On Maximum Coverage in the Streaming Model & Application to Multi-topic Blog-Watch

Barna Saha
Department of Computer Science
University of Maryland
College Park, MD 20742, USA
barna@cs.umd.edu

Lise Getoor
Department of Computer Science
University of Maryland
College Park, MD 20742, USA
getoor@cs.umd.edu

Abstract

We generalize the graph streaming model to hypergraphs. In this streaming model, hyperedges are arriving online and any computation has to be done on-the-fly using a small amount of space. Each hyperedge can be viewed as a set of elements (nodes), so we refer to our proposed model as the "set-streaming" model of computation. We consider the problem of "maximum coverage", in which k sets have to be selected that maximize the total weight of the covered elements. In the set-streaming model of computation, we show that our algorithm for maximum-coverage achieves an approximation factor of $\frac{1}{4}$. When multiple passes are allowed, we also provide a $\Theta(\log n)$ approximation algorithm for the set-cover. We next consider a multi-topic blog-watch application, an extension of blog-alert like applications for handling simultaneous multiple-topic requests. We show how the problems of maximum-coverage and set-cover in the set-streaming model can be utilized to give efficient online solutions to this problem. We verify the effectiveness of our methods both on synthetic and real weblog data.

1 Introduction

The data streaming model has gained popularity for a variety of monitoring applications where a large volume of data is generated rapidly and continuously, and must be analyzed on the fly, using space significantly sublinear in the data size. An emerging class of such monitoring applications deal with massive dynamic graphs. For example the web graph, where each web page represents a node and each hyperlink represents an edge. The edges in the web graph are generated in a streaming fashion by a web-crawler. As a result, different models of graph streaming have been developed [9, 18, 8, 6, 5] in the past decade to handle such voluminous edge streams.

Hypergraphs are a natural extension of graphs, where an edge can represent any collection of vertices. Hypergraphs capture more complex relationships between objects and have been used in different application areas of computer science including vision [23], network design [1], system architecture [17] etc. For example, the blogosphere can be modeled as a hypergraph, where nodes are topics and each blog is represented by a hyperedge describing the collection of topics that the blog covers. Different statistical properties of this hypergraph reveal important information about topical behavior of the blogosphere. Another example is the community affiliation network of the social network graph, where each node represents a member of the social network and each community is represented by a hyperedge covering the members of the community. All these networks are massive in size and growing continuously. New web-logs and communities are being created every moment and thus they are generating a huge stream of hyperedges, which if processed efficiently will reveal deep structural properties of these networks.

We thus extend the graph-streaming model to hypergraphs, where hyperedges are streaming in. Since each hyperedge can be viewed as a set of vertices, we call our new model, the *set-streaming* model (SS-model) of computation. We consider two problems in this model, the *maximum coverage* problem [12] and the *set cover* problem [14]. A maximum coverage problem is defined as follows: a collection of sets $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ is defined over a domain of elements $\mathcal{E} = \{e_1, e_2, ..., e_n\}$ with associated weights $\{w_i\}_{i=1}^{n}$. The goal is to find k sets that maximize the total weight of the elements covered. For set cover, we are interested in finding minimum number of sets which cover all the elements. In our model, sets are coming online as a stream and for the constraint of space a set not explicitly stored in the main memory at its arrival time, cannot be accessed in future. Solutions to these problems in the SS-model will be useful for applications in scientific navigational queries [13], in multi-topic blog alert applications, in community search on social networks and in many other applications where the underlying data resource can be viewed as a dynamic hypergraph.

In this paper, we give new algorithms for the maximum coverage and the set cover problem in the SS-model with strong theoretical guarantees. In addition, we elaborate on one of their application in blog-monitoring, which we call (*Blog-Watch*). We briefly motivate the *Blog-Watch* application here and then describe it in detail in Section 3.1.

Blog-Watch: The blogosphere hosts millions of blogs and grows everyday. Each blog contains posts on multiple topics and the topics show a rich variation from one blog to another. Sometimes blogs have overlapping topics of interest, sometimes they are disjoint.

Given this scenario, users are likely to be interested in the following types of questions:

- Find at most k blogs to read interesting articles on a list of topics.
- Find a minimum collection of blogs which contains relevant articles on a list of topics.

The challenge is how these questions can be answered and maintained over time. In blog alert like applications when user requests a single alert for more than one topic, the performance is very poor. Therefore for obtaining good results, users need to create separate alert for every topic of their interest, which is certainly not desirable. Blog search applications as well cannot handle simultaneous search on multiple topics. For example, when searched with topics music,travel, book and sports in a single search query (on 15th February 2008), the Google blog search returned the top-4 blogs as "www.wakulla.com", "www.dumblittleman.com", "www.cooking guidelive.com" and "fundraiseitforward.com". None of these seem to match any of the topics of music, travel, book or sports.

The expanding size of the blogosphere and its changing nature makes the problem even more difficult. New blogs are constantly added, and new topics are emerging. The best set of blogs that is now related to a user's topics of interest will change and evolve over time. In a rapid changing scenario like this, an offline algorithm that stores all the blogs and needs to re-evaluate the result every time there is an addition of a blog or a post within a blog will not be practical in terms of both space and time. Rerunning the offline algorithm periodically might miss some important discussions on relevant topics if the period is not small enough. Thus to maintain an updated list at all time efficiently, we need algorithms that process the updates in blogs incrementally. Also since the size of the total blogosphere is huge, the algorithms must analyze blog-streams using very little in-memory space. Therefore the algorithms cannot afford to load all the blogs in memory or do heavy disk IO operations for every update.

This problem can be solved efficiently, as we will see in Section 3.1, by maintaining relevant statistics over the blogs in a carefully designed data-structure and exploiting the solutions of *maximum coverage* problem and the *set cover* problem in the *Set Streaming* model (SS-model) of computation.

Contributions: The contributions of this paper include:

1. We propose a new model of computation called the *set-streaming* model (SS-model) (Section 2.2) and give a $\frac{1}{4}$ approximation algorithm for the *maximum coverage* (*MC-k*) problem in this model (Section 2.3). In addition to this, we show how a $\log_4 n + 1$ approximation algorithm for *set-cover* (*SC*) problem can be obtained in the SS-model (Section 2.5).
2. We introduce a new blog monitoring algorithm called *Blog-Watch* which on given a list of topics and a maximum cardinality k of the number of blogs a user is willing to monitor, returns at most k blogs covering those topics (Section 3). We present details of our *Blog-Watch* algorithm in Section 3.1.
3. We evaluate our algorithm both on synthetic and real blog datasets. Our preliminary experiments with $35,000$ blogs containing nearly 2 million posts demonstrate the strength of our method (Section 4.1).

2 Coverage Problems in the Set-Streaming Model

In this section, we review some basic definitions, introduce the SS-model and provide approximation algorithms for the *maximum coverage* and the *set cover* problem in this model.

2.1 Background

We begin by reviewing the definitions of *maximum coverage* and *set cover* problem.

DEFINITION 1. Maximum-k Coverage (*MC-k*) *Given a set of elements $\mathcal{E}$ with an associated weight $w : \mathcal{E} \rightarrow \mathbb{R}$, $|\mathcal{E}| = n$, a collection of subsets $\mathcal{S}$ of $\mathcal{E}$ and an integer k, the maximum-k coverage problem finds k sets from $\mathcal{S}$, that maximize the total weight of the covered elements in $\mathcal{E}$.*

If instead of specifying k, the number of sets, a budget B is specified and each set has an associated cost, then the objective becomes to find a collections of sets such that the total cost of these sets are $\leq B$ and the weight of the elements covered is maximized. This is known as *budgeted maximum coverage* (BMC) problem [15].

DEFINITION 2. S*et Cover (*SC*) Given a set of elements $\mathcal{E}$, $|\mathcal{E}| = n$, and a collection of subsets $\mathcal{S}$ of $\mathcal{E}$, the set cover problem finds a minimum number of sets from $\mathcal{S}$ that cover all the elements in $\mathcal{E}$. An element $e \in \mathcal{E}$ is said to be covered by $S \in \mathcal{S}$, if $e \in S$.*

If each set $S \in \mathcal{S}$ has an associated weight $w : \mathcal{S} \rightarrow \mathbb{R}$ and the objective is to obtain a collection of sets from $\mathcal{S}$ which cover all the elements and have minimum total weight, it is known as *weighted set cover* (WSC) problem.

Both the problems of *MC-k* and *SC* are well-known to be NP-Hard. There are various approximation algorithms for *SC* [22, 14] achieving either one of the two competitive factors $O(\log n)$ or f, where f is the maximum number of sets in which a particular element occurs. A maximization (minimization) approximation algorithm is said to be α competitive iff the resultant solution S is always $\geq \frac{1}{\alpha}$ ($\leq \alpha$) times of the optimum. For *MC-k* problem a simple greedy algorithm achieves an approximation factor of $1 - (1 - \frac{1}{k})^k < (1 - \frac{1}{e})$. These bounds are the best possible assuming $P \neq NP$. The approximation bound for *BMC* and *WSC* are similar to *MC-k* and *SC* respectively. However all these algorithms are essentially offline and need to know the elements and the sets before running the algorithm.

2.2 The Set-Streaming Model & Related Work

Alon et al.[20] introduced the online set-cover problem. In their work, the collection of sets is known a priori and the elements are arriving online. Buchbinder and Naor [21] discuss a primal dual framework for several packing (dual of covering) and covering algorithms and improve the approximation bound of [20]. An online algorithm for maximum coverage problem, when elements have unit weight is discussed in [2]. However all these algorithms have an essential feature that *the sets are known in advance*. Only the elements are arriving online and the algorithms do not know in advance which subset of elements will arrive.

We propose an alternate model for online coverage problems, which we refer to as the *set-streaming model (SS)*. In this model, the elements in $\mathcal{E}$ are known, but the sets are not known in advance. The sets arrive in a streaming fashion. A solution, consistent with the stream of sets seen so far, has to be computed on-the-fly. Only a subset of the sets seen so far should be kept in memory. An algorithm under this model will be said to have optimum space if the subsets of the sets kept in the memory at any instance of time is the desired solution. A set that is dropped at any point of time cannot be retrieved at a later time. The model ideally allows only a single pass over the set stream. Like some other models of data streaming computation, if the SS-model allows multiple passes, we call it multi-pass SS-model.

There has been previous work on online algorithm for pipelined set-cover problem [19], where the sets are coming online. However in the pipelined set-cover problem, the weight of a set depends upon the order in which the set is presented and cannot be arbitrary. None of the previous online algorithms for *MC-k* and *SC* can be applied for these problems in the SS-model (both single pass and multi-pass).

2.3 Online Maximum-K Coverage

We begin by presenting a $\frac{1}{4}$ competitive algorithm for the *MC-k* problem in the SS-model. We assume that a ground set of elements $\mathcal{E}$ are provided. The objective is to compute a solution close to the optimum by storing only a few sets from the set-stream. In the SS-model, once a set is discarded it cannot be recovered. Our algorithm at any instance works only with $k + 1$ sets and selects k of them to store. Since any optimum algorithm will store at least k sets, our algorithm has optimum space complexity in terms of the number of sets.

Online Maximum-k Coverage Algorithm: Let us denote the i-th set in the stream by S_i and let $T_{curr} = \{T_1, T_2, \ldots, T_k\}$ denote at any point the k sets in the current solution. Initially when no more than k sets are seen, T_{curr} contains all those sets. Therefore assuming a set arrives at each time unit, at $time = k$, $T_{curr} = \{S_1, S_2, \ldots, S_k\}$. The algorithm always maintains T_{curr}. At any time the new set that arrives is compared with T_{curr}. Depending on whether the new set is selected in the solution or not, T_{curr} is adjusted or left unaltered. A set not explicitly saved in T_{curr} cannot be retrieved at a later time.

The algorithm *Maximum-k Coverage*(T_{curr}, S_t) describes how T_{curr} is updated at time t, when S_t arrives in the stream. The update procedure makes a greedy choice in a constrained manner. Instead of always favoring the new set if it is better by the greedy criteria, the algorithm selects it only when the gain is substantial.

The algorithm compares the newly arrived set S_t with the k sets of T_{curr} in turn. From each T_j a distinct set R_j is computed. From S_t, the set B is obtained. R_j represents all those elements of T_j which will be dropped from T_{curr} if T_j is replaced by S_t. So $e \in R_j$ iff $e \in T_j - (S_t \bigcup_{i \neq j} T_i)$, i.e e is only present in T_j and not in any of the other sets in T_{curr} or S_t. On the otherhand B represents all the new elements which will be included in T_{curr}, if S_t replaces some set in T_{curr}. Therefore $e \in B$ iff $e \in S_t - \bigcup_i T_i$, i.e e is not present in T_{curr} but S_t contains e.

If there exists a set $T_j \in T_{curr}$ such that the weight of the elements in R_j is less than twice the weight of the elements in B, then S_t is selected for inclusion. S_t replaces T_j if the weight of the elements in R_j, represented by $w(R_j)$, is minimum. Therefore S_t is chosen in T_{curr}, if the weight of the new elements that get covered by S_t is substantially more than the weight of the elements that get removed. Therefore the coverage of the elements are improved significantly when the algorithm decides to change the sets in its current solution.

To compute R_j for each set efficiently, we maintain a chained hash table for the elements covered. Each element has a counter associated with it. An element which gets covered for the first time has its counter set to 1. For each new set that covers it, its counter is incremented. While computing R_j, for each $e \in T_j$, its counter is decremented by 1. If the decremented value is 0 and $e \notin S_t$, then $e \in R_j$. All the elements in T_j with counter value of 1 are possible candidates for R_j. So only those elements are checked. B can be computed analogously, by checking for each element $e \in S_t$, whether its count is 0 in T_{curr}. So all the operations in the update procedure can be done in time linear in $size(T_{curr}) = \sum_{i=1}^{k} |T_i|$.

Algorithm 2.1: MAXIMUM-K COVERAGE(T_{curr}, S_t)

```
if $t == 0$
  then $T_{curr} = \emptyset, C = \emptyset$
       for $i \leftarrow 1$ to $k$
         do $T_i = \emptyset$
if $t > 0$ and $t \leq k$
  then $T_i \leftarrow S_i$
       for each $e \in T_i$
         do if $e \notin C$
              then $count[e] = 1$
                   $C \leftarrow C \cup \{e\}$
              else $count[e] \leftarrow count[e] + 1$
       $T_{curr} \leftarrow T_{curr} \cup T_i$
if $t > k$
  then $B \leftarrow 0$
       for each $e \in S_t$
         do if $e \notin C$
              then $B \leftarrow B + w(e)$
                   $C \leftarrow C \cup \{e\}$
                   $count[e] \leftarrow 1$
              else $count[e] \leftarrow count[e] + 1$
       for $j \leftarrow 1$ to $k$
         do $R_j \leftarrow 0$
            for each $e \in T_j$
              do if $count[e] == 1$
                   then $R_j \leftarrow R_j + w(e)$
       $R_{min} \leftarrow MINIMUM(R_1, R_2, \ldots, R_k)$
       if $B \geq 2 * R_{min}$
         then $T_{min} \leftarrow S_t$
              for each $e \in T_{min}$
                do $count[e] \leftarrow count[e] - 1$
                   if $count[e] == 0$
                     then Delete $count[e]$
                          $C \leftarrow C - \{e\}$
         else for each $e \in S_t$
                do $count[e] \leftarrow count[e] - 1$
                   if $count[e] == 0$
                     then Delete $count[e]$
                          $C \leftarrow C - \{e\}$
return ($T_{curr}$)
```

Note that it is crucial to consider S_t for inclusion, only when the gain in coverage is substantial (twice in our algorithm) over the set that it replaces. Following example shows that if a new set is allowed to replace a set in the current solution, whenever its gain is more than the set it replaces, no approximation guarantee better than $\Theta(\frac{1}{k})$ can be obtained.

Shortcoming of Simple Greedy Strategy. Let there be $\frac{k^2(k+1)}{2}$ elements in the universe. Partition these elements into k disjoint sets $S_1, S_2, .., S_k$, such that S_1 contains the first k elements, S_2 contains the next $2k$ elements,, S_i contains the next iS elements and so on. Let there be $k+1$ sets defined on each of S_i. Call them $T_{i,1}, T_{i,2}, , T_{i,k}, T_{i,k+1}$. $T_{i,1}$ contains the first i elements of S_i, $T_{i,2}$ contains the next i elements and so on. Set $T_{i,k+1}$ contains all the elements of S_i. The order in which the sets arrive is $T_{1,1}, T_{1,2}, , T_{1,k}, T_{1,k+1}, T_{2,1}, .., T_{2,k+1}, T_{3,1}, .., T_{k,k+1}$. When $T_{1,k+1}$ arrives, there are already k sets in the solution and replacing any $T_{1,i}$ with $T_{1,k+1}$ does not give any gain. So $T_{1,k+1}$ is discarded. Next since replacing $T_{1,1}$ by $T_{2,1}$ improves the coverage, $T_{1,1}$ is replaced by $T_{2,1}$. Similarly $T_{1,2}$ is replaced by $T_{2,2}$ and so on. When the set $T_{2,k+1}$ arrives the current coverage contains the sets $T_{2,1}, T_{2,2}, .., T_{2,k}$. Replacing any of the sets with $T_{2,k+1}$ does not give any gain, so $T_{2,k+1}$ is dropped. Continuing in this fashion greedy returns the final coverage as $T_{k,1}, T_{k,2}, .., T_{k,k}$, covering k^2 elements. However $OPT = \{T_{1,k+1}, T_{2,k+1}, .., T_{k,k+1}\}$. Thus OPT covers $\frac{k^2(k+1)}{2}$ elements, giving an approximation bound of $\frac{2}{(k+1)}$. In our algorithm $T_{3,1}, T_{3,2}$ will not remove $T_{2,1}, T_{2,2}$ etc. Therefore the set $T_{3,k+1}$ will be included and so on. In fact, for this example, our algorithm gives an approximation ratio much better than $\frac{1}{4}$.

Next, we prove that the Algorithm 3.1 achieves an approximation factor of $\frac{1}{4}$ for the maximum coverage problem in the SS-model.

2.4 Analysis of *MC-k* in the SS Model

Let the optimum solution for *MC-k* be $\{O_1, O_2, \ldots, O_k\}$. W.l.o.g we can assume each of these O_i's, $i = 1, 2, \ldots, k$ are disjoint. Otherwise let O_i denote those elements not included in $O_1, O_2, .., O_{i-1}$. We charge the cost of the optimum, denoted by OPT, to the solution obtained by the algorithm 2.1. If by this charging scheme each element covered by the final T_{curr} is charged at most $\alpha \geq 1$ times of their weight, then $\alpha * cost(T_{curr}) \geq OPT$, or we have an $\frac{1}{\alpha}$ approximation algorithm for *MC-k*. In the same way we can define charge on a set. The weight of a set s is the total weight of the elements in s. If by the charging scheme each set in the final T_{curr} is charged at most $\beta \geq 1$ times of their weight and each element is charged at most $\alpha \geq 1$ times of their weight, then we get a $\frac{1}{\alpha+\beta}$ approximation algorithm for *MC-k*.

Charging of elements and sets can be divided into several subcases.

Case 1: If O_i is included in T_{curr} at some point of time, let $T_j = O_i$ at that time. Then for each element $e \in O_i$, charge the element $e \in T_{curr}$ by $w(e)$.

Case 2: Else O_i was not included when it arrived in the stream. Let again the current cover at the time when O_i arrived be T_{curr}.

Case 2.a: For any $e \in O_i \cap T_{curr}$, charge the element by $w(e)$. That is if the element $e \in O_i$ is covered by the current solution, charge it by its weight.

We refer to the charge acquired by Case 1 and Case 2.a as *"element charge (EC)"*.

Case 2.b: Now if O_i had been included, the new elements which would have got covered are $B_i = O_i - T_{curr}$. Consider the elements which would have got removed in the process. If O_i had replaced T_j, then the elements removed would be R_j. We charge each $e \in R_j$, for $j = 1, 2, \dots, k$, by $\frac{w(B_i)w(e)}{w(\cup_{j=1}^{k} T_j)}$.

We refer to the charge acquired in Case 2.b by *"set-charge(SC)"*.

LEMMA 2.1. *Each set T_j, for $j = 1, 2, \dots, k$, gets a charge of at most $\frac{2w(R_j)}{k}$, by Case 2.b, for each set O_i in optimum which is never included in T_{curr} by the algorithm (2.1).*

Proof: We know $R_j = T_j - \{O_i \cup (\cup_{l=1, l\neq j}^{k} T_l)\}$ for $j = 1, 2, \dots, k$. Therefore $R_j \subseteq T_j$ and $R_j \cap R_{j'} = \phi$, if $j \neq j'$. Hence $w(\cup_{j=1}^{k} R_j) = \sum_{j=1}^{k} w(R_j)$. Since O_i was not selected by the algorithm, it must hold:

$$w(B_i) \leq 2w(R_1), w(B_i) \leq 2w(R_2), \dots w(B_i) \leq 2w(R_k)$$

Or, we have by summing both sides of the inequality

$$kw(B_i) \leq 2\sum_{j=1}^{k} w(R_j) = 2w(\cup_{j=1}^{k} R_j)$$

So we have, $w(B_i) \leq \frac{2}{k} w(\cup_{j=1}^{k} R_j)$.
Therefore $\forall e \in \cup_{j=1}^{k} R_j$,

$$\text{charge on } e = \frac{w(B_i)w(e)}{w(\cup_{j=1}^{k} R_j)} \leq \frac{2w(e)}{k}$$

Hence we get,

$$\begin{aligned} \text{charge on } T_j &= \sum_{e \in R_j} \text{charge on } e \\ &\leq \sum_{e \in R_j} \frac{2w(e)}{k} = \frac{2w(R_j)}{k} \leq \frac{2w(T_j)}{k} \end{aligned}$$

Now we show how the accumulated charge is distributed among the elements and the sets of the final solution.
Transfer of Set-Charge:

If S replaces T_j in some iteration of the algorithm (2.1), then $w(B_S) \geq 2w(R_j)$. If any element $e \in T_j - R_j$ is now covered by T'_j, $j' > j$, then transfer the set-charge on T_j for e to set-charge on T'_j for e. Since *set-charge* on any element e by any O_i is $\leq \frac{2w(e)}{k}$. The Lemma 2.1 thus holds for T'_j.

The remaining *set-charge* on T_j is $\leq \frac{2w(R_j)}{k}$. Transfer the remaining *set-charge* on T_j as *set-charge* on S. Since $w(B_S) \geq w(R_j)$, and S covers only the elements in B_S, the Lemma 2.1 holds for S as well.

Now since there are k sets in the optimum cover, a set can be charged at most k times by Case 2.b. Hence we have the following lemma:

LEMMA 2.2. *The total set-charge in the final cover, T_{final}, returned by the Algorithm 2.1, is at most $2w(T_{final})$.*

Transfer of Element-Charge:

$\forall e \in T_j - R_j$, e continues to remain covered after replacement of T_j by S. Now let $R_1, R_2, .., R_l$ be all the elements removed at each iteration and $B_1, B_2, ..., B_l$ be all the elements that get newly covered in those iterations respectively. Let initially all the elements covered by the first k sets be B_0. Let the final elements which are covered be denoted by F, where $w(F) = \sum_{i>=0} w(B_i) - \sum_{i>=1} w(R_i)$. We have,

$$2w(R_1) < w(B_1), 2w(R_2) < w(B_2), \dots, 2w(R_l) < w(B_l)$$

Or, $2\sum_{i>=1} w(R_i) < \sum_{i>=1} w(B_i)$. Hence we obtain:

$$\begin{aligned} 2\sum_{i>=1} w(R_i) &\leq \sum_{i>=1} w(B_i) = \sum_{i>=0} w(B_i) - w(B_0) \\ &= w(F) + \sum_{i>=1} w(R_i) - w(B_0) \end{aligned}$$

Or, $\sum_{i>=1} w(R_i) < w(F) - w(B_0) < w(F)$. Now since the *element-charge* on the elements which are removed does not exceed their weight and $\sum_{i>=1} w(R_i) < w(F)$, the *element-charge* on the removed elements is at most $w(F)$. In addition to this, each element in F might have an *element-charge*. Therefore, noting that $w(T_{final}) = w(F)$, we have the following lemma:

LEMMA 2.3. *The total element-charge in the final cover, T_{final}, returned by the Algorithm 2.1, is at most $2w(T_{final})$.*

Finally we have the theorem,

THEOREM 2.1. *Algorithm 3.1 achieves an approximation factor of $\frac{1}{4}$ for the* MC-k *problem in the SS model.*

Proof: Total *set-charge* and *element-charge* is $\geq w(OPT)$. And total *set-charge* and *element-charge* $\leq 4w(F)$ from the Lemma 2.2 and 2.3. Hence $w(OPT) \leq 4w(F)$, or $w(F) \geq \frac{1}{4} w(OPT)$.

2.5 Online Set-Cover If multiple passes are allowed (precisely $\log_2 n(\log_4 n+1)$ passes), then the algorithm for *MC-k* can be used to give a $(\log_4 n+1)$ approximation algorithm for *SC*. The algorithm is an adaptation of *SCG* algorithm from [4].

Algorithm 2.2: SET COVER(g)

```
$lower \leftarrow 1$
$upper \leftarrow n$
while $upper > lower$
  do
    $L_{curr} \leftarrow \emptyset$
    $\hat{E} \leftarrow \mathcal{E}$
    $g \leftarrow \left\lfloor \frac{lower+upper}{2} \right\rfloor$
    for $i \leftarrow 1$ to $\log_4 n + 1$
      do
        $T_{curr} \leftarrow \emptyset$
        $T_{curr} \leftarrow$ Maximum-g Coverage($\hat{E}$)
        $L_{curr} \leftarrow L_{curr} \cup \{T_{curr}\}$
        $\hat{E} \leftarrow \mathcal{E} - E(L_{curr})$
    if $\hat{E} == \emptyset$
      then {$upper \leftarrow g$
      else {$lower \leftarrow g$
    $g \leftarrow \left\lfloor \frac{lower+upper}{2} \right\rfloor$
$g \leftarrow upper$
$L_{curr} \leftarrow \emptyset$
$\hat{E} \leftarrow \mathcal{E}$
for $i \leftarrow 1$ to $\log_4 n + 1$
  do
    $T_{curr} \leftarrow \emptyset$
    $T_{curr} \leftarrow$ Maximum-g Coverage($\hat{E}$)
    $L_{curr} \leftarrow L_{curr} \cup \{T_{curr}\}$
    $\hat{E} \leftarrow \mathcal{E} - E(L_{curr})$
return ($L_{curr}$)
```

Suppose we are able to guess l^*, the optimum number of sets in the cover. We will run the *MC-k* algorithm with $k = l^*$. By the approximation guarantee of Algorithm 2.1, the number of elements covered is at least $\frac{1}{4}$-th of the total elements. Iterating this algorithm for $\log_4 n + 1$ times results in a solution that covers all the elements. Since in each iteration at most l^* sets are included, we obtain a solution which contains at most $l^*(\log_4 n + 1)$ sets.

To guess l^*, remember that $1 \leq l^* \leq |\mathcal{S}|$. We do a binary search in this interval. If our guessed value $g < l^*$, then by iterating for $\log_4 n + 1$ times, the obtained sets will not be able to cover all the elements. If the guess $g \geq l^*$, all the elements get covered in $\log_4 n + 1$ iterations. Therefore if with our current guess, all the elements are not covered in $\log_4 n + 1$ iterations, we increase our guess (increase the lower boundary of binary search), else we lower the guess (decrease the upper boundary of binary search). Using $\Theta(\log_2 n)$ iterations we will be able to guess l^* within an interval of $l^* \pm 1$. Therefore all together we require $\log_2 n(\log_4 n + 1)$ iterations to obtain an approximation factor of $(\log_4 n + 1)$.

Algorithm 2.2 describes the *SC* algorithm in the SS-model in detail. In it, $lower$ and $upper$ represent the two boundaries of the guessed value of the optimum set cover. g represents the guessed value. Here, one function call to *Maximum-g Coverage($\hat{E}$)* runs Algorithm 2.1 for the entire stream of sets, considering only the elements in $\hat{E}$ and returns the cover in T_{curr}. L_{curr} includes all the sets in T_{curr} over the $\log_4 n+1$ iterations. $E(L_{curr})$ denotes the elements covered by the sets in L_{curr}. The first while loop detects the correct guess and after that, using that guess, the final cover is obtained.

Improving the number of passes is an important open problem. However it can be proved that in a single pass any set streaming algorithm that maintains a minimal cover cannot achieve an approximation guarantee better than the trivial bound of K, where K is the maximum size of any set. Here a cover is said to be minimal, if deleting any set from the cover, uncovers at least one element.

3 The Blog Watch Problem

This section introduces the blog watch problem in further detail. We first give some high-level desiderata for a blog watch solution and then give detailed description of the components of the *Blog-Watch* application and show how the *MC-k* algorithm in the SS-model can be used in this application.

Consider the following excerpts from two results returned in the top ten results of Google Blog Search when searched with topics "travel + music + sports + book" on May 2008:

Life is like a computer game.. We are all characters in our own game fighting to get next level.... I take inspiration from street/ skate/ **sports** *culture, subcultures and tribes but also* **music**, *animals, comics, computer games, science and science fiction, ...*

Do you watch any **sports** *on TV/which ones?: Nope. I'm not too into* **sports**. *Do you watch* **music** *videos?: Sometimes. Do you like watching I Love the 80's even if you weren't living in the 80's?*

When searching for good blogs to read on music and sports, it is unlikely that blogs containing these posts will be of interest. Among the other eight returned entries, only two were remotely related to the topics. These keyword-based searches are unlikely to retrieve appropriate results, since they apply AND semantics and try to retrieve blog-posts that match *all* of the topics they specify. A better solution is returning a collection blogs/posts which together covers the topics, without the requirement that each blog covers all topics of interest.

Here, we propose a *Blog-Watch* application for monitoring blogs that avoids these shortcomings. In this setting, users specify a set of topics in which they are interested, and the algorithm returns a set of *blogs* that the users should monitor for upcoming posts. Users specify the maximum number of blogs they are willing to monitor and the application figures out which blogs best satisfy the user's requirements.

The blogosphere is growing very quickly; new posts and new blogs are getting created every moment. In addition, as blogs evolve, the topics they cover change as well. A blog watch algorithm must be able to maintain the set of blogs for each user very efficiently, needs to scale with the growth of the blogosphere, and needs to be able to adapt to changing blog topics. The algorithm should update the list of recommended blogs for each user efficiently in memory as new blogs and posts are available, without requiring intensive disk I/O operations.

Therefore to summarize, the essential properties of *Blog-Watch* are:

- fast incremental update procedure for scalability,
- efficient use of memory space, and
- ability to adapt to evolving blog topics.

Our proposed *Blog-Watch* system requires a crawler, an input module, an estimator module and a monitoring module. The crawler handles the blog-stream and collects new posts and new blogs as they are available. Using proper indexing and the estimator module, for each new-coming blog or updated blog (because of new posts), a set of topics is determined. These sets represents the topics that each blog covers. The input module allows users to maintain their preferences and watches. The input module helps in establishing weight for each topic for every user. The monitoring module obtains this set-stream and perform the main algorithmic tasks. We describe the modules in detail next.

3.1 Blog Watch Modules In order to do a good job at monitoring blogs, *Blog-Watch* needs to estimate how well a blog covers a topic. It also needs to discover when a blog previously not selected becomes an attractive choice and must be added to the watch list, and it also needs to determine when a blog must be dropped. We describe a few estimators which are essential to maintain and will help to establish connection between *Blog-Watch* with the maximum coverage algorithm.

Estimators: The topic coverage estimator predicts, given a blog and a topic, whether the blog covers that topic well. Topic coverage estimators can be classified into two categories: *attribute-based estimators* and *collective estimators*. Attribute-based estimators are further categorized into frequency-based and concentration-based estimates[1].

Attribute-based estimators are local estimators for each blog. Users tag each post they write with a sequence of categories. The categories from each post in a blog are collected to form a category distribution vector (CDV) for each blog. Frequency-based estimate (FE) and concentration-based estimate (CE) uses CDV to determine whether a blog b cover a topic t.

- Frequency-based estimate: The frequency-based estimate counts the number of posts in b on the category t over a time window. Let $f(t, b)$ denote the count of posts on topic t in the blog b in the time window. Then if $f(t, b) \geq \theta_{FE}$, t is reported as a topic covered in b. Here θ_{FE} is a threshold.
- Concentration-based estimate: The concentration-based estimate depends on the density of posts on the topic t in b over a time window. If T_b denotes the total number of posts in blog b in the time window, then if $\frac{f_{t,b}}{T_b} \geq \theta_{CE}$, t is reported as a topic covered by the blog b. Here θ_{CE} is another threshold.

If the user wants to read only a few blogs but wants to cover many topics, she/he is likely looking for summarizer blogs. Whereas, if the number of blogs she/he is willing to monitor is large, it is likely the user prefers blogs that are more focussed. The concentration-based estimator can capture this property. If user is willing to read at most k blogs and is interested in l topics, then depending on the ratio of k to l, and choosing θ_{CE} appropriately, the estimator can determine whether a blog covers a certain topic.

The collective estimator considers the local goodness property of a blog b and also the blogs which have a link from b. Each blog b gets a local score $s_{b,t}$ on topic t, depending on the attribute-based estimator. The final score of the blog b on t is computed as follows:

$$score(b, t) = s_{b,t} + \sum_{v \text{ is a neighbour of } b} \alpha s_{v,t} \tag{3.1}$$

Here a blog v is a neighbor of blog b, if there is a link from blog b to v. α is a constant between 0 and 1, known as the decay factor. When $\alpha = 0$, the above is the simple attribute-based estimator.

In addition to topic coverage estimators, it is also possible to maintain an estimate of the growth of a blog. To determine how much a blog has grown, we can maintain two scores for each blog, $PrevScore$ and $CurrScore$. $PrevScore$ reflects the score when the blog was last considered by the *Blog-Watch* algorithm for possible inclusion in the recommended list. Whenever there is a new post, the current score of the blog, $CurrScore$ is updated. If the difference in $CurrScore$ and $PrevScore$ is substantial, the blog can be reconsidered for inclusion by the algorithm. In our current implementation, we have not used this estimator.

Indexing: *Blog-Watch* maintains a two-way index structure by the blog id and topic id. For each blog, its topic vector and list of neighbors is maintained for calculating the various scores. The structure is indexed by the blog id,

[1]Our current implementation of *Blog-Watch* uses variations of attribute-based estimates only.

which enables efficient update computation on each blog. The crawler, on receiving a post from a blog, extracts the topic tags and any reference to other blogs. It sends this information to the index manager. The index manager uses the index on blog id to extract the topic vector for that blog and update it with the topic tags of the current post. Neighbors of each blog are also kept updated similarly.

The index manager sends this updated entry to estimator module. The index manager has a second index structure maintained on topics. For each topic, there is a set of links to blog ids that has a high score on that topic. There is a third index based on userid which maintains the list of topics a user is interested.

3.2 Monitoring Module The monitoring module makes use of the information from the other modules and uses the *MC-k* algorithm in the SS model to retrieve a set of blogs for each user satisfying his/her interest.

We begin by defining some notations:

$\mathcal{B}$: The universe of weblogs.

$\mathcal{T}$: The universe of topics.

UI_u: The set of topics a user u is interested in, $UI_u \subseteq \mathcal{T}$.

$TW_u(t)$: The interest, or topic weight, the user u has for topic $t \in UI_u$.

$Categories(b)$: The topics that blog b covers; $Categories(b) \subseteq \mathcal{T}$.

$Benefit_u(B)$: The benefit of the blogs $b \in B$ for user u:

$$Benefit_u(B) = \sum_{t \in UI_u \cap (\cup_{b \in B} Categories(b))} TW(t).$$

The correspondence of these definitions with the parameters of *MC-k* and *SC* is as follows: We define an element $e \in \mathcal{E}$ for each topic $t \in UI_u$. Each blog b represents a set. The set corresponding to a blog b is $Categories(b)$. The entries in TW correspond to the weight function $w : \mathcal{E} \to \mathbb{R}$. Our *Blog-Watch* algorithm, given a budget of k by user u, UI_u, and TW_u will find k blogs from $\mathcal{B}$ that cover the maximum number of topics in UI_u. If all the topics in UI_u can not be covered by k blogs, then the topics with higher weight will be given more preference, that is the application will try to maximize $Benefit_u(B)$. This problem exactly maps to the *MC-k* problem. If the requirement is to cover all the topics by using minimum number of blogs, the problem becomes identical to the *SC* problem.

Here, though UI_us are known in advance, the blogs are discovered online by the crawler. Even though a blog had been seen earlier, the $Categories(b)$ may change and the blog may appear as a new one to the application. The SS-model becomes necessary in this scenario, by processing the blogs online in a space effective way.

Note that here in fact we are getting a stream of posts. However topics of all the posts contained within a blog determines the topic set of a blog. Therefore when a new post on a new topic is added, the corresponding topic set of the blog is updated. And this new topic set is now viewed as the incoming set in the SS-model.

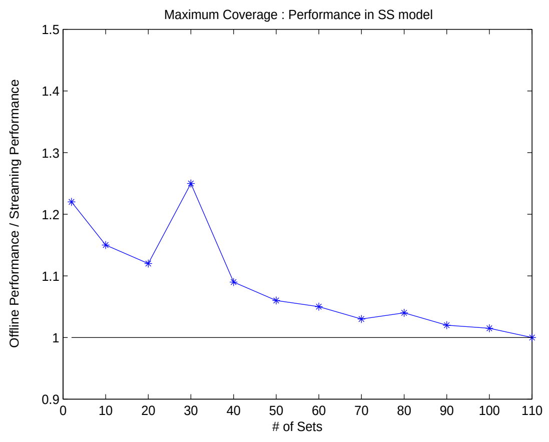
3.3 Topic and Interest Shift Note that the topics that a blog covers may undergo changes over time. Some old topics may fail to be covered and some new topics might emerge in a blog. The algorithm reflects these changes in its computation by maintaining the topic and growth estimator. A blog which was previously not a good choice for a user may become a viable choice as topics shift. This change will be reflected in the growth estimator and the algorithm will consider that blog for inclusion in its recommended list. If there is a topic shift in one of the blogs in the recommended list, then a lazy deletion technique can be employed. The blog is not removed, however the blog benefit is lowered. Therefore if a new better alternative comes, the old set gets dropped. If user's interest shifts, *Blog-Watch* uses the topic-based index structure for fast detection of good replacements.

4 Experimental Evaluation

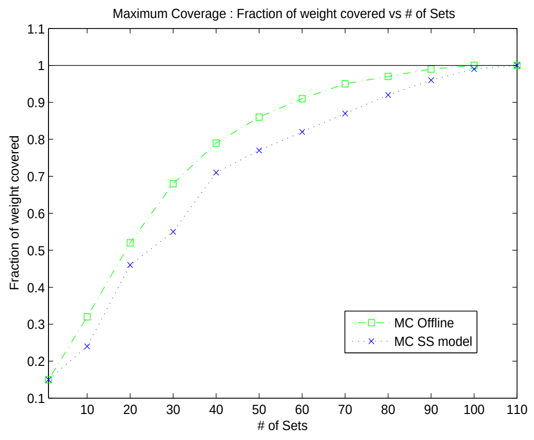
In this section, we present our experimental evaluation comparing the approximation bounds of *MC-k* and *SC* algorithms in the SS-model with their offline counterparts. We also demonstrate the performance of *Blog-Watch* on a sequence of topics, using attribute-based estimators, for a large collection of real-world blogs.

4.1 Results on Synthetic Data Set The first set of experiments were performed on a synthetically generated data set to compare the performance of the *MC-k* and the *SC* algorithms in the SS model. In the results reported here, we considered $|\mathcal{E}| = 1000$ with element weights following a normal distribution with mean 500 and variance 200. The number of sets was $10,000$ and their average length was 10 (the length varies from 1 to 20 following a normal distribution). The elements for each set were chosen from $\mathcal{E}$ uniformly at random. We also experimented with other distributions for generating the sets and the weight functions. The results were qualitatively similar in nature.

Figure 1(a) shows the competitive analysis of *MC-k* problem in the SS-model compared to the offline version. The y axis plots the fraction of the total weight of the elements covered by the the offline algorithm compared to the *MC-k* algorithm in the SS-model. A ratio closer to 1 represents a better solution. The result is shown for different

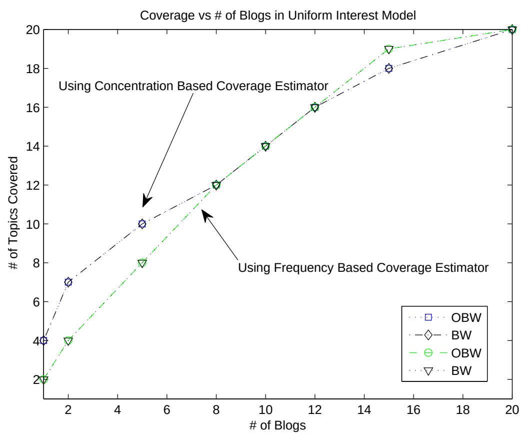


(a)

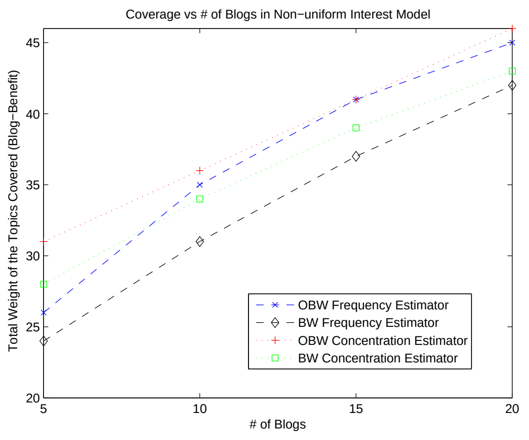


(b)

Figure 1: *MC-k* performance of offline vs. SS model. (a) Performance ratio (b) Fraction of elements covered.



(a)



(b)

Figure 2: Topic coverage performance of *BW* and *OBW* with *FE* and *CE*, using (a) uniform and (b) non-uniform topic weighting.

values of k. At $k = 30$, the worst performance ratio of 1.25 was observed. This implies the total weight of the elements covered in the SS-model is at least $(1-1/e)/1.25 = 0.504$ of the OPT. For $k \geq 40$, the ratio is lower than 1.1. Figure 1(b) shows how the fraction of elements covered grows with the number of sets. The rate of growth of *MC-k* algorithm in the SS-model is nearly same as the offline one. In the data used, the offline algorithm covers all the elements by 92 sets, whereas the algorithm in *SS*-model requires 101 sets, just 9 sets more. In this dataset, the offline greedy algorithm [14] for *SC* covers the elements in 108 sets, while the online SS-model uses 124 sets.

4.2 Results on a Real-world Blog Collection Our *Blog-Watch* experiments were run on a weblog collection crawled between July 4, 2005 to July 24, 2005 [2]. The data format included among other information, topic tags/categories for each post. Our experiments were restricted to only those posts which had non-empty English alphanumeric topic tags; this resulted in a dataset with $\sim 35,000$ blogs and about 2 million posts. We used the attribute-based coverage estimators over the topic tags of the posts to determine the topics each blog covers. Due to the short time frame of our collected data, each blog is considered for inclusion in the algorithm whenever there is a new post, ignoring the growth estimator. The growth estimator will be useful for long term running of *Blog-Watch*.

A set of 100 blogs were sampled uniformly from the dataset and a set of 20 topics were picked randomly from the collection of topic tags of these blog posts. Table 1 lists the 20 selected topics, which were used as the user's interest in our experiments. The performance of *Blog-Watch* is compared with the one which instead uses the offline *MC*-

[2]http://www.blogpulse.com/www2006-workshop

Topic	Weight1	Weight2	Topic	Weight1	Weight2
movie	3	1	life	1	1
health	3	3	art	1	1
health insurance	3	3	jewelry	1	1
internet	1	1	apartment	5	1
financial news	4	4	clothing	1	1
education	2	2	software	3	3
politics	2	2	technology	2	2
travel	4	1	blog	1	1
music	4	4	media	1	4
sports	1	1	podcast	1	1

Table 1: Distribution of preference weights on topics of interest.

k algorithm while monitoring. We use the notation *BW* and *OBW* to refer to *Blog-Watch* and its offline version. Under the uniform interest model (all the topics have same preference weight), the performance of *BW* in terms of topic coverage is nearly identical to *OBW*. Figure 2(a) shows that for uniform interest model, the number of topics covered by *BW* and *OBW* remains same on day 1, with varying k. The coverage on the successive days does not show any different trend as well.

Figure 2(a) also shows that, when k is low, using *CE* for coverage covers more topics than using *FE*. The difference diminishes as k increases and eventually the coverage by *FE* is more. This suggests that the appropriate choice of estimator may depend on the value of k.

When the preference weights on topics are not same and are taken from the Weight 1 column of Table 1, the *BW* performance differs from the *OBW*. The performance also depends on the type of coverage estimator used (*FE* or *CE*). Figure 2(b) illustrates the different behavior of *BW* and *OBW* for the two different coverage estimators. The total blog benefit is more for *OBW* and the gain grows as the number of blogs to be monitored increases. However it is always within a factor of 1.3 and 1.15 over the *BW* respectively for *FE* and *CE*. In these experiments we have set $\theta_{FE} = \lfloor 2 *$ Number of days $\rfloor$ and $\theta_{CE} = 1/2$ for the concentration-based estimates.

A sample set of blogs returned by our *Blog-Watch* application when *FE* is used and k is set to 10 is shown in Table 2. When *CE* is used, the results are quite similar, although `barbho we.typepad/lucky` was replaced by `agentlebossanova.net/rio`. At the time at which these blogs were crawled, `barbhowe.typepad/lucky` had a large number of posts on movies (now its topics have changed). *CE* preferred `agentlebossanova.net/rio`, because it covered 4 topics from the user's prescribed list, namely travel, music, sport and media.

Another trend that we observed in *Blog-Watch* is blog continuity. When $k = 5$, *CE* is used and the preference weights from column Weight 1 of Table 1 are chosen, the difference in the coverage of *BW* and *OBW* is observed for the topics "financial news" and "software". *OBW* covers the topic "financial news", whereas in *BW* "software" is covered. Note that the preference weight for financial news is less than twice the preference weight of software. Closer observation at the operation of *BW* reveals that the blog covering financial news was crawled after the blogs in software. Since the user's interest in financial news is not significantly more than software and because both of them cannot be covered in the limited budget of k without lowering the blog benefit, *BW* maintained its previous sets of blogs. Thus *BW* will maintain continuity, by not switching to a new blog unless there is a substantial gain in doing so. A similar trend is observed when *FE* is used in *BW*.

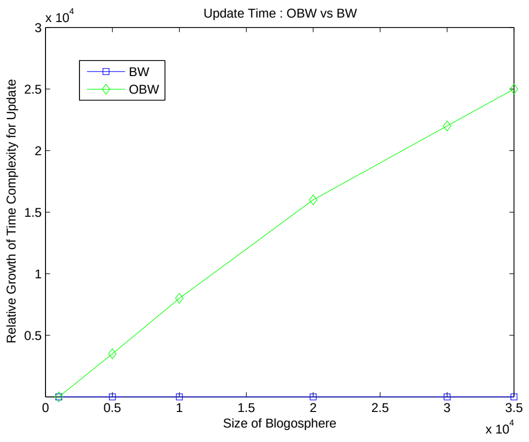


Figure 3: Relative growth of update time for *OBW* compared to *Blog-Watch*.

The topic 'apartment' has a high weight in the Weight 1 column of Table 1 and the blog 'allaboutguide/apartments' covers it, when $k = 5$ in *BW*. However when there is a slight modification in the weight distribution (column Weight 2 of Table 1), allaboutguide/apartments is replaced by a media blog at $k = 5$. The rest of the blogs chosen remained unaltered. This shows that *BW* does not alter the set of blogs substantially, when there is a little change in the preference weights. This is another desirable property of an application like this.

Figure 3 shows the relative growth of the time requirement of *OBW* compared to *BW*. The time to update the recommended list of blogs for each user in *BW* remains nearly constant when a new post/blog becomes available. However for *OBW*, the recommended list of blogs cannot be maintained incrementally. Moderate changes in the blogosphere will require *OBW* to rerun the monitoring module on the entire dataset. The update time thus grows proportionately with the number of blogs in this case.

586.typepad /hecklerspray	**Music**,**Movie**, Games,TV News, **Music Gossip**, Current Affairs
18minutegap	**Politics**
angelique.typepad	Apple, Cell Phones,Music,iTunes, iPod, Mobile phones,**Software**, Entertainment, **music**,**technology**
allaboutguide/apartments	**apartments**
awads.net/wp	**Technology**, Oracle, Java, XML, SQL
barbhowe. typepad/lucky	**Movie Reviews**
101-healthinsurance	**Health**, **Health Insurance**
101lab.net/blog	**Internet**
freddyblog.crossnet.se	**Blogosphere**
hotdeals.dealworld.info	Furniture,OfficeDepot,Electronics, **Software**,Printers,CompUSA, Coupon,Computer Hardware Computer Accessories,CircuitCity

Table 2: The blogs returned by *Blog-Watch* at $k = 10$ using frequency based coverage estimator.

4.3 Discussion The experimental results indicate the approximation capability of *MC* and *SC* in *SS*-model is even better than the worst case bound of $\frac{1}{4}$ and $O(\log n)$ predicted by the theoretical analysis. For blog monitoring purposes, the results verify that the streaming algorithm is capable of identifying al most the same set of blogs as an offline algorithm, at significantly less cost. The update time for the streaming algorithm remains constant, whereas for the offline algorithm it increases more than linearly with the growth of the blogosphere.

5 Related Work on Weblogs

In a recent work on weblogs [16], Leskovec et al. have addressed question such as: *which blogs one should read to avoid missing important stories ?* The experimental results indicate that, when the blogs have identical cost, the majority of the returned blogs are on politics. When weighted by the number of posts, the summarizers are returned. The processing is done offline and no topic oriented blog detection is addressed. Glance et al. [10] analyzed online discussion about various consumer products to make timely decision regarding brands, products and strategies in the corporate space. They approached topic classification in discussions with machine learning techniques and showed online winnow classifiers do a reasonable job in this scenario. Gruhl et al. [11] explored the dynamics of information diffusion in weblogs using topic level propagation and individual-to-individual propagation. Other works on blogs have focussed on modeling traffic characteristics and communication patterns [7], detecting communities using the topology of weblogs [3], modeling trust and influence, etc.

6 Conclusion

We have introduced a new streaming model of computation, the *Set Streaming* model and have considered the problems of *maximum coverage* and *set cover* problem in this model. We have designed approximation algorithms with guaranteed approximation bound for both the problems. As an application of these algorithms in the new model, we have designed the blog monitoring application *Blog-Watch.* We have shown how the various desirous properties of *Blog-Watch* can be captured by different parameters of these abstract algorithms. Incorporating all the features of *Blog-Watch* using link analysis and scalable topic detection algorithms is a future direction of work. Improving the approximation guarantee for the *maximum coverage* problem and obtaining a trade off between the approximation guarantee of set cover and the number of passes required are open theoretical questions.

Acknowledgments

This research has been supported in part by AFOSR Grant No. FA95500610405.

References

[1] A. W. Richa H. Rivano N. Stier Moses A. Ferreira, S. Perennes. Models, complexity and algorithms for the design of multifiber wdm networks. In *Telecommunication Systems 24*, pages 12–18, 2003.

[2] A. Fiat T. Leighton B. Awerbuch, Y. Azar. Making commitments in the face of uncertainty: how to pick a winner almost every time (extended abstract). In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 519–530, 1996.

[3] J. Bulters and M. de Rijke. Discovering weblog communities: A content- and topology-based approach. In *ICWSM 07: International Conference on Weblogs and Social Media*, pages 211–214, 2007.

[4] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In *APPROX-RANDOM:8th Intl. Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 72–83, 2004.

[5] G. Cormode and S. Muthukrishnan. “Space Efficient Mining of Multigraph Streams”. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 271–282, 2005.

[6] C. Demetrescu, I. Finocchi, and A. Ribichini. “Trading off space for passes in graph streaming problems”. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 714–723, 2006.

[7] A. Bestavros V. Almeida F. Duarte, B. Mattos and J. Almeida. Traffic characteristics and communication patterns in blogo-

sphere. In *ICWSM 07: International Conference on Weblogs and Social Media*, 2007.

[8] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, 2005.

[9] S. Ganguly and B. Saha. On estimating path aggregates over streaming graphs. In *ISAAC: International Symposium on Algorithms and Computation*, pages 163–172, 2006.

[10] N. Glance, M. Hurst, K. Nigam, M. Siegler, R. Stockton, and T. Tomokiyo. Deriving marketing intelligence from online discussion. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 419–428, 2005.

[11] D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. Information diffusion through blogspace. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 491–501, 2004.

[12] D. S. Hochbaum. Approximation algorithms for np hard problems. In *SIGACT News*, volume 28, pages 40–52, 1997.

[13] F. Naumann L. Raschid Y. Wu J. Bleiholder, S. Khuller. Query planning in the presence of overlapping sources. In *EDBT: 10th International Conference on Extending Database Technology*, pages 811–828, 2006.

[14] D. S. Johnson. Approximation algorithms for combinatorial problems. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49, 1973.

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

[16] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. Van-Briesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429, 2007.

[17] K. K. Saluja M. Franklin. Hypergraph coloring and reconfigured ram testing. *IEEE Trans. Comput.*, 43(6), 1994.

[18] A. McGregor. "Finding Graph Matchings in Data Streams". In *APPROX-RANDOM:9th Intl. Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 170–181, 2005.

[19] K. Munagala, S. Babu, R. Motwani, and J. Widom. The pipelined set cover problem. In *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK*, pages 83–98, 2005.

[20] Y. Azar N. Alon, B. Awerbuch. The online set cover problem. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 100–105, 2003.

[21] J. Naor N. Buchbinder. Online primal-dual algorithms for covering and packing problems. In *ESA 2005 : 13th annual European sympoisum*, pages 689–701, 2005.

[22] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2001.

[23] Shashua A. Zass, R. Probabilistic graph and hypergraph matching. In *Computer Vision and Pattern Recognition 2008. IEEE Conference on*, pages 1–8, 2008.