performance of MrGreedy is almost indistinguishable from Greedy, both for various values of k and for instances with various characteristics. Third, our algorithm exploits and achieves parallelism in practice.

5.1 Data description

We use the following five data sets in our experiments.

(1) *User–hosts.* This instance consists of users (elements) and hosts (sets). Here, the set for a given host consists of all the users who visited some web page on the host during browsing. This data is derived from the Yahoo! toolbar logs; the users are anonymized and the hosts hashed. The instance is a subset of the toolbar data during the week of July 18-25, 2009. The coverage problem is to determine the hosts that are visited by as many users as possible.

(2) *Query–hosts.* Our second instance consists of queries (elements) and hosts (sets). The set for a given host contains all queries for which the host appeared in the top ten search results. These queries were sub-sampled from Yahoo! query logs on July 1, 2009. We may study both weighted and unweighted versions by introducing a weight to each query corresponding to the number of times it occurs. We work here with the unweighted instance, so we seek the hosts that together cover as many unique queries as possible.

(3) *Photos–tags.* Our third graph is derived from the Flickr data set. Here, the elements are photographs uploaded to the Flickr web service, and the sets are tags. The set for a given tag contains the photos to which that tag has been applied. The coverage problem is to find those tags that together cover as many photos as possible.

(4) *Page–ads.* Our fourth instance is derived from Yahoo!'s *content match* advertising system, which allows arbitrary internet content publishers to place textual ads on their pages. The elements are web pages, and the sets are the ads shown on those pages. The coverage problem is to determine a collection of ads that covers as many pages as possible.

(5) *User–queries.* Our fifth graph is derived once again from the web search query logs in Yahoo! We consider query logs over a 75-day period in which each query is annotated with an anonymized userid. The elements are the anonymized ids, and the sets are queries. The set for a particular query contains all the anonymized userids that issued the query in our sample. The coverage problem is to determine the queries that cover as many users as possible.

Table 1 shows the overall statistics of the data including the number of sets (m), number of elements (n), the number of element–set memberships (E), the maximum degree of an element (Δ), and the maximum cardinality of a set ($w^*(S)$). From the table it is clear that the five instances are widely varying in terms of their characteristics. Unless otherwise specified, we use the large User–hosts instance to illustrate our experiments.

Data set	m	n	E	Δ	$w^*(S)$
User–hosts	5.64M	2.96M	72.8M	2,115	1.19M
Query–hosts	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

Table 1: Details of the data sets.

5.2 Approximation performance

First, we wish to show that the approximation guarantee of MrGreedy is almost on par with Greedy. For the purposes of this experiment, we choose $\epsilon = .75$. (In theory, this is a non-sensical choice since ϵ has to be at most $1 - 1/e$; but as we will see, this choice does not impact the algorithm.) We also illustrate the performance of the naive algorithm (Naive) that sorts the sets by sizes and takes the prefix as a solution.

We plot the value of the solution returned by the various algorithms on the instances. We choose varying values of k for each instance. In Figure 1, we show the approximations on the User–hosts instance. The x-axis specifies k and the y-axis gives the fraction of elements in the universe that are covered by a prefix of length k in the solution. It is clear that MrGreedy is almost indistinguishable from Greedy for all values of k.

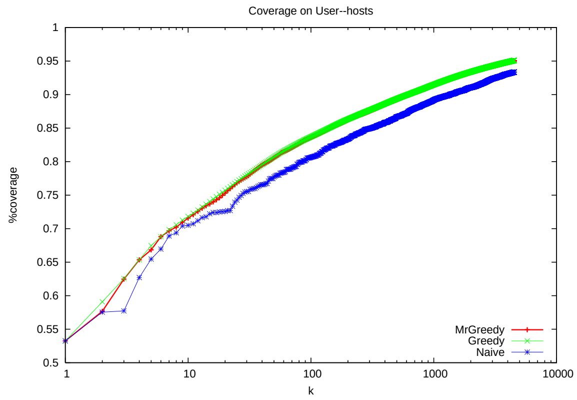


Figure 1: Performance of MrGreedy, Greedy, and Naive on User–hosts.

To visualize the plot better, in Figure 2, we show the relative performances, where we compare the approximation of Greedy and Naive against MrGreedy. The relative performances on the other three instances are shown in Figure 3. The top line shows the performance of Naive against MrGreedy whereas the bottom line shows the performance of Greedy against MrGreedy. As we see, the bottom line is nearly horizontal and very close to $y = 1$, indicating that the performance of Greedy and MrGreedy are almost indistinguishable.

The reader might wonder why some of the plots in Figure 2 and Figure 3 have a "saw tooth" pattern. This is caused due to the parallel nature of MrGreedy and each "spike" corresponds to the beginning of a book-keeping phases. MrGreedy arbitrarily sorts the sets output during a book-keeping phase to increase parallelism at the cost of precision, i.e., if a prefix that cuts a phase in two parts is chosen, the algorithm incurs a small loss in the quality of the cover returned. Both Greedy and Naive return the sets in a total order instead, i.e., each prefix is correctly sorted. Thus, the relative performances of MrGreedy, compared to both Greedy and Naive, decreases between book-keeping phases.

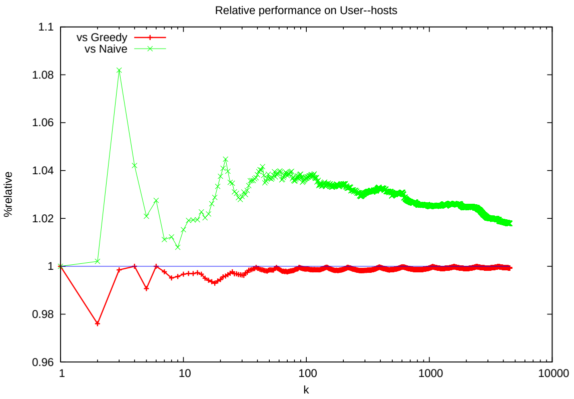


Figure 2: Relative performance on User–hosts.

The performance of the Naive algorithm varies widely. In some instances, such as the Query–hosts dataset, the performance of Naive is within a few percent of the other algorithms, which may be acceptable for some settings. In other settings, such as the Photos–tags dataset, the approximation performance of Naive is quite poor and worsens as the result size increases. Naive in general will perform poorly when there are many high-coverage sets with significant overlap, which is a common occurrence in max cover problems; hence, algorithms that take overlaps into account are usually necessary.

Comparing the performance of MrGreedy with Greedy, we see that MrGreedy performs comparably over all instances and all ranges of k. So, in terms of approximation guarantees, we conclude that MrGreedy performs on par with Greedy, despite our setting of $\epsilon = 0.75$. Even if choosing such a large ϵ does not give us, theoretically, any approximation guarantee (see Theorem 7 and Lemma 5), it produces very good results in practice. In the next section, we further study the role of ϵ in the experimental evaluation of MrGreedy.

5.3 Effect of ϵ

In this section we study the effect of choosing the value of ϵ. We examine two parameters: the coverage obtained and the running time. Figure 4 shows the relative coverage (base is when $\epsilon = 0.75$) for two smaller values of ϵ, namely, 0.1 and 0.01, on the User–hosts instance. From the curves, we see how a smaller value of ϵ achieves only a 1–2% improvement, even for moderate values of k. On the other hand, a smaller value of ϵ means a higher running time. Clearly, the benefit of using a lower value of ϵ is unnoticeable in practice. In other words, even though the performance of our algorithm is $1 - 1/e - O(\epsilon)$, the effect of a large value of ϵ seems negligible in practice.

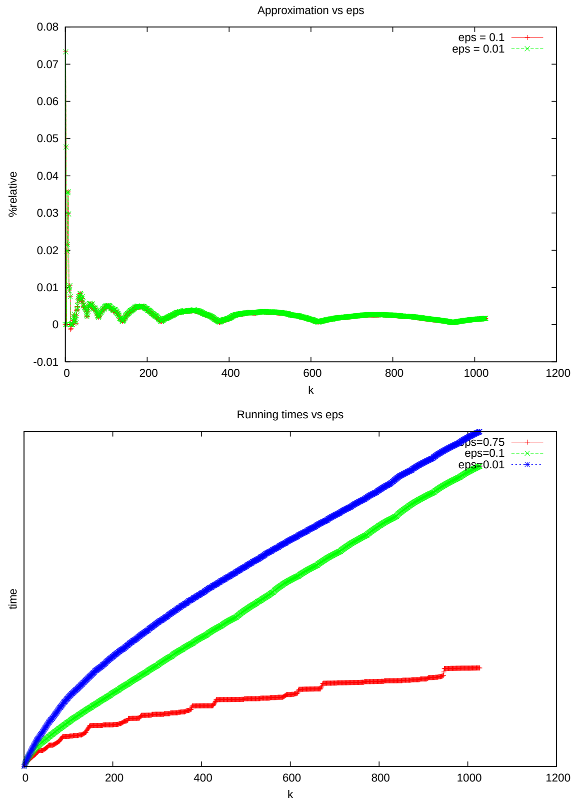
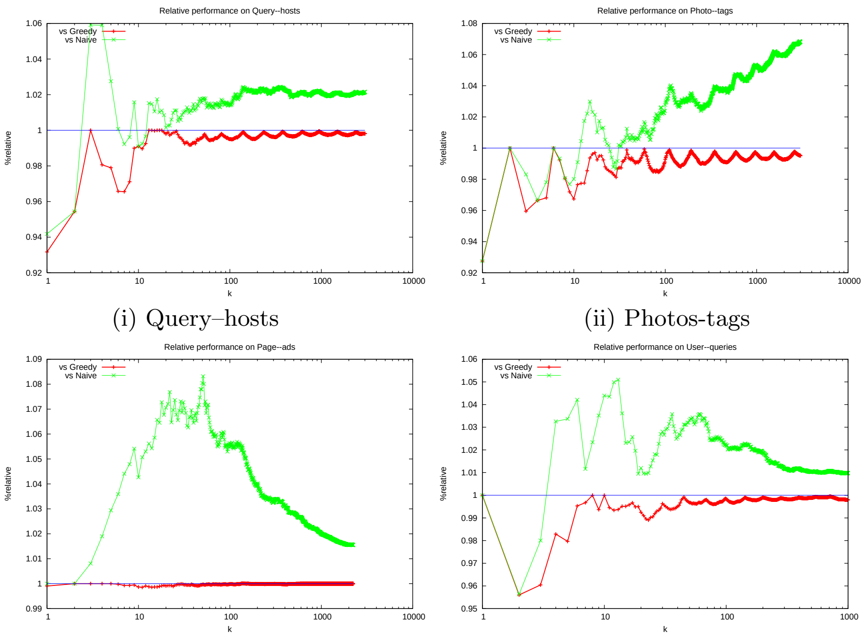


Figure 4: Effect of ϵ: relative deviation when using $\epsilon = 0.75$ and running times for different ϵ on Query–hosts.

5.4 Exploiting the parallelism

In this section we study the running time of MrGreedy compared to Greedy. Specifically, we study two aspects: the wall-clock time and the number of parallel executions made possible by our algorithm (as a function of k). Figure 5 shows the running time of MrGreedy vs Greedy on the User–hosts graph. It is clear that MrGreedy vastly outperforms Greedy in terms of the running time. Observe the "horizontal steps" in the running of MrGreedy. These are precisely the points where a "batch" addition to the current solution is performed: lines 14–15 of MrGreedy.

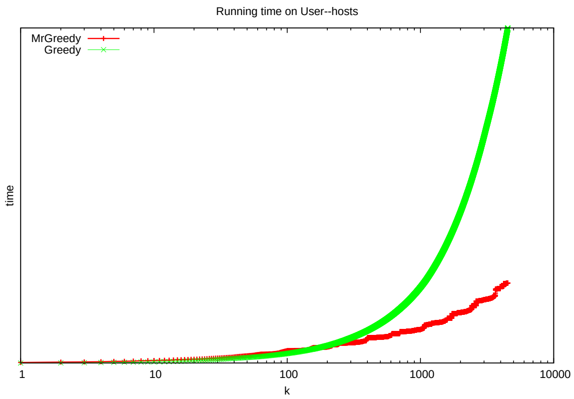
To study this further, we count the number of parallel steps enabled by MrGreedy; in particular, we study the number of times lines 14–15 were invoked for the User–hosts instance. In all, these were invoked about 90 times and more than 2,400 sets were added in that step. Note that each of these 90 additions is a "batch" update, i.e., the element–set memberships have to be updated only 90 times (as opposed to each time in Greedy). Figure 6 shows these updates over time. It is clear that as the algorithm progresses, on average, it is able to add more and more sets to the solution in a single batch.



(i) Query–hosts (ii) Photos-tags

(iii) Page-ads (iv) User-queries

Figure 3: Relative performance on other four data sets.



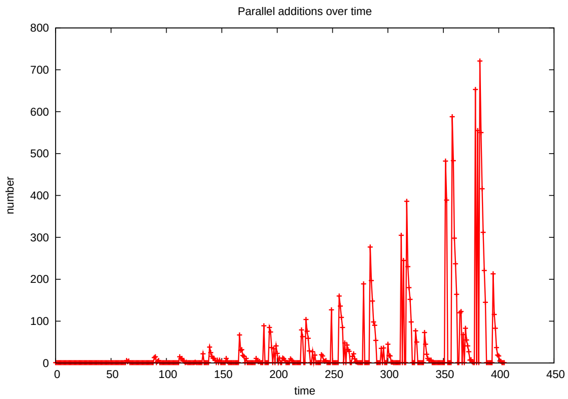


Figure 5: Running times for MrGreedy vs Greedy on User-hosts, on a semi-log scale (time vs. log-k).

Figure 6: Number of "batch" additions over time for MrGreedy on User–hosts.

6. CONCLUSIONS

We developed a Map-Reduce-based algorithm for Max-k-Cover. Our algorithm obtains a nearly 0.63-approximation and can be implemented with polylogarithmic many Map-Reduce steps over the input instance. Thus, we match the performance of Greedy, while obtaining an algorithm that can be implemented in the scalable and widely-used Map-Reduce framework. This is one of the few instances of a non-trivial Map-Reduce realization of a classical algorithm with provable guarantees of performance. Our experiments on five large-scale real-world instances show that it is feasible to implement MrGreedy in practice and obtain reasonable speedup over Greedy. Moreover, the approximation performance of MrGreedy is virtually indistinguishable from that of Greedy.

As we stated in Section 3, there has been very little work on Map-Reduce-versions of classical graph algorithms. With the growth of graph mining in web contexts, it becomes inevitable to focus on developing Map-Reduce-friendly versions of those graph algorithms that have an inherently sequential flavor, with provable performance and running time guarantees. A viable and promising candidate in this aspect is the sequential greedy algorithm to find the densest subgraph [4]: a Map-Reduce version of this algorithm will be of immense interest to practitioners who are interested in finding dense communities in massive graphs.

7. REFERENCES

[1] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, 2006.

[2] B. Berger, J. Rompel, and P. W. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. *J. Comput. Syst. Sci.*, 49(3):454–477, 1994.

[3] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large language models in machine translation. In *Proc. EMNLP*, pages 858–867, 2007.

[4] M. Charikar. Greedy approximations for finding dense components in a graph. In *Proc. 3rd APPROX*, pages 84–95, 2000.

[5] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *Proc. 15th KDD*, pages 199–208, 2009.

[6] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Proc. NIPS*, pages 281–288, 2006.

[7] J. Cohen. Graph twiddling in a MapReduce world. *Computing in Science and Engineering*, 11(4):29–41, 2009.

[8] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins. The discoverability of the web. In *Proc. 16th WWW*, pages 421–430, 2007.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *C. ACM*, 51:107–113, 2008.

[10] J. Ekanayake, S. Pallickara, and G. Fox. MapReduce for data intensive scientific analyses. In *IEEE Fourth International Conference on eScience*, pages 277–284, 2008.

[11] T. Elsayed, J. Lin, and D. W. Oard. Pairwise document similarity in large collections with MapReduce. In *Proc. 46th ACL/HLT*, pages 265–268, 2008.

[12] U. Feige. A threshold of $\ln n$ for approximationg set cover. *J. ACM*, 45:634–652, 1988.

[13] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. In *Proc. 19th SODA*, pages 710–719, 2008.

[14] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 2(1):55–84, 2004.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[16] D. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. Course Technology, July 1996.

[17] D. Hochbaum and A. Pathria. Analysis of the greedy approach in covering problems, 1994. Unpublished manuscript.

[18] U. Kang, C. E. Tsourakakis, A. Appel, C. Faloutsos, and J. Leskovec. HADI: Fast diameter estimation and mining in massive graphs with Hadoop. Technical Report CMU-ML-08-117, CMU, 2008.

[19] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proc. 20th SODA*, 2010.

[20] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proc. 9th KDD*, pages 137–146, 2003.

[21] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[22] A. Kimball. Parallel graph algorithms with MapReduce. `http://youtube.com/watch?v=BT-piFBP4fE`.

[23] R. Kumar, A. Tomkins, and E. Vee. Connectivity structure of bipartite graphs via the KNC-plot. In *Proc. 1st WSDM*, pages 129–138, 2008.

[24] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *Proc. 13th KDD*, pages 420–429, 2007.

[25] J. Lin. Exploring large-data issues in the curriculum: A case study with MapReduce. In *3rd Workshop on Issues in Teaching Computational Linguistics (TeachCL-08) at ACL*, pages 54–61, 2008.

[26] J. Lin and C. Dyer. Text processing with MapReduce, 2009. NAACL/HLT Tutorial, `http://www.umiacs.umd.edu/~jimmylin/cloud-computing/NAACL-HLT-2009/index.html`.

[27] J. J. Lin. Scalable language processing algorithms for the masses: A case study in computing word co-occurrence matrices with MapReduce. In *Proc. EMNLP*, pages 419–428, 2008.

[28] R. M. C. McCreadie, C. Macdonald, and I. Ounis. On single-pass indexing with MapReduce. In *Proc. 32nd SIGIR*, pages 742–743, 2009.

[29] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[30] S. Muthukrishnan. MapReduce again, 2008. `http://mysliceofpizza.blogspot.com/2008/01/mapreduce-again.html`.

[31] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1999.

[32] S. Papadimitriou and J. Sun. Disco: Distributed co-clustering with Map-Reduce: A case study towards petabyte-scale end-to-end mining. In *Proc. of 8th ICDM*, pages 512–521, 2008.

[33] D. Rao and D. Yarowsky. Ranking and semi-supervised classification on large scale graphs using Map-Reduce. In *Proc. 4th TextGraphs at ACL/IJCNLP*, 2009.

[34] B. Saha and L. Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *Proc. 9th SDM*, pages 697–708, 2008.

[35] C. E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *Proc. 8th ICDM*, pages 608–617, 2008.

[36] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. DOULION: Counting triangles in massive graphs with a coin. In *Proc. 15th KDD*, pages 837–846, 2009.