


Improved Algorithms for Maximum Coverage in Dynamic and Random Order Streams

Amit Chakrabarti  

Department of Computer Science, Dartmouth College, Hanover, NH, USA

Andrew McGregor  

Manning College of Information and Computer Sciences, University of Massachusetts, Amherst, MA, USA

Anthony Wirth  

School of Computing and Information Systems, The University of Melbourne, Australia
School of Computer Science, The University of Sydney, Australia

Abstract

The maximum coverage problem is to select k sets, from a collection of m sets, such that the cardinality of their union, in a universe of size n , is maximized. We consider $(1 - 1/e - \epsilon)$ -approximation algorithms for this NP-hard problem in three standard data stream models.

1. *Dynamic Model.* The stream consists of a sequence of sets being inserted and deleted. Our multi-pass algorithm uses $\epsilon^{-2}k \cdot \text{polylog}(n, m)$ space. The best previous result (Assadi and Khanna, SODA 2018) used $(n + \epsilon^{-4}k) \text{polylog}(n, m)$ space. While both algorithms use $O(\epsilon^{-1} \log m)$ passes, our analysis shows that, when $\epsilon \leq 1/\log \log m$, it is possible to reduce the number of passes by a $1/\log \log m$ factor without incurring additional space.
2. *Random Order Model.* In this model, there are no deletions, and the sets forming the instance are uniformly randomly permuted to form the input stream. We show that a single pass and $k \text{polylog}(n, m)$ space suffices for arbitrary small constant ϵ . The best previous result, by Warneke et al. (ESA 2023), used $k^2 \text{polylog}(n, m)$ space.
3. *Insert-Only Model.* Lastly, our results, along with numerous previous results, use a sub-sampling technique introduced by McGregor and Vu (ICDT 2017) to sparsify the input instance. We explain how this technique and others used in the paper can be implemented such that the amortized update time of our algorithm is polylogarithmic. This also implies an improvement of the state-of-the-art insert only algorithms in terms of the update time: $\text{polylog}(m, n)$ update time suffices, whereas the best previous result by Jaud et al. (SEA 2023) required update time that was linear in k .

2012 ACM Subject Classification Theory of computation \rightarrow Sketching and sampling

Keywords and phrases Data Stream Computation, Maximum Coverage, Submodular Maximization

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.40

Related Version *Full Version:* <https://arxiv.org/abs/2403.14087>

Funding *Amit Chakrabarti:* This work was supported in part by NSF under Award 2006589. This work was done in part while the author was visiting the Simons Institute for the Theory of Computing.

Andrew McGregor: This work was supported in part by NSF under Award 1934846. This work was done in part while the author was visiting the Simons Institute for the Theory of Computing.

Anthony Wirth: Wirth's visits to McGregor and Chakrabarti were funded by the Faculty of Engineering and Information Technology at The University of Melbourne.

Acknowledgements We thank the reviewers for helpful comments.



© Amit Chakrabarti, Andrew McGregor, and Anthony Wirth;
licensed under Creative Commons License CC-BY 4.0

32nd Annual European Symposium on Algorithms (ESA 2024).

Editors: Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman; Article No. 40; pp. 40:1–40:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The input to the *maximum coverage problem* is an integer k and a collection of m sets S_1, \dots, S_m , each a subset of the *universe* $[n] := \{1, \dots, n\}$. The goal is to find k sets whose union has maximum cardinality. This longstanding problem has several applications, including facility and sensor allocation [23], circuit layout and job scheduling [14], information retrieval [2], influence maximization in marketing strategy design [20], and content recommendation [29]. In terms of theoretical importance, it is perhaps the simplest example of a submodular maximization problem, a rich family of problems in machine learning [22]. A natural variant of the problem is the *set cover problem* and was one of Karp’s original 21 NP-complete problems [19]. A greedy approach that iteratively chooses the set with the greatest contribution yields asymptotically optimal approximation algorithms for set cover and maximum coverage [11]. Indeed, the greedy algorithm in practice typically outperforms its proven approximation factor for many problem instances [24]. Unfortunately, in many real-world applications, even the conceptually simple greedy approach is not practical. This can occur if the underlying data is too large to be stored in a single location and is therefore distributed across multiple machines; or if the data changes over time, e.g., sets are added or removed from the input collection.

Over the last decade, there has been a growing body of work that addresses the challenges of solving the maximum coverage problem (henceforth, MAXCOV), the related set cover problem (SETCOV), and the general problem of submodular optimization, in computational models such as the data stream model, that are relevant when the underlying data sets are massive [1, 3–5, 7–10, 13, 15, 16, 25, 27–29, 31, 33]. Assadi, Khanna and Li [4] have a comprehensive summary of results and discussion. The majority of work has focused on three specific models. In the *insert-only set streaming model* the data stream comprises m sets, in arbitrary order, each described once, contiguously. In the *random-order set streaming model*, we assume that these sets, each described once and contiguously, are ordered uniformly at random. This model captures a natural form of average-case analysis: the sets are chosen adversarially, but we can choose a random ordering to process them [3, 31]. In the *dynamic set streaming model*, each set may be added and deleted several times: the resulting MAXCOV instance involves only those sets that remain added at the end of the stream. In each case, the goal is to solve MAXCOV in space that is sublinear in the size of the input. We note that there has also been interest in an alternative “edge streaming” model [7, 15, 21], but this line of work is less relevant to our paper.

Selected Prior Results. To provide context for our results, we recap some state-of-the-art results from prior work. For the MAXCOV problem, in the insert-only setting, McGregor and Vu [27] gave a $(1 - 1/e - \epsilon)$ -approximation algorithm that uses $O(\epsilon^{-1})$ passes and $\tilde{O}(\epsilon^{-2}k)$ space; the algorithm can be implemented to achieve $\tilde{O}(k)$ update time [16]. In the dynamic setting, Assadi and Khanna [3] achieved the same approximation using $O(\epsilon^{-1} \log m)$ passes and $\tilde{O}(n + \epsilon^{-4}k)$ space. The threshold of $1 - 1/e$ is a barrier, thanks to known lower bounds [27].

Submodular maximization – specifically, maximizing a monotone submodular function under a cardinality constraint – generalizes MAXCOV; in the (insert-only) streaming setting [6, 12], we are given a stream of elements from a universe U (these elements correspond to sets S_i in MAXCOV) and have oracle access to the function $f: 2^U \rightarrow \mathbb{R}$ that is to be maximized. For this problem, [6] gave a $(1/2 - \epsilon)$ -approximation that uses one pass and $\tilde{O}(\epsilon^{-1}k)$ space, while [25] considered the random-order setting and gave a $(1 - 1/e - \epsilon)$ -approximation using

one pass and $\tilde{O}(\varepsilon^{-1}k)$ space. Importantly, these space bounds assume that remembering an element of U takes $\tilde{O}(1)$ space. Specializing these results to MAXCOV and properly accounting for the super-constant space required to store an element of U (i.e., a set S_i) blows up these space bounds. In particular, [31] achieves a space usage of $\tilde{O}(\varepsilon^{-2}k^2)$ for MAXCOV in random-order streams.

Our Results, Approach, and Closely Related Work. Our two primary results address the *space usage* of MAXCOV algorithms in two different streaming settings. We state these below, along with an outline of our approach, which refines and extends approaches from specific previous works. In the process, we also establish secondary results on *speeding up* existing algorithms and, in one case, developing a slightly more pass-efficient algorithm.

In the dynamic stream model, our main result is the following.

► **Result 1** (Algorithm for Dynamic Model). *There is an $O((1 + \varepsilon^{-1}/\log \log m) \log m)$ -pass algorithm for MAXCOV in the dynamic set streaming model that uses $\varepsilon^{-2} \cdot k \cdot \text{polylog}(n, m)$ space and returns a $(1 - 1/e - \varepsilon)$ -approximation.*

This result reappears as Theorem 5. It improves over the aforementioned Assadi–Khanna algorithm [3], which used $(n + \varepsilon^{-4}k) \text{polylog}(n, m)$ space. The high-level approach is similar and is based on the existing ℓ_0 -sampling primitive that allows uniform sampling amongst the sets that are inserted but not deleted. In each pass, we pick additional sets and maintain the union, C , of the selected sets. We pick the additional sets by sampling only among the sets that include a “meaningful” number of elements that are not currently in C . Unfortunately, when we simultaneously sample multiple sets in this way, once we start adding sampled sets to our solution, some of the sampled sets may no longer make a meaningful increment to $|C|$. However, by repeating the process over multiple passes, with a slowly decreasing threshold for what constitutes a meaningful improvement, we ensure that either we exhaust the collection of sets that could potentially be worth adding or we add enough sets and together these yield a good approximate solution. The main point of departure from [3] is in the way we handle the decreasing threshold. The largest threshold considered in [3] is about OPT/k ; this is sufficient for achieving a $1 - 1/e - \varepsilon$ approximation factor, but has the downside that there can be many overlapping sets that would contribute more than OPT/k new elements to C ; storing these sets during the sampling process is expensive. Instead, we consider thresholds decreasing from OPT but handling the sets that make a contribution between $\text{OPT}/2^i$ and $\text{OPT}/2^{i+1}$ for $i = 0, \dots, \log k$ separately but in parallel. We analyze a stochastic process – we call it the *cascading urns* process – to show how our new algorithm uses a similar number of passes as the algorithm of [3] but uses significantly less space. We present the details in Section 3. In fact, a more careful analysis of the process shows that it is possible to reduce the number of passes to $O(\log m + \varepsilon^{-1} \log m / \log \log m)$ in contrast to the $O(\varepsilon^{-1} \log m)$ passes used by previous algorithm. We believe that a similar pass saving can be achieved in Assadi and Khanna [3] but it requires an extra log factor in the space.

In the random order model, our main result is the following.

► **Result 2** (Algorithm for Random Order Model). *There is a single pass algorithm in the random order data stream model that uses $O_\varepsilon(k \cdot \text{polylog}(n, m))$ space and returns a $(1 - 1/e - \varepsilon)$ -approximation of MAXCOV in expectation.*

The above result is established in Section 4. Note that the dependence on ε in the result is exponential, hence the algorithm is less practical. The main significance of the result is removing a factor k in the space required by the best previous result [31], which used

$\varepsilon^{-2}k^2 \text{polylog}(n, m)$ space. Both their and our approach are based on modifying existing algorithms for the cardinality-constrained monotone submodular optimization problem. This is a more general problem than maximum coverage, but it is assumed in the general problem that the algorithm has oracle access to the function being optimized. For maximum coverage, we need to store enough information to be able to simulate this oracle. An $O_\varepsilon(k^2 \log m)$ -space algorithm for maximum coverage follows immediately for any $O_\varepsilon(k)$ space algorithm for monotone submodular optimization because the universe subsampling technique discussed in Section 2 allows us to focus on the case where the optimal solution covers $O(\varepsilon^{-2}k \log m)$ elements. To reduce the dependence on k to linear, rather than quadratic, we need a more careful adaption of an algorithm by Agrawal, Shadravan, and Stein [1]. We maintain a set of size $O(\varepsilon^{-2}k \log m)$ corresponding to the items covered by a collection of sets chosen for the solution so far, along with $O_\varepsilon(1)$ sets of size $O_\varepsilon(\log m)$. We present the details in Section 4.

Lastly, our results, along with numerous previous results, use the aforementioned universe sub-sampling technique introduced by McGregor and Vu [27] to sparsify the input instance. We explain how this technique and others used in the paper can be implemented such that the amortized *update time* of our algorithm is polylogarithmic. This also implies an improvement of the state-of-the-art insert only algorithms in terms of the update time: $\text{polylog}(m, n)$ update time suffices whereas the best previous result by Jaud et al. [16] required update time $k \cdot \text{polylog}(m, n)$. We present the details in Section 5.

2 Preliminaries

Without loss of generality, we assume each set is tagged with a unique ID in the range $[m^3n]$, for if not, we could generate suitable IDs via random hashing of the contents of the sets themselves.¹ In the dynamic setting, we will want to sample uniformly from the sets that are inserted but not deleted (sometimes we will put additional requirements on the relevant sets). To do this, we will use the standard technique of ℓ_0 sampling.

► **Theorem 1** (ℓ_0 sampling [17]). *There is a one-pass algorithm that processes a stream of tokens $\langle x_1, \Delta_1 \rangle, \langle x_2, \Delta_2 \rangle, \dots$, where each $x_i \in \{1, \dots, M\}$ and $\Delta_i \in \{-1, 1\}$, using $O(\log^2(M) \log(1/\delta))$ bits of space, that, with probability $1 - \delta$, returns an element chosen uniformly at random from the set $\{x \in [M] : \sum_{i:x_i=x} \Delta_i \neq 0\}$.*

Specifically, we will use the above technique when $M = m^3n$, with each x_i being the ID of a set and the corresponding Δ_i specifying whether the set is being inserted or deleted.

As mentioned in the introduction, the natural (offline) greedy approach achieves a $(1 - 1/e)$ -approximation for maximum coverage. In streaming settings, a “quantized” version of the algorithm achieves similar results, as summarized below.

► **Theorem 2** (Quantized Greedy Algorithm, e.g., [27]). *Consider the algorithm that processes a stream $\langle S_1, S_2, \dots \rangle$ of sets as follows. Starting with an empty collection, Y , it makes p passes: in the i^{th} pass, it adds to Y each encountered set that covers $\geq \tau_i$ uncovered elements. If, at some point, $|Y| = k$, then the algorithm stops and returns Y . If these threshold parameters, $\tau_1 > \dots > \tau_p$, satisfy*

$$\tau_1 \geq \text{OPT}/k, \quad \tau_p < \text{OPT}/(4ek), \quad \text{and} \quad \tau_i/\tau_{i+1} \leq 1 + \varepsilon \text{ for all } i \in [p-1],$$

then the algorithm achieves a $(1 - 1/e - \varepsilon)$ -approximation for the maximum coverage problem.

¹ E.g., hash each set $S \in [n]$, to $f_S(r)$, where $f_S(X) = \prod_{u \in S} (X - u)$ is a polynomial over a prime field \mathbb{F}_p of cardinality $\Theta(m^3n)$ and r is uniformly random in \mathbb{F}_p . By elementary analysis, the map $S \mapsto f_S(r)$ is non-injective with probability at most $\binom{m}{2}n/p \leq 1/m$.

Both of our main algorithmic results appeal to the following technique that, given a guess², v , of the value OPT , transforms the MAXCOV instance into one in which the optimum solution covers $O(\varepsilon^{-2}k \log m)$ elements. Note that this immediately implies that every set in the input has cardinality $O(\varepsilon^{-2}k \log m)$. In what follows, we will tacitly assume that the given instance is filtered through this technique before being fed into the algorithms we design. In particular, we will appeal to the following theorem [16, 27].

► **Theorem 3** (Universe Subsampling). *Let the function $h : [n] \rightarrow \{0, 1\}$ be drawn uniformly at random from a family of $O(k \log m)$ -wise independent hash functions, where*

$$p := \Pr[h(e) = 1] = \lambda/v, \quad \text{for } \lambda = 10\varepsilon^{-2}k,$$

and v satisfies $\text{OPT}/2 \leq v \leq \text{OPT}$. An h -sparsification of an instance \mathcal{I} of MAXCOV is formed by replacing every set S in \mathcal{I} by $\{e \in S : h(e) = 1\}$. With high probability, a) any α -approximation for the h -sparsification of \mathcal{I} yields an $(\alpha - \varepsilon)$ -approximation for \mathcal{I} and b) the optimum solution for the h -sparsification of \mathcal{I} covers $O(\varepsilon^{-2}k \log m)$ elements.

In Section 5, we discuss how to implement the universe subsampling such that the update time of the resulting algorithm is logarithmic for each element of a set in the stream.

3 Dynamic Streams

Our main algorithm for dynamic streams is based on the following variant (Algorithm 1) of the quantized greedy algorithm. It requires an estimate v satisfying $\text{OPT}/2 \leq v < \text{OPT}$. To describe it compactly, we set up some notation. Let \mathcal{S} denote the collection of sets yielded by the input stream. For a given $C \subset [n]$, define the subcollections

$$\begin{aligned} \mathcal{F}_i^C &= \{S \in \mathcal{S} : \theta_i \leq |S \setminus C| < \theta_{i-1}\}, \quad \text{for } i \in [\ell], \\ \mathcal{G}_i^C &= \{S \in \mathcal{S} : \tau_i \leq |S \setminus C| < \tau_{i-1}\}, \quad \text{for } i \in [1 + \lceil \log_{1+\varepsilon}(16e) \rceil], \end{aligned}$$

where,

$$\ell = \lceil \log k \rceil, \quad \theta_i := \frac{2v}{2^i}, \quad \text{and} \quad \tau_i := \frac{2v/2^\ell}{(1 + \varepsilon)^{i-1}}.$$

In the pseudocode below, the computed solution, Y , is a set of IDs and the set C is maintained to be the subset of the universe covered by the sets represented in Y . The macro³ $\text{GROW-SOLUTION}(Y, \mathcal{S}, C, \theta)$ implements the following logic: if $|S \setminus C| \geq \theta$, then update $Y \leftarrow Y \cup \{\text{ID}(S)\}$ (i.e., add the set S to the solution) and $C \leftarrow C \cup S$; if this makes $|Y| = k$, then stop and return Y .

The most involved part of the analysis of Algorithm 1 is the proof of the following lemma, which we shall defer slightly.

► **Lemma 4.** *With high probability, the while loop at Line 2 makes $O(\log m)$ iterations and each invocation of the while loop at Line 7 makes $O(\log m / \log \log m)$ iterations.*

► **Theorem 5.** *There is a $(1 - 1/e - \varepsilon)$ -approximation algorithm for max coverage in the dynamic stream model that uses $\tilde{O}(k/\varepsilon^2)$ space and $O((1 + \varepsilon^{-1} / \log \log m) \log m)$ passes.*

² We run the final algorithms with guesses $v = 1, 2, 4, 8, \dots, 2^{\log n}$ and use the fact that one of these guesses is within a factor 2 of OPT .

³ Being a macro, GROW-SOLUTION can cause the invoking algorithm to return.

■ **Algorithm 1** Quantized greedy adapted for dynamic set streams.

```

1: start with an empty solution ( $Y \leftarrow \emptyset$ ) and let  $C \leftarrow \emptyset$ 
2: while  $\mathcal{F}_1^C \cup \dots \cup \mathcal{F}_\ell^C \neq \emptyset$  do
3:   for  $i \in [\ell]$  do sample  $2^i$  sets from  $\mathcal{F}_i^C$  with replacement
4:   for  $i \in [\ell]$  and each set  $S$  sampled from  $\mathcal{F}_i^C$  do
5:      $\text{GROW-SOLUTION}(Y, S, C, \theta_i)$ 
6:   for  $i \leftarrow 1$  to  $1 + \lceil \log_{1+\varepsilon}(16e) \rceil$  do
7:     while  $\mathcal{G}_i^C \neq \emptyset$  do
8:       sample  $k$  sets from  $\mathcal{G}_i^C$  with replacement
9:       for each sampled set  $S$  do  $\text{GROW-SOLUTION}(Y, S, C, \tau_i)$ 

```

Proof. The idea is to implement Algorithm 1 in a small number of space-efficient streaming passes. To sample from \mathcal{F}_i^C and check the non-emptiness condition of the while loop at Line 2, we use an ℓ_0 -sampling pass on the IDs of the sets where $|S \setminus C|$ is in the relevant range. Using an additional pass, we store $\{S \setminus C\}$ for all the sampled sets, so that we can implement the logic of GROW-SOLUTION. These sets contain at most

$$\sum_{i=1}^{\ell} 2^i \cdot 2v/2^{i-1} = 4v\ell = O(\varepsilon^{-2}k \log m \log k)$$

elements. Similarly we can sample sets from each \mathcal{G}_i^C in $O(\varepsilon^{-2}k \log k)$ space.

Since each iteration of each while loop corresponds to two streaming passes, Lemma 4 shows that w.h.p. the algorithm uses at most the claimed number of passes.

The approximation guarantee follows from Theorem 2, since $\tau_1 = 2v/2^\ell > \text{OPT}/k$ and

$$\tau_{1+\lceil \log_{1+\varepsilon}(16e) \rceil} = \frac{2v/2^\ell}{(1+\varepsilon)^{\lceil \log_{1+\varepsilon}(16e) \rceil}} < \frac{4\text{OPT}}{16ek} = \frac{\text{OPT}}{4ek}. \quad \blacktriangleleft$$

It remains to prove Lemma 4. We need to understand how the collections \mathcal{F}_i^C and \mathcal{G}_i^C evolve as we grow our solution Y , thus changing C . To this end, we introduce two stochastic *urn processes*: the first (and simpler) process models the evolution of the collection \mathcal{G}_i^C , for a particular i ; the second process models the evolution of the ensemble $\{\mathcal{F}_i^C\}$. How exactly the urn processes model these evolutions is explained in Section 3.3.

3.1 The Single Urn Process and its Analysis

An urn contains m balls, each initially *gold*. Balls are drawn from the urn in phases, where a phase consists of drawing d balls uniformly at random, with replacement. When a ball is drawn, if it is gold, then we earn one point and some arbitrary subset of the balls in the urn turn into *lead*, including the ball that was just drawn (and put back). At the end of the phase, all lead balls are removed from the urn and the next phase begins. The process ends when either d points have been earned or the urn is empty.

We will prove the following result about this process.

► **Theorem 6.** *With probability $\geq \frac{9}{10}$, the urn process ends within $O(\log m / \log \log m)$ phases.*

The intuition behind the result is as follows. The observation is that if the fraction of balls in the urn remains above d'/d then each draw is gold with probability at least d'/d and it would be reasonable to expect at least $d'/d \times d = d'$ of the draws in a phase to be gold. Define $m(i)$

to be number of gold balls left in the urn after i phases and let $d(i)$ be the number of gold balls observed in the i^{th} phase. For the sake of intuition (the formal proof follows shortly) suppose the observation holds for all rounds, i.e., $m(i)/m(i-1) \leq d(i)/d$ for all i ; then

$$\frac{m(i)}{m} = \prod_{j=1}^i \frac{m(j)}{m(j-1)} \leq \prod_{j=1}^i \frac{d(j)}{d} \leq \left(\frac{\sum_{j=1}^i d(j)/d}{i} \right)^i \leq 1/i^i,$$

by the AM-GM inequality and the fact that every drawn gold ball is turned into lead, so only contributes to at most one term $d(j)$, hence $\sum_j d(j)/d \leq 1$. Consequently, i can be at most $O(\log m / \log \log m)$ before $m(i)$ becomes zero.

Our result is stronger than analogous results in previous work in two specific ways. Previous analyses essentially need (i) a factor $O(\log m)$ in the number of draws taken in each phase and (ii) only establish that $O(\log m)$ phases suffice, rather than $O(\log m / \log \log m)$ phases. The improvements in our approach stem from avoiding the need to guarantee that each phase makes progress with high probability and considering both the progress made towards observing d gold balls and the progress made in terms of turning balls into lead.

Proof of Theorem 6. Let $m(j)$ be the number of balls in the urn after the j^{th} phase ends and all lead balls are removed. Also, let $m(0) = m$. Put $\gamma = \log \log m / \log m$ and assume $d \geq 12/\gamma$, otherwise the result follows trivially, since each phase earns at least one point. The j^{th} phase is deemed *successful* if, during it, either $\gamma d/2$ points are earned or the fraction of gold balls drops below γ (causing $m(j)/m(j-1) < \gamma$). In Lemmas 7 and 8, we will establish that each phase is successful with probability at least $1/2$, even when conditioned on the outcomes of previous phases. Thus, by a Chernoff bound, after $10/\gamma$ phases the probability that there were at least $4/\gamma$ successful phases is at least $1 - \exp(-O(1/\gamma))$. If $4/\gamma$ phases are successful, either we have earned $(2/\gamma)(\gamma d/2)$ points or the number of balls in the urn has reduced from $m(0) = m$ to at most $(\gamma)^{2/\gamma} m$. By definition, this number is less than 1, implying that the urn is empty. ◀

We turn to lower bounding the success probability of a phase as claimed in the above proof. Fix a particular phase and a particular realization of all previous phases of the stochastic process. Let G_i be the event that the i^{th} draw of this phase reveals a gold ball (and thus earns a point); let L_i be the event that after $i-1$ draws, the fraction of gold balls in the urn is less than γ . Define indicator variables $X_i = \mathbb{1}_{G_i \cup L_i}$ and set $X = \sum_{i=1}^d X_i$.

$$\mathbb{E}[X_i] = \Pr[L_i] + \Pr[G_i \cap \bar{L}_i] = \Pr[L_i] + \Pr[\bar{L}_i] \Pr[G_i \mid \bar{L}_i] \geq \Pr[G_i \mid \bar{L}_i] \geq \gamma. \quad (1)$$

► **Lemma 7.** *If $X \geq \gamma d/2$, then the phase is successful.*

Proof. If $X \geq \gamma d/2$, either some event L_i occurs or we collect $\gamma d/2$ gold balls (and points). In the latter case, the phase is successful by definition. In the former case, the fraction of gold balls drops below γ during the phase. Since lead never turns into gold, this fraction remains below γ . Upon removing lead balls at the end of the phase, the number of balls in the urn drops to at most γ times the number at the start of the phase, i.e., the phase is successful. ◀

Finally, we show that X stochastically dominates a binomial random variable, which lower-bounds $\Pr[X \geq \gamma d/2]$.

► **Lemma 8.** $\Pr[X \geq \gamma d/2] \geq 1 - \exp(-\gamma d/12) \geq 1/2$, where the latter inequality uses our assumption that $d \geq 12/\gamma$.

Proof. Let Y_1, \dots, Y_d be independent draws from the Bernoulli distribution $\text{Bern}(\gamma)$. Put $X^{\leq j} = \sum_{i=1}^j X_i$ and $Y^{\leq j} = \sum_{i=1}^j Y_i$. We first show by induction on j that $X^{\leq j}$ stochastically dominates $Y^{\leq j}$ for all j . Inequality (1) with $i = 1$ establishes the base case of $j = 1$. Assuming the result for j , for an arbitrary integer, z , we have

$$\begin{aligned} \Pr[X^{\leq j+1} \geq z] &= \Pr[X^{\leq j} \geq z] + \Pr[X^{\leq j} = z-1] \Pr[X_{j+1} = 1 \mid X^{\leq j} = z-1] \\ &\geq \Pr[X^{\leq j} \geq z] + \Pr[X^{\leq j} = z-1] \gamma \\ &= \Pr[X^{\leq j} \geq z](1-\gamma) + \Pr[X^{\leq j} \geq z-1] \gamma \\ &\geq \Pr[Y^{\leq j} \geq z](1-\gamma) + \Pr[Y^{\leq j} \geq z-1] \gamma = \Pr[Y^{\leq j+1} \geq z]. \end{aligned}$$

Thus, $\Pr[X \geq \gamma d/2] \geq \Pr[Y \geq \gamma d/2]$. The lemma now follows by applying a Chernoff bound to the binomial random variable $Y \sim \text{Bin}(d, \gamma)$. ◀

3.2 Cascading Urns

To prove the first part of Lemma 4, we will also need to analyze a more complex version of the above process; we call it the *cascading urn process*. Now we have t urns, numbered 1 through t , with the r^{th} urn starting out with m_r balls; let $m = \sum_{r \in [t]} m_r$. Drawing a gold ball from urn r is worth $d/2^r$ points. Initially, all balls are gold. The process works as follows.

- Balls are drawn in phases: a phase consists of drawing 2^r balls from urn r , in order of increasing r .
- When a ball is drawn from urn r , if it is gold, the following things happen: (i) the ball turns into lead and is returned to urn r ; (ii) some arbitrary subset of the balls in the urns (all urns, not just urn r) turn into lead; and (iii) we earn $d/2^r$ points.
- At the end of each phase, each lead ball in an urn is removed from its current urn and either destroyed or transformed back into gold and placed in a higher-numbered urn. We then start the next phase with all balls being gold again.

The process ends when either d points have been earned or all urns have become empty.

► **Theorem 9.** *With probability $\geq \frac{9}{10}$, the cascading urn process ends in $O(t + \log m)$ phases.*

Proof. Fix a particular phase, p . For each $r \in [t]$, the analysis of one phase of the single-urn process, applied to urn j , establishes that, during this phase, with probability at least

$$\Pr \left[\text{Bin} \left(2^r, \frac{1}{2} \right) \geq \frac{2^r}{4} \right] \geq 1 - \exp \left(\frac{-2^{r-1}}{12} \right) \geq 1 - e^{-\frac{1}{12}} > \frac{1}{13} =: 8\xi,$$

either at least $d/4$ points are earned from this urn (call this event $G_{r,p}$) or the fraction of gold balls in this urn drops below $1/2$ (call this event $L_{r,p}$). Let $Z_{r,p} = \mathbb{1}_{G_{r,p} \cup L_{r,p}}$ and observe that $\mathbb{E}[Z_{r,p}] > 8\xi$.

Let $m_{r,p}$ be the number of balls in urn r at the start of the phase p (recall that these are all gold balls) and consider the quantity $Q_p := \sum_{r \in [t]} 2^{t-r} m_{r,p}$. Note that if at least half of the $m_{r,p}$ balls in urn r turn into lead, then Q_p decreases by at least $2^{t-r} m_{r,p}/4$ because the contribution of these ball decreases by at least a factor 2. If $\bigcup_r G_{r,p}$ does not occur, then

$$Q_{p+1} \leq \sum_{r \in [t]} 2^{t-r} (Z_{r,p} \cdot 3m_{r,p}/4 + (1 - Z_{r,p})m_{r,p}) = \sum_{r \in [t]} 2^{t-r} (1 - Z_{r,p}/4)m_{r,p}.$$

Using the lower bound on $\mathbb{E}[Z_{r,p}]$, we get

$$\mathbb{E}[Q_{p+1}] \leq \sum_{r \in [t]} 2^{t-r} (1 - 2\xi)m_{r,p} = (1 - 2\xi)Q_p.$$

Applying a Markov bound gives

$$\Pr[Q_{p+1} \leq (1 - \xi)Q_p] \geq 1 - \mathbb{E}[Q_{p+1}] / ((1 - \xi)Q_p) \geq \xi / (1 - \xi).$$

Hence, with probability at least $\xi / (1 - \xi)$, either $\bigcup_r G_{r,p}$ occurs (in which case we collect $d/4$ points) or the fraction of balls that remain gold until the end of the phase is at most $1 - \xi$. Note that $Q_0 \leq 2^{t-1}m$ and if $Q_p < 1$ for some p , then every urn must be empty after phase p .

With probability at least $\xi / (1 - \xi)$, in each phase, we either earn $d/4$ points or Q_p reduces by a factor $1 - \xi$. By an application of the Chernoff bound, in at least

$$2 \log_{1/(1-\xi)} Q_0 \geq 4 + \log_{1/(1-\xi)} Q_0$$

of the first $O(\log Q_0) = O(t + \log m)$ phases, either $d/4$ points have been collected or Q_p has decreased by a factor $1 - \xi$. This would imply either $4 \cdot d/4 = d$ points have been collected or all the urns are empty. ◀

3.3 Applications to Our Algorithm

We now circle back to Algorithm 1. To finish its analysis, we need to prove Lemma 4, which we can now do, as an application of what we have established about these urn processes. To analyze the while loop at Line 2, apply Theorem 9 with $t = \lfloor \log k \rfloor$ as follows. The balls in the i th urn correspond to the sets in \mathcal{F}_i^C . A ball is *gold* if the corresponding set S satisfies $|S \setminus C| \geq \theta_i$; otherwise, it is *lead*. Adding a set S to the candidate solution Y grows the set C of covered elements, thereby turning gold balls to lead in some complicated way that the algorithm cannot easily track. The bound on the number of phases until the urn process terminates translates to a bound of $O(\log k + \log m)$ on the number of iterations of that while loop. Noting that $k \leq m$ gives Lemma 4. Similarly, Theorem 6 implies that the number of iterations of the while loop at Line 7 is $O(\log m / \log \log m)$.

4 Random Order Model

In this section, we consider the stream to be a random permutation of the collection of sets. We will show that it is possible to approximate the maximum coverage problem up to a factor $1 - 1/e - \varepsilon$ using $O_\varepsilon(k \log m)$ space in a single pass. As mentioned in the introduction, our approach is based on modifying an algorithm by Agrawal et al. [1] for cardinality constrained monotone submodular maximization. This is a more general problem than maximum coverage, but their algorithm assumes oracle access to the function being optimized. For maximum coverage we need to store enough information to be able to simulate this oracle. A $O_\varepsilon(k^2 \log m)$ -space algorithm for maximum coverage follows immediately via the universe subsampling technique discussed in Section 2 since we may assume that every set has size $O_\varepsilon(k \log m)$. To reduce the dependence on k to linear rather than quadratic, we need a more careful adaptation of the algorithm of Agrawal et al. [1], to ensure that the algorithm can be implemented via maintaining a set of size $O_\varepsilon(k \log m)$ corresponding to the items covered by a collection of sets chosen for the solution so far, along with $O_\varepsilon(1)$ sets of size $O_\varepsilon(\log m)$.

Warm-Up. To motivate the approach, consider a variant of the problem in which an algorithm is presented with a random permutation of the input sets, but we make the following two changes:

40:10 Maximum Coverage in Dynamic and Random Order Streams

1. Fix an optimal collection of sets O_1, \dots, O_k . For each $i \in [k]$, replace the occurrence of O_i in the stream by a set drawn uniformly at random from $\{O_1, \dots, O_k\}$. These replacements are performed independently and so it is now possible that some sets amongst O_1, \dots, O_k never appears in the stream or appears multiple times.
2. We give the algorithm foreknowledge of a way to segment the stream into k contiguous sub-streams $\mathcal{C}_1, \dots, \mathcal{C}_k$ such that each contains exactly one element from $\{O_1, \dots, O_k\}$. Then consider the greedy algorithm that outputs the sequence $\langle S_1, \dots, S_k \rangle$ where

$$S_j = \operatorname{argmax}_{S \in \mathcal{C}_j} |S \setminus (S_1 \cup \dots \cup S_{j-1})| .$$

To analyze this algorithm, define $u_j = \operatorname{OPT} - |S_1 \cup \dots \cup S_j|$ and $F_j = \frac{|S_j \setminus (S_1 \cup \dots \cup S_{j-1})|}{u_{j-1}}$ and note that

$$|S_1 \cup \dots \cup S_k| = \operatorname{OPT} - u_k = \operatorname{OPT} - \operatorname{OPT} \cdot \prod_{j=1}^k (1 - F_j),$$

since for all $j \in [k]$,

$$u_j = \operatorname{OPT} - |S_1 \cup \dots \cup S_{j-1}| - |S_j \setminus (S_1 \cup \dots \cup S_{j-1})| = (1 - F_j)u_{j-1} .$$

Note also that

$$\begin{aligned} \mathbb{E}[F_j \mid S_1, \dots, S_{j-1}] &\geq \sum_{i=1}^k \frac{|O_i \setminus (S_1 \cup \dots \cup S_{j-1})|}{u_{j-1}} \cdot \Pr[O_i \in \mathcal{C}_j] \\ &= \frac{1}{k} \cdot \sum_{i=1}^k \frac{|O_i \setminus (S_1 \cup \dots \cup S_{j-1})|}{u_{j-1}} \geq 1/k . \end{aligned}$$

Hence,

$$\mathbb{E}[|S_1 \cup \dots \cup S_k|] = \operatorname{OPT}(1 - \mathbb{E}[\prod_{j=1}^k (1 - F_j)]) \geq 1 - (1 - 1/k)^k \rightarrow 1 - 1/e .$$

The space required to implement this algorithm is $O(k\varepsilon^{-2} \log m)$ since it suffices to maintain the union of the sets chosen thus far rather than the sets themselves.

To handle the original random order setting, we obviously need to deal with the fact that the algorithm does not have foreknowledge of a segmentation of the stream such that each segment contains exactly one set from the optimum solution. A naive approach would be to segment the stream into βk contiguous segments for some large constant $\beta > 0$ with the hope that few pairs of optimum sets appear in the same segment. We could then consider all $\binom{\beta k}{k} = e^{O(k)}$ subsequences of these segments but this is clearly inefficient. A better approach is to use limited number of guesses in parallel using an approach by Agrawal et al. [1]. More complicated to analyze, especially when combined with the goal of using limited space, is the fact that because the O -sets appear randomly permuted, rather than independently sampled, there are now various dependencies to contend with.

General Setting. We will randomly partition the input collection \mathcal{C} of sets into $k\beta$ groups $\mathcal{C}_1, \dots, \mathcal{C}_{k\beta}$. These groups will ultimately correspond to segments of the stream, i.e., the first $|\mathcal{C}_1| \sim \operatorname{Bin}(m, 1/(k\beta))$ sets define \mathcal{C}_1 etc. For any $\mathcal{I} = \{i_1, i_2, \dots\} \in \{1, \dots, k\beta\}$ we define a sequence of groups

$$\Sigma_{\mathcal{I}} = \langle \mathcal{C}_{i_1}, \mathcal{C}_{i_2}, \dots, \mathcal{C}_{i_{|\mathcal{I}|}} \rangle \quad \text{where } i_1 < i_2 < \dots < i_{|\mathcal{I}|} .$$

At the heart of the algorithm is a greedy process run on such sequences. For $\mathcal{I} = \{i_1, i_2, \dots\} \subset \{1, \dots, k\beta\}$, $C \subset [n]$, and a “reserve” collection of sets, \mathcal{R} – the name will become clear later – define the greedy sequence:

$$\sigma_{\mathcal{I}}(C, \mathcal{R}) = \langle S_1, S_2, \dots, S_{|\mathcal{I}|} \rangle \quad (2)$$

where $S_j = \operatorname{argmax}_{S \in \mathcal{R} \cup \mathcal{C}_{i_j}} |S \setminus (C \cup S_1 \cup \dots \cup S_{j-1})|$.

■ **Algorithm 2** Max Coverage for Random Order Streams.

-
- 1: initialize $\mathcal{R} \leftarrow \emptyset$, $C \leftarrow \emptyset$, $k' \leftarrow 0$.
 - 2: **for** $i \leftarrow 1$ to k/α **do**
 - 3: Compute $\sigma_{\mathcal{I}}(C, \mathcal{R})$ for all $\mathcal{I} \in P_i$, where P_i consists of all size $k^+ \equiv \min\{\alpha, k - k'\}$ subsets of

$$W_i = \{\alpha\beta(i-1) + 1, \dots, \alpha\beta i\} .$$

We will subsequently refer to W_i as the *ith window*.
 - 4: Let

$$\mathcal{I}^* = \operatorname{argmax}_{\mathcal{I} \in P_i} \left| C \cup \left(\bigcup_{S \in \sigma_{\mathcal{I}}(C, \mathcal{R})} S \right) \right|$$

and update

$$C \leftarrow C \cup \left(\bigcup_{S \in \sigma_{\mathcal{I}^*}(C, \mathcal{R})} S \right) \quad \text{and} \quad k' \leftarrow k' + k^+ .$$
 - 5: For all $\mathcal{I} \in P_i$, add all sets in $\sigma_{\mathcal{I}}(C, \mathcal{R})$ to \mathcal{R} . If there exists at least one set S in \mathcal{R} such that $|S \setminus C| \geq \text{OPT}/k$, select such a set uniformly at random and update

$$C \leftarrow C \cup S \quad \text{and} \quad k' \leftarrow k' + 1 .$$

Repeat until either $k' = k$ or no more such sets exist in \mathcal{R} .
-

Algorithm 2 contains most details, but to fully specify the algorithm, we need to define the original groups $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_{k\beta}$. To do this, for each set S define a random variable Y_S that is uniform over $\{1, 2, \dots, k\beta\}$. Then, we define $\mathcal{C}_i = \{S : Y_S = i\}$. Equivalently, for a random permutation of the stream, \mathcal{C}_1 is the first $|\{S : Y_S = 1\}|$ sets in the stream, \mathcal{C}_2 is the next $|\{S : Y_S = 2\}|$ sets in the stream, etc. Given this definition of the stream, Algorithm 2 maintains sets whose total size is

$$|\text{OPT}| + \alpha \binom{\alpha\beta}{\alpha} |\text{OPT}| + \alpha(k/\alpha) \binom{\alpha\beta}{\alpha} \text{OPT}/k = O_{\alpha, \beta}(\text{OPT}), \quad (3)$$

where the first term corresponds to maintaining C , the next term consists of all sets appearing in greedy subsequences during the processing of each window (defined in line 3 of Algorithm 2), and the last term corresponds to storing sets in \mathcal{R} . For sets S in \mathcal{R} it suffices to store $S \setminus C$ rather than S itself. Hence, the total number of elements in sets in \mathcal{R} is at most $|\mathcal{R}| \text{OPT}/k$.

The motivation for how to set α and β is as follows. Let $\mathcal{O} = \{O_1, \dots, O_k\}$ be an optimum collection of k sets. We say a group \mathcal{C}_i is *active* if it contains a set from \mathcal{O} and note that if β is large enough the number of active groups κ is close to k . Furthermore, in each window the number of active groups is expected to be $\kappa \cdot (\alpha\beta)/(k\beta) = \alpha\kappa/k \approx \alpha$. We will later set α to be large enough such that the number of active groups in each window is typically close to α , i.e., there is an α -length subsequence of the window that mainly consists of active groups.

Analysis. The analysis in Agrawal et al. [1] establishes that, during the i^{th} window, the algorithm finds a collection of α sets S_1, \dots, S_α – via one of the greedy subsequences considered – that, as explained below, covers a significant number of new elements when added to the solution. For any subset $A \subset [n]$, define $u(A) = (1 - \alpha/k) \text{OPT} - |A|$. Lemma 15 of Agrawal et al. [1] establishes that if C_{i-1} is the set of elements covered by the sets chosen during the processing of the first $i - 1$ windows, and T_{i-1} is the entire set of greedy subsequences constructed during the first $i - 1$ windows, then⁴

$$\mathbb{E}[u(S_1 \cup \dots \cup S_\alpha \cup C_{i-1}) \mid T_{i-1}] \leq e^{-\alpha(1-\delta)/k} u(C_{i-1}), \quad (4)$$

where $\delta = \sqrt{8/\beta}$, assuming $k \geq \alpha\beta$ and $\alpha \geq 4\beta^2 \log(\beta/8)$. The crucial observation is that their proof of Equation (4) holds true regardless of how C_{i-1} is constructed from the greedy subsequences in T_{i-1} . In particular, Equation (4) holds true given our modification of the Agrawal et al. algorithm, i.e., the possibility of adding additional sets from \mathcal{R} at the end of previous windows. At the end of the i^{th} window, we potentially add additional sets $S_{\alpha+1}, \dots, S_{\alpha'_i}$ where

$$\begin{aligned} u(S_1 \cup \dots \cup S_{\alpha'_i} \cup C_{i-1}) &\leq u(S_1 \cup \dots \cup S_\alpha \cup C_{i-1}) - (\alpha'_i - \alpha) \text{OPT} / k \\ &\leq (1 - (\alpha'_i - \alpha)/k) \cdot u(S_1 \cup \dots \cup S_\alpha \cup C_{i-1}) \\ &\leq e^{-(\alpha'_i - \alpha)/k} \cdot u(S_1 \cup \dots \cup S_\alpha \cup C_{i-1}) \end{aligned}$$

Hence, in the i^{th} window (except potentially in the window where we add the k^{th} set to our solution) we add $\alpha'_i \geq \alpha$ sets $S_1, \dots, S_{\alpha'_i}$ such that if $C_i = S_1 \cup \dots \cup S_{\alpha'_i}$

$$\mathbb{E}[u(C_i) \mid T_{i-1}] \leq e^{-\alpha'_i(1-\delta)/k} \cdot u(C_{i-1}). \quad (5)$$

Since there are k/α windows, and we find a sequence of length $\alpha'_i > \alpha$ until we have a sequence of length k , such that Eq. (5) holds in each window, for some $j \leq k/\alpha$, with $\delta < \varepsilon/2 < 1$,

$$\begin{aligned} \mathbb{E}[(1 - \alpha/k) \text{OPT} - |C_j|] &\leq (1 - \alpha/k) \cdot \text{OPT} \cdot \prod_{i \geq 1} e^{-\alpha'_i(1-\delta)/k} \\ &= (1 - \alpha/k) e^{-1+\delta} \text{OPT} < (1 - \alpha/k)(1/e + \varepsilon/2) \text{OPT}, \end{aligned}$$

where we used the fact $\sum_i \alpha'_i = k$ and $\exp(-1 + x) \leq 1/e + x$ for $0 < x < 1$. Hence, $\mathbb{E}[|C_j|] \leq 1 - 1/e - \varepsilon$ assuming $\alpha/k \leq \varepsilon/2$. Setting the constants to $\alpha = 4096\varepsilon^{-4} \log(2/\varepsilon)$ and $\beta = 32\varepsilon^{-2}$ ensures all the necessary conditions are met assuming $k = \omega(1)$. With these values of α and β , we have $\binom{\alpha\beta}{\alpha} \leq \exp(O(\varepsilon^{-4} \ln^2 \varepsilon^{-1}))$. Hence, the space dependence on ε in Eq. (3) becomes $O(k \log m \cdot \exp(O(\varepsilon^{-4} \ln \varepsilon^{-1})))$, since we may assume $\text{OPT} = O(k\varepsilon^{-2} \log m)$ and the exponential dependencies on ε dominate the polynomial dependencies.

5 Fast Algorithms

Fast Universe Sub-Sampling. The universe sub-sampling approach described in Section 2, requires the use of a γ -wise independent hash function where $\gamma = O(k \log m)$. This bound on the independence was actually an improvement by Jaud et al. [16] over the bound of $O(\varepsilon^{-2} k \log m)$ originally given by McGregor and Vu [27]. Jaud et al. were motivated to

⁴ Note the statement of [1, Lemma 15] has the RHS of Eq. (4) as $(1 - \delta)e^{-\alpha/k} u(C_{i-1})$, rather than the expression in Eq. (4), where the $(1 - \delta)$ appears in the exponent. Their proof actually only implies the weaker statement, in Eq. (4); fortunately, this is still sufficient for their and our purposes.

improve the bound not for the sake of reducing the space of any streaming algorithms, but rather to reduce the update time from $O(\varepsilon^{-2}k \log m)$ to $O(k \log m)$. However, we note that the update time, at least in an amortized sense, can actually be reduced to polylogarithmic. Specifically, we can use the hash family given by degree- γ polynomials over a suitably large finite field [32]. Although evaluating such a polynomial at a single point (which corresponds to hashing a single element) requires $\Omega(\gamma)$ time, evaluating the polynomial on γ points can actually be done in $O(\gamma \log^2 \gamma \log \log \gamma)$ time [30, Chapter 10]. Hence, by buffering sets of $O(k \log m)$ elements, we may ensure that the hash function can be applied with only $\text{poly}(\log k, \log \log m)$ amortized update time. We note that a similar approach was used by Kane et al. [18] in the context of frequency moment estimation.

Fast ℓ_0 Sampling. In the dynamic algorithm we needed to sample r sets with replacement from a stream in the presence of insertions and deletions. This can be done via ℓ_0 sampling as discussed in the preliminaries. However, a naive implementation of this process requires $\Omega(r)$ update time if r different ℓ_0 samplers need to be updated with every set insertion or deletion. To avoid this, a technique by McGregor et al. [26] can be used. Specifically, we use a $O(\log r)$ -wise independent hash function to partition $[M]$ into $t := r/\log r$ groups P_1, P_2, \dots, P_t . During the stream we compute $\rho_i = |\{x \in P_i : \sum_{j:x_j=x} \Delta_j \neq 0\}|$, i.e., the number of values in P_i that are inserted a different number of times from the number of times they are deleted. Note that it is simple to compute ρ_1, \dots, ρ_t on the assumption that $\sum_{j:x_j=x} \Delta_j \in \{0, 1\}$, i.e., every set is inserted either the same number of times it is deleted or exactly one extra time. We also compute $2 \log r$ independent ℓ_0 samplers for each P_i ; note that each set insert or delete requires updating at most $2 \log r$ independent ℓ_0 samplers. Then at the end of the stream, to generate a sequence of samples with replacement we first sample i with probability $\rho_i / \sum_j \rho_j$ and then use the next unused ℓ_0 sampler from group P_i . Assuming $r \ll |\{x \in [M] : \sum_{j:x_j=x} \Delta_j \neq 0\}|$ (if this is not the case, we can just use sparse recovery), then with high probability each $\rho_i / \sum_j \rho_j \leq 2(\log r)/r$ and the process will use at most $2 \log r$ independent ℓ_0 samplers from each group in expectation and less than $4 \log r$ with probability at least $1 - 1/\text{poly}(r)$.

6 Conclusion

We presented new algorithms for $1 - 1/e - \varepsilon$ approximation of the maximum coverage problem in the data stream model. These algorithms improve upon the state-of-the-art results by a) reducing the space required in the dynamic model from $(n + \varepsilon^{-4}) \text{polylog}(m, n)$ to $\varepsilon^{-2}k \text{polylog}(m, n)$ when given $O(\varepsilon^{-1} \log m)$ passes, b) reducing the space required in the single-pass random order model from $\varepsilon^{-2}k^2 \text{polylog}(m, n)$ to $O_\varepsilon(k \text{polylog}(m, n))$ although we emphasize that the O_ε hides an exponential dependence on ε , and c) reducing the amortized update time to polylogarithmic in m and n . We conjecture that $\varepsilon^{-2}k \text{polylog}(m, n)$ space is sufficient in the random order model. In fact there is a monotone submodular optimization algorithm in the random order model that, when applied to the maximum coverage problem would maintain only $O(\varepsilon^{-1}k)$ sets [25]. Given that, via universe sub-sampling, we may assume that the sets have size $O(\varepsilon^{-2}k \log m)$; this yielded the Warneke et al. [31] result. However, it seems unlikely that the algorithm of Liu et al. [25] can be modified to ensure that the total number of elements across all sets maintained is $O(\varepsilon^{-2}k \log m)$ so a new approach is likely to be necessary.

References

- 1 Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPICs*, pages 1:1–1:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ITCS.2019.1.
- 2 Aris Anagnostopoulos, Luca Becchetti, Ilaria Bordino, Stefano Leonardi, Ida Mele, and Piotr Sankowski. Stochastic query covering for fast approximate document retrieval. *ACM Transactions on Information Systems (TOIS)*, 33(3):1–35, 2015.
- 3 Sepehr Assadi and Sanjeev Khanna. Tight bounds on the round complexity of the distributed maximum coverage problem. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2412–2431. SIAM, 2018. doi:10.1137/1.9781611975031.155.
- 4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *STOC*, pages 698–711. ACM, 2016.
- 5 Giorgio Ausiello, Nicolas Boria, Aristotelis Giannakos, Giorgio Lucarelli, and Vangelis Th. Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13-14):1901–1913, 2012.
- 6 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 671–680, 2014.
- 7 Mohammad Hossein Bateni, Hossein Esfandiari, and Vahab S. Mirrokni. Almost optimal streaming algorithms for coverage problems. In Christian Scheideler and Mohammad Taghi Hajiaghayi, editors, *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pages 13–23. ACM, 2017. doi:10.1145/3087556.3087585.
- 8 Amit Chakrabarti and Anthony Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. In *SODA*, pages 1365–1373. SIAM, 2016.
- 9 Graham Cormode, Howard J. Karloff, and Anthony Wirth. Set cover algorithms for very large datasets. In *CIKM*, pages 479–488. ACM, 2010.
- 10 Yuval Emek and Adi Rosén. Semi-streaming set cover - (extended abstract). In *ICALP (1)*, volume 8572 of *Lecture Notes in Computer Science*, pages 453–464. Springer, 2014.
- 11 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.
- 12 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1363–1374, 2020.
- 13 Sarel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards tight bounds for the streaming set cover problem. In *PODS*, pages 371–383. ACM, 2016.
- 14 Dorit S Hochbaum and Anu Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics (NRL)*, 45(6):615–627, 1998.
- 15 Piotr Indyk and Ali Vakilian. Tight trade-offs for the maximum k -coverage problem in the general streaming model. In *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 200–217, 2019.
- 16 Stephen Jaud, Anthony Wirth, and Farhana Choudhury. Maximum Coverage in Sublinear Space, Faster. In *21st International Symposium on Experimental Algorithms (SEA 2023)*, volume 265, pages 21:1–21:20, Dagstuhl, Germany, 2023. doi:10.4230/LIPICs.SEA.2023.21.
- 17 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 49–58. ACM, 2011. doi:10.1145/1989284.1989289.

- 18 Daniel M Kane, Jelani Nelson, Ely Porat, and David P Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 745–754, 2011.
- 19 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.
- 20 David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.
- 21 Sanjeev Khanna, Christian Konrad, and Cezar-Mihail Alexandru. Set cover in the one-pass edge-arrival streaming model. In Floris Geerts, Hung Q. Ngo, and Stavros Sintos, editors, *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 127–139. ACM, 2023. doi:10.1145/3584372.3588678.
- 22 Andreas Krause and Daniel Golovin. Submodular function maximization. In Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014. doi:10.1017/CB09781139177801.004.
- 23 Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, pages 1650–1654. AAAI Press, 2007.
- 24 Ching Lih Lim, Alistair Moffat, and Anthony Wirth. Lazy and eager approaches for the set cover problem. In *Proceedings of the 37th Australasian Computer Science Conference*, pages 19–27, 2014.
- 25 Paul Liu, Aviad Rubinfeld, Jan Vondrák, and Junyao Zhao. Cardinality constrained submodular maximization for random streams. In Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 6491–6502, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/333222170ab9edca4785c39f55221fe7-Abstract.html>.
- 26 Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. In *MFCS (2)*, volume 9235 of *Lecture Notes in Computer Science*, pages 472–482. Springer, 2015.
- 27 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory Comput. Syst.*, 63(7):1595–1619, 2019. doi:10.1007/S00224-018-9878-X.
- 28 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavi-far, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3826–3835. PMLR, 2018. URL: <http://proceedings.mlr.press/v80/norouzi-fard18a.html>.
- 29 Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SDM*, pages 697–708. SIAM, 2009.
- 30 Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.
- 31 Rowan Warneke, Farhana Murtaza Choudhury, and Anthony Wirth. Maximum coverage in random-arrival streams. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPICs*, pages 102:1–102:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ESA.2023.102.
- 32 Mark N. Wegman and Larry Carter. New classes and applications of hash functions. In *Proc. 20th Annual IEEE Symposium on Foundations of Computer Science*, pages 175–182, 1979.
- 33 Huiwen Yu and Dayu Yuan. Set coverage problems in a one-pass data stream. In *SDM*, pages 758–766. SIAM, 2013.