Maximum Coverage in Sublinear Space, Faster

Stephen Jaud
School of Computing and Information Systems, The University of Melbourne, Australia

Anthony Wirth
School of Computing and Information Systems, The University of Melbourne, Australia

Farhana Choudhury
School of Computing and Information Systems, The University of Melbourne, Australia

Abstract

Given a collection of m sets from a universe $\mathcal{U}$, the *Maximum Set Coverage* problem consists of finding k sets whose union has largest cardinality. This problem is NP-Hard, but the solution can be approximated by a polynomial time algorithm up to a factor $1 - 1/e$. However, this algorithm does not scale well with the input size.

In a streaming context, practical high-quality solutions are found, but with space complexity that scales linearly with respect to the size of the universe $n = |\mathcal{U}|$. However, one randomized streaming algorithm has been shown to produce a $1 - 1/e - \varepsilon$ approximation of the optimal solution with a space complexity that scales only poly-logarithmically with respect to m and n. In order to achieve such a low space complexity, the authors used two techniques in their multi-pass approach:

- F_0-*sketching*, allows to determine with great accuracy the number of distinct elements in a set using less space than the set itself.
- *Subsampling*, consists of only solving the problem on a subspace of the universe. It is implemented using γ-independent hash functions.

This article focuses on the sublinear-space algorithm and highlights the time cost of these two techniques, especially subsampling. We present optimizations that significantly reduce the time complexity of the algorithm. Firstly, we give some optimizations that do not alter the space complexity, number of passes and approximation quality of the original algorithm. In particular, we reanalyze the error bounds to show that the original independence factor of $\Omega(\varepsilon^{-2} k \log m)$ can be fine-tuned to $\Omega(k \log m)$; we also show how F_0-*sketching* can be removed. Secondly, we derive a new lower bound for the probability of producing a $1 - 1/e - \varepsilon$ approximation using only pairwise independence: $1 - \frac{4}{ck \log m}$ compared to $1 - \frac{2e}{m^{ck/6}}$ with $\Omega(k \log m)$-independence.

Although the theoretical guarantees are weaker, suggesting the approximation quality would suffer, for large streams, our algorithms perform well in practice. Finally, our experimental results show that even a pairwise-independent hash-function sampler does not produce worse solution than the original algorithm, while running significantly faster by several orders of magnitude.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases streaming algorithms, subsampling, maximum set cover, k-wise independent hash functions

Digital Object Identifier 10.4230/LIPIcs.SEA.2023.21

Related Version *Full Version*: `https://arxiv.org/abs/2302.06137`

Supplementary Material
Software (Source Code): `https://github.com/caesiumCode/streaming-maximum-cover`
archived at `swh:1:dir:1012da79a9177f4dc0ae4e5851608b597e79fa8d`

Funding This research was supported by the Australian Government through the Australian Research Council's Discovery Projects funding scheme (project DP190102078).
Farhana Choudhury: Farhana Choudhury is a recipient of the ECR22 grant from The University of Melbourne.

Acknowledgements Rowan Warneke, for reading and advising on an earlier version.

© Stephen Jaud, Anthony Wirth, and Farhana Choudhury;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Maximum Coverage, also known as *Maximum-k-Coverage* is a classic problem in computer science. Unless P = NP, the decision version is unsolvable in polynomial time. The input is a **family** of m sets, $\mathcal{F}$, each a subset of universe $\mathcal{U}$, comprising n elements, and a positive **integer**, k. The task is to find a subfamily of k sets in $\mathcal{F}$ whose union has largest cardinality. The best-known polynomial-time approximation algorithm for *Max Coverage* and the "dual" *Set Cover* problem[1], is a greedy approach. For Max Coverage, the greedy algorithm has been shown to return a solution whose coverage is at least a $1 - 1/e$ approximation of the optimal solution. This is known to be asymptotically optimal [11].

In practice, the greedy algorithm is much more effective than its theoretical guarantee would suggest, and typically produces a near-optimal solution on realistic inputs [14]. However, the greedy algorithm does not scale well with the size of the input. In the last 15 years, there has been increasing interest in efficient implementation of greedy and greedy-like approaches for Set Cover and Maximum Coverage [7, 10, 20]. In the streaming setting, there have been several innovative algorithms, as detailed below in Table 1. We focus in this paper on engineering the only sublinear-space Set Streaming algorithm [17] so that it runs much faster, and sacrifices no space.

In the Set Streaming model [20], the input stream, $\mathcal{S}$, comprises a sequence of the sets in $\mathcal{F}$, i.e., $\mathcal{S} = S_1, S_2, \ldots, S_m$. Each set S_i in $\mathcal{S}$ appears in full before the next set, S_{i+1}, appears. The design of a streaming algorithm trades off memory, throughput, query/solution time, and solution quality. Let I denote the indexes of the sets in the solution (so far). The coverage of (the sets in) I is $C = \cup_{i \in I} S_i$. Given I, and hence C, the *contribution* of each set S_j, for every $j \notin I$ is $S_j \setminus C$. In the greedy algorithm, we add a set to the solution whenever it has largest contribution, breaking ties arbitrarily. Additionally, another well-studied variant of this streaming model is *random set arrivals*, a reasonable assumption for many applications, and it makes the problem *easier*. Many results regarding trade-off between space complexity and approximation factor improve upon the classic *set arrival* setting [1, 18]. Another common model for Maximum Coverage, although not discussed in this paper, is the Edge-arrival Streaming model. Here the stream consists of pairs $(i, x) \in [m] \times \mathcal{U}$ to indicates that $x \in S_i$. In this more general context, Indyk and Vakilian [15] showed a space lower bound $\Omega(ma^2)$ and upper bound $\tilde{\mathcal{O}}(ma^2)$ for an arbitrary factor a-approximation factor in single pass.

1.1 Sublinear Space

Several of the greedy-like approaches for Set Cover in the Set Streaming model assume $\Omega(n)$ memory is available [7, 10, 16]: at least one bit per item, to record the coverage, and thus determine a set's contribution. Unlike Maximum Coverage, in Set Cover, we expect that the subfamily of sets returned, indexed by I, covers all of $\mathcal{U}$, so $n = |\mathcal{U}|$ bits seem necessary. In contrast, for Max Coverage, the minimum space requirement seems depend on m. For example, it is known that every one-pass $(1/2 + \varepsilon)$-approximation algorithm must work in $\Omega(\varepsilon m/k^3)$ space [12]. Also, $\Omega(m)$ space is necessary to achieve better approximation factors than $1 - 1/e$ [17]. Regarding $(1 - \varepsilon)$-approximation algorithms, Assadi [2] showed that $\tilde{\Omega}(m/\varepsilon^2)$ space is required. It should be noted that all these lower bounds are tight and several one-pass $\tilde{\mathcal{O}}(\varepsilon^d m)$-space algorithms do exist [4, 17].

[1] In Set Cover, the aim is to return a subfamily of *minimum cardinality* whose union is $\mathcal{U}$.

In this context, one algorithm for Maximum Coverage stands out. McGregor and Vu [17] introduced a family of streaming algorithms for Max Coverage. They describe, in §2.2 of their paper, an approximation algorithm that in $\mathcal{O}(\varepsilon^{-1})$ passes and in $\tilde{\mathcal{O}}(\varepsilon^{-2}k)$ space returns a $(1 - 1/e - \varepsilon)$-approximate solution[2]. This is the only reasonable approximation algorithm for Max Coverage that runs in $o(\min\{m, n\})$ space. For convenience[3], we name this algorithm MACH_*. Like some of the first streaming/external-memory algorithms for Set Cover, MACH_* takes multiple passes, achieving a near-greedy approach via a sequence of decreasing thresholds for the contribution of a set: further details of thresholding are in §1.4. And to save space, MACH_* has a randomized subsampling component achieved with multi-way independent hash functions. These hash functions are slow to evaluate, and it is this component that we accelerate.

1.2 Motivation

In terms of approximation quality and space complexity, MACH_* is the favored approach for Maximum Coverage. The space complexity of $\tilde{\mathcal{O}}(\varepsilon^{-2}k)$ is only a little more than the space required to store[4] solution I: $\tilde{\Omega}(k)$. However, in MACH_*, McGregor and Vu [17] invoke a γ-independent hash function, where $\gamma = \lceil 2c\varepsilon^{-2}k \log m \rceil$, with c a constant to be discussed in §4.1. At first glance, this seems to slow the algorithm down, as $\Omega(\gamma)$ operations are required for each component of the input. Our experiments (refer to Figure 1 below) confirm that the running time of MACH_* is particularly high compared to other alternatives. Our research motivation is:

> Can we accelerate this space-efficient Max Coverage algorithm, MACH_*, without significantly deteriorating space complexity or solution quality?

One promising direction is to *simplify* the subsampling process. McGregor and Vu show that, with $\gamma = \lceil 2c\varepsilon^{-2}k \log m \rceil$, an approximation factor of $1 - 1/e - \varepsilon$ is guaranteed with probability at least $1 - 1/m^{10k}$. In the original version of MACH_*, this γ parameter can easily exceed 10^3. So we would anticipate a thousand-fold reduction in throughput compared to a simpler, if theoretically less guaranteed, *sampling scheme*, such as pairwise independent hashing. Since we are designing a space-efficient algorithm, a pre-computed hash function table is infeasible.

1.3 Our contributions

Firstly, we show that the same space complexity and approximation quality can be achieved with $\Theta(k \log m)$ independence (Corollary 5) instead of the original $\Theta(\varepsilon^{-2}k \log m)$ and in fact without invoking F_0-sketching (Lemma 6). Removing F_0-sketching slightly reduces the probability of producing a $1 - 1/e - \varepsilon$ approximation from $1 - e/m^{ck/6}$ to $1 - 2e/m^{ck/6}$.

[2] In this paper, the $\tilde{\mathcal{O}}(\cdot)$ notation hides polylogarithmic factors in m and n.
[3] MACH represents "Maximum Andrew Coverage Hoa": the $*$ represents their parameter choices, which we generalize in this paper.
[4] An approach that avoids storing at least one bit per index in I, as working space, is in principle possible. For example, I could be a size-k subset of $\{1, \ldots, m\}$ chosen uniformly at random; this is not an effective solution, but a valid one, generated in $\tilde{\mathcal{O}}(1)$ working space.

Secondly, if a weaker probabilistic guarantee on the approximation quality is allowed, we show that the algorithm still works with only pairwise independence (Proposition 8). This leads to a significant speed-up, from 10× to more than 1000× for $k \geq 100$ (Figure 1), while maintaining the same space complexity[5] as the original algorithm of McGregor and Vu [17].

Finally, our experimental results demonstrate the efficiency and quality of our generalized algorithm, MACH'_γ. In particular, reducing the independence factor does not lead to significantly worse solutions. We show that for reasonable values (< 0.27) of ε, our algorithm returns consistently better solutions than comparator streaming algorithms (Figure 3).

1.4 Related Work

Thresholding. Before surveying the algorithms for Max Coverage, we set out one of the important algorithmic frameworks. Several algorithms invoke a thresholding technique, first applied to Set Cover by Cormode et al. [10]. It *relaxes* the notion of greedy algorithm, and calculates a near-greedy solution. Instead of *searching* for the set whose contribution is $R^* = \max_j |S_j \setminus C|$, a thresholding algorithm might add a set S_i to the solution if its contribution is at least αR^*, where $\alpha \in [0, 1]$ describes the *greediness* of the thresholding algorithm. Applying this principle repeatedly results in a solution whose coverage is $\alpha(1-1/e)$ fraction of the optimum coverage.

Now the guarantee of αR^* contribution arises from a multi-pass approach to the stream. In pass j, all sets with contribution at least r are added, then in pass $j + 1$, all sets with contribution at least αr are added. Since a set's contribution can only decrease as (other) sets are added to I, with this approach, we only add a set if its contribution is αR^*.

Prior art. There are several existing streaming algorithms for the Max Coverage problem, which we summarize in Table 1. Badanidiyuru et al. [3] presented a generic algorithm for maximizing submodular functions on a stream, which can be adapted to Max Coverage. This is a one-pass thresholding algorithm, somewhat similar to MACH_*, that *guesses* the optimal coverage size. Yu and Yuan [22] developed an algorithm that creates a specific ordering $(\tilde{S}_1, \ldots, \tilde{S}_m)$ of the entire collection of sets $\{S_1, \ldots, S_m\}$ such that for all k, $(\tilde{S}_1, \ldots, \tilde{S}_k)$ is a solution of the *Maximum-k-Coverage*. Saha and Getoor [20], who pioneered set streaming, took a *swapping* approach. A putative solution of k sets is stored, and sets in the putative solution can be replaced by new sets in the stream depending on the number of items uniquely covered by sets in the putative solution. More recently, Bateni et al. [4] used a sketching technique and they almost match the optimal approximation factor of $1 - 1/e$. This is an algorithm designed for the edge-arrival streaming model, but can be adapted to the set streaming model with a space complexity independent to the size of the universe. Norouzi-Fard et al. [18], in the continuation of Badanidiyuru et al. [3], presented a 2-pass and a multi-pass approach to maximize a submodular function on a stream. Developed at a similar time, McGregor and Vu [17] presented two polynomial-time algorithms that achieve the same approximation factor of $1 - 1/e - \varepsilon$: one taking a single pass, the other, MACH_*, taking multiple passes. The algorithms developed by McGregor and Vu [17] are *thresholding* algorithms.

[5] Actually, removing F_0-sketching and reducing the independence factor strictly reduces the space complexity, although not asymptotically.

Table 1 Streaming algorithms for Maximum Coverage. We focus on the $o(\min\{m, n\})$-space algorithm, MACH_*.

Author	Name	Passes	Space	Approx.
Badanidiyuru et al. [3]	BMKK	1	$\tilde{\mathcal{O}}(\varepsilon^{-1}n)$	$1/2 - \varepsilon$
Yu and Yuan [22]		1	$\tilde{\mathcal{O}}(n)$	~ 0.3
Saha and Getoor [20]	SG	1	$\tilde{\mathcal{O}}(kn)$	$1/4$
Bateni et al. [4]		1	$\tilde{\mathcal{O}}(\varepsilon^{-3}m)$	$1 - 1/e - \varepsilon$
Norouzi-Fard et al. [18]	2P	2	$\tilde{\mathcal{O}}(\varepsilon^{-1}n)$	$5/9 - \varepsilon$
Norouzi-Fard et al. [18]		$\mathcal{O}(\varepsilon^{-1})$	$\tilde{\mathcal{O}}(\varepsilon^{-1}n)$	$1 - 1/e - \varepsilon$
McGregor and Vu [17]	OP	1	$\tilde{\mathcal{O}}(\varepsilon^{-2}m)$	$1 - 1/e - \varepsilon$
McGregor and Vu [17]	MACH_*	$\mathcal{O}(\varepsilon^{-1})$	$\tilde{\mathcal{O}}(\varepsilon^{-2}k)$	$1 - 1/e - \varepsilon$

Sampling. Sampling via hashing is a key component of many streaming algorithms. Relaxing the independence requirement for hash functions was explored in the context of ℓ_0-samplers: Cormode and Firmani [9] invoked γ-independent hash functions. They showed theoretical bounds on γ to guarantee the probability of sampling a non-zero coordinate. In addition, their experimental results suggest that *constant*-independence hashing schemes produce similar successful sampling rate to *linear*-independent hash functions, while being significantly more efficient to compute.

Furthermore, some theoretical results [19] show that many strong guarantees generally associated with *high*-independence families of hash functions can be achieved with simpler hashing schemes. Tabulation hashing [19], for example, is not even 4-independent, but manages to implement γ-independent hash function based algorithms, such as *Cuckoo Hashing*. Pătraşcu and Thorup [19] also prove Chernoff-type inequalities with relaxed assumptions on the independence of the random variables.

2 Tools

The Introduction includes most of our notation; in addition, we let I_{OPT} be an optimal solution and OPT the size of the optimal coverage $\left|\bigcup_{i \in I_{\mathrm{OPT}}} S_i\right|$.

2.1 Subsampling

▶ **Definition 1** (subsampling). *Given $\mathcal{F}$, $\mathcal{U}$, and hash function $h : \mathcal{U} \to \{0, 1\}$, the subsampled universe is $\mathcal{U}' = \{x \in \mathcal{U} \mid h(x) = 1\}$, with subsampled sets $S' = S \cap \mathcal{U}'$ for every $S \in \mathcal{F}$.*

Instead of computing with respect to universe $\mathcal{U}$, algorithm MACH_* focuses on $\mathcal{U}' \subset \mathcal{U}$, and tracks only the subsampled coverage $C' = \cup_{i \in I} S'_i$. The size of the optimal coverage of $\mathcal{U}'$, by a subfamily of k sets from $\mathcal{F}$, is henceforth called OPT'.

▶ Remark. The value $\mathrm{OPT}' = \max_{|J|=k} \left|\bigcup_{i \in J} S'_i\right|$ is not necessarily the same as the size of the union of the subsampled sets in the optimal coverage of $\mathcal{U}$, i.e., $\left|\bigcup_{i \in I_{\mathrm{OPT}}} S'_i\right|$.

▶ **Definition 2.** *Let $\gamma, v, p \in \mathbb{N}$ such that $p > |\mathcal{U}|$:*

$$\mathcal{H}_{\gamma,v} = \left\{ x \longmapsto \sum_{i=0}^{\gamma-1} a_i x^i \bmod p \mod v \mid 0 \le a_i < p \right\}$$

is a family of hash functions $\mathcal{H}_{\gamma,v} \subset \{f : \mathcal{U} \to [v-1]\}$

Such a family has the property of being γ-*independent*[6]. Evaluating a hash function $f \in \mathcal{H}_{\gamma,v}$ takes $\Theta(\gamma)$ operations, including expensive *modulo* operations, but these can be accelerated using the overflow mechanism on unsigned integer types. $\mathcal{H}_{\gamma,v}$ are the families of hash functions used in MACH_*.

2.2 Sketching

To estimate the size of a set, McGregor and Vu invoke F_0 sketching.

▶ **Theorem 3** (F_0-sketching [8]). *Given a stream s, there exists a data structure, $\mathcal{M}(s)$, that requires $\mathcal{O}(\varepsilon^{-2}\log\delta^{-1})$ space and, with probability $1-\delta$, returns the number of distinct elements in s within multiplicative factor $1 \pm \varepsilon$. Processing each new element takes $\mathcal{O}(\varepsilon^{-2}\log\delta^{-1})$ time, the same time as finding the number of distinct elements.*

2.3 Thresholding on the sampled universe

The core of MACH_* is thresholding and subsampling. The solution, I, and the associated subsampled coverage $C' = \cup_{i\in I} S'_i$ are built incrementally, as new sets arrive in the stream and are selected. Given a threshold, r, the selection rule for set S_i is:

$$\text{If } |S'_i \setminus C'| \geq r, \text{ then } I \leftarrow I \cup \{i\} \text{ and } C' \leftarrow C' \cup S'_i\,, \tag{1}$$

where $|S'_i \setminus C'|$ is called the *contribution* of S'_i – from the context, it is clear this is in the sampled universe. In choosing the sequence of thresholds there is a trade-off [7]: the larger the threshold, the higher the solution quality, but the more passes.

3 Low-space Streaming Algorithm

In this section, we describe in detail MACH_* developed by McGregor and Vu [17], which solves Max Cover in sublinear space with a respectable approximation factor. Algorithm MACH_* depends on two variables:

- v, an estimate of the optimal coverage, OPT; and
- λ, an estimate of the optimal coverage on the subsampled universe, OPT′.

These variables determine the probability of subsampling an element, and the initial value of the threshold, r, as applied above (1). The subsampling hash function is implemented as $h(x) = \mathbb{1}_{f(x)<\lambda}$ where $f \in \mathcal{H}_{\lceil 2\lambda\rceil,v}$, hence the probability an item is subsampled is λ/v. The threshold, r, is initially set to $2(1+\varepsilon)\lambda/k$ and after each pass, r decreases by a factor $1+\varepsilon$. McGregor and Vu [17] showed that if

$$\lambda = c\varepsilon^{-2}k\log m\,, \qquad \text{with } c \geq 60\,, \text{ and} \qquad \mathrm{OPT}/2 \leq v \leq \mathrm{OPT}\,, \tag{2}$$

then this thresholding procedure, which we call TP, gives a $1-1/e-\varepsilon$ approximation using $\tilde{\mathcal{O}}(\varepsilon^{-2}k)$ space with probability at least $1-1/m^{10k}$.

3.1 Guessing

Algorithm MACH_* relies on a reasonable estimate of OPT: a v such that $\mathrm{OPT}/2 \leq v \leq \mathrm{OPT}$. Of course, we do not know OPT in advance! The algorithm naively finds the right value for v by executing TP_v for different values of v, called *guesses*, in parallel. Denote by v_g the

[6] Different definitions exist; our definition of γ-independent is stated in the Appendix.

Algorithm 1 Algorithm $\mathsf{MACH}_\gamma(\mathcal{S}, k, \varepsilon, ||S||_\infty)$.

begin
- /* **Initialise the guesses** */
- $V \leftarrow \{2^{g-1}||S||_\infty \le \min(n, k||S||_\infty),\ g \in \mathbb{N}\}$
- Duplicates each variable $|V|$ times: h, I, C', $\mathcal{M}$ and `active`
- $r \leftarrow 2(1+\varepsilon)\lambda/k$
- /* **Multiple passes** */
- **for** $p \leftarrow 1$ ***to*** $1 + \lceil \log_{1+\varepsilon}(4e) \rceil$ **do**
  - /* **One pass** */
  - **for** $S_i \in \mathcal{S}$ ***(stream)*** **do**
    - /* **Iterate over the guesses** */
    - **for** $g \leftarrow 0, \ldots, |V| - 1$ **do**
      - $S'_i \leftarrow$ Subsample S_i with h_g
      - $R_i \leftarrow S'_i \setminus C'_g$ /* **Contribution** */
      - /* **Check the *bad guess* condition** */
      - **if** $|C'_g| + |R'_i| > 2(1+\varepsilon)\lambda$ **then**
        - $\texttt{active}_g \leftarrow \texttt{false}$
      - /* **Thresholding procedure** */
      - **if** $\texttt{active}_g$ ***and*** $|I_g| < k$ ***and*** $|R_i| \ge r$ **then**
        - update $\mathcal{M}_g$ with S_i
        - $C'_g \leftarrow C'_g \cup R_i$
        - $I_g \leftarrow I_g \cup \{i\}$
  - /* **Update the threshold** */
  - $r \leftarrow r/(1+\varepsilon)$
- /* **Find the best coverage among the potentially correct guesses** */
- $s \leftarrow \operatorname{argmax}_{\texttt{active}_g} \{|\mathcal{M}_g|\}$
- **return** I_s

g^{th} guess. To reduce the number of guesses, we assume the maximum set size, which we call $||\mathcal{F}||_\infty$, is known. This assumption requires only one additional pass through the set stream, $\mathcal{S}$: the *asymptotic* number of passes is unchanged. Hence the guesses for v can be restricted to all the values $v_g = 2^{g-2}||\mathcal{S}||_\infty$, with $g \ge 1$, smaller than $k||\mathcal{S}||_\infty$. These *parallel* instantiations increase the running time and space by factor $\log_2 k$: there are separate copies of variables I, C' and h (the subsampling hash function) for each guess: these variables for guess v_g are I_g, C'_g and h_g. Now I, C and C' refer to the variable associated with the output of the algorithm.

Which is the right guess?

This guessing method begs the question: how do we detect the *right* guess? Also, MACH_* is only guaranteed to *work* under the condition $\mathrm{OPT}/2 \le v \le \mathrm{OPT}$. Some instances, with a *wrong* guess, might necessitate more space than the bound $\tilde{\mathcal{O}}(\varepsilon^{-2}k)$. McGregor and Vu introduce two mechanisms to deal with these questions.

First, the right guess is found by estimating the (non-subsampled) coverage of $\mathcal{U}$associated with each guess: the biggest coverage is considered the right guess. However, only the subsampled coverages, of $\mathcal{U}'$, are calculated. To resolve this, McGregor and Vu adopt F_0-*sketching*, see Theorem 3, which approximates the number of distinct elements in a collection of sets using less space than the collection itself. More particularly, in addition to the subsampled coverage, C'_g, for each v_g, MACH_* maintains a *sketch* $\mathcal{M}_g$ of the coverage in

$\tilde{\mathcal{O}}(\varepsilon^{-2})$ space. Each time a set is selected, $\mathcal{M}_g$ is updated accordingly. Once all the instances TP_g are completed, the sketches $\{\mathcal{M}_g\}$ determine which guess produced the biggest coverage. We still need the subsampled coverages, $\{C'_g\}$, for calculating the set's contributions: $\{S_i \setminus C'_g\}$. The F_0-sketches are too inefficient to be queried that often.

Second, the space complexity never exceeds $\tilde{\mathcal{O}}(\varepsilon^{-2}k)$ due to a consequence of Corollary 9 of McGregor and Vu [17]: if v is the right guess then $\mathrm{OPT}' \leq 2(1+\varepsilon)\lambda$. Thus, if the subsample coverage C'_g of an instance TP_g exceeds $2(1+\varepsilon)\lambda$, the associated guess is necessarily wrong and this instance can be terminated (Line 14 in Algorithm 1). Hence every instance runs in space $\mathcal{O}(\lambda) = \tilde{\mathcal{O}}(\varepsilon^{-2}k)$. Thus, λ can be referred as the *space budget* of the algorithm.

3.2 Properties

Algorithm 1 is our generalisation of MACH_*, which we call MACH_γ. The independence factor, γ, is not fixed, but is instead a parameter that influences the implementation of the subsampling hash functions $\{h_g\}$. The original algorithm, MACH_*, has an independence factor of $\lceil 2\lambda \rceil$: **so $\mathsf{MACH}_* = \mathsf{MACH}_{\lceil 2\lambda \rceil}$ in our generalization**.

McGregor and Vu [17] showed that MACH_* has space complexity $\tilde{\mathcal{O}}(\varepsilon^{-2}k)$ and with probability at least $1 - \frac{1}{m^{10k}}$ produces a $1 - 1/e - \varepsilon$ approximation. As our focus is improving run time, with only a small trade-off in the other properties, we first dissect the run time. The coverages C'_g are implemented with hash tables in order to efficiently compute $R_i = S'_i \setminus C'_g$. If the g^{th} guess is reading the i^{th} set then:

Line 11: Subsampling S_i: $\mathcal{O}(\gamma)|S_i|$ time (evaluate degree-$\mathcal{O}(\gamma)$ polynomial for each element in S_i)
Line 12: Computing R_i: $\mathcal{O}(|S'_i|) \subset \mathcal{O}(|S_i|)$ time
Line 19: Updating C'_g: $\mathcal{O}(|R_i|) \subset \mathcal{O}(|S_i|)$ time
Line 20: Updating $\mathcal{M}_g$: $\mathcal{O}(\varepsilon^{-2}\log n)|S_i|$ time (Theorem 3)

Therefore, the expected time complexity, T_γ, of MACH_γ is

$$\begin{aligned} T_\gamma &= \underbrace{\mathcal{O}(\log k)}_{\text{guesses}} \left(\underbrace{\mathcal{O}(\varepsilon^{-1})}_{\text{passes}} \cdot \underbrace{\mathcal{O}\left(\sum_{i=1}^{m} \gamma |S_i|\right)}_{\text{subsampling}} + \underbrace{\mathcal{O}\left(\sum_{i\in I} \varepsilon^{-2}|S_i|\log n\right)}_{F_0\text{-sketching}} \right) \\ &= \mathcal{O}\left(\varepsilon^{-1}\gamma m|\overline{\mathcal{S}}|\log k + \varepsilon^{-2}k|\overline{\mathcal{C}}|\log n \log k\right) \end{aligned} \tag{3}$$

► Note. $|\overline{\mathcal{S}}| = \frac{1}{m}\sum_i |S_i|$ is the average set size over the entire stream, $\mathcal{S}$, while $|\overline{\mathcal{C}}| = \frac{1}{k}\sum_{i\in I}|S_i|$ is the average set size over the selected sets (in I).

Therefore, $\mathsf{MACH}_* = \mathsf{MACH}_{\lceil 2\lambda \rceil}$ has expected time complexity of

$$T_{\lceil 2\lambda \rceil} = \mathcal{O}\left(\varepsilon^{-3}km|\overline{\mathcal{S}}|\log m \log k + \varepsilon^{-2}k|\overline{\mathcal{C}}|\log n \log k\right) . \tag{4}$$

Regarding the space complexity for $\gamma < \lceil 2\lambda \rceil$, it remains (asymptotically) the same. Indeed the cost for storing a γ-independent hash function is $\mathcal{O}(\gamma) \subset \tilde{\mathcal{O}}(\varepsilon^{-2}k)$.

Interestingly, MACH_γ does not guarantee the solution returned actually has k sets. Given that $|I| \leq k$, we can simply append $k - |I|$ random indices to the returned solution. However, the goal of this paper is to assess the probabilistic nature of the algorithm that arises from the γ-independent hash functions. Therefore, in our experiments in §5, we do not alter the returned solution given by MACH_γ.

4 Accelerating the Algorithm

Algorithm MACH_γ is a breakthrough: it runs in sublinear space. As we surmise from the expression (3) for T_γ, however, the high independence factor, γ, induces a significant bottleneck. Our experiments in §5 validate this conjecture. In this section we demonstrate how to improve the running time without too much sacrifice in the other properties. Indeed, when lowering the independence factor, γ, we maintain the space complexity and the number of passes.

McGregor and Vu [17] showed that if $\mathrm{OPT}/2 \leq v$ and $\big|\, |C'| - p|C| \,\big| < \varepsilon v p$ then the solution produced by MACH_γ is a $1 - 1/e - \varepsilon$ approximation. Assuming condition $\mathrm{OPT}/2 \leq v \leq \mathrm{OPT}$ is met, i.e., we have the right guess for v, $\mathbb{P}(E_I)$ is a lower bound on the probability of producing a $1 - 1/e - \varepsilon$ approximation, where E_I is the event $\big\{ \big|\, |C'| - p|C| \,\big| < \varepsilon v p \big\}$. Therefore, the goal is to estimate $\mathbb{P}(E_I)$ for independence factor γ smaller than $\lceil 2\lambda \rceil$.

We provide proofs of several of the following results in the Appendix.

4.1 Maintaining the Approximation Property

Recalculating c. The first optimisation is actually an observation about the constant c in the definition of $\lambda = c\varepsilon^{-2} k \log m$. Indeed, the independence factor $\gamma = \lceil 2\lambda \rceil$ depends on c: the authors state "Let c be some sufficiently large constant." At first glance, it seems that to get $\mathbb{P}(\overline{E_I})$, an upper bound on the failure probability, below e/m^{10k}, the constant c must be greater than 60 because the inequality in their Lemma 8 is $\mathbb{P}(\overline{E_I}) \leq e/m^{ck/6}$. Indeed, with a bound on $\mathbb{P}(\overline{E_I})$, we apply a union bound to upper bound $\mathbb{P}\left(\cup_{|I|=k} \overline{E_I}\right)$. Unpacking this, we find the inequalities

$$\mathbb{P}\left(\bigcup_{|I|=k} \overline{E_I} \right) \leq \sum_{|I|=k} \mathbb{P}(\overline{E_I}) \leq \binom{m}{k} \frac{e}{m^{ck/6}} \leq \frac{e}{k!\; m^{(c/6-1)k}} ,$$

whence we conclude that $c = 6$ is the smallest reasonable value to be sure the upper bound is $o(1)$. This reduction in c does not reduce the asymptotic time complexity of MACH_γ, but in practice it reduces the independence factor by a factor 10 so this is still a $10\times$ speed up compared to the original $c \geq 60$.

Reducing γ. In order to reduce further the independence factor γ, we express $\mathbb{P}(E_I)$ with respect to γ. To that end, we use the following concentration bound, from the same paper cited by McGregor and Vu.

▶ **Theorem 4** (Schmidt et al. [21]). *Let $X_1, \ldots, X_n$ be γ-wise independent r.v.s, $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}(X)$. If $X_i \in [0,1]$ and $\gamma \leq \lfloor \min(\delta, \delta^2) \mu e^{-1/3} \rfloor$ then $\mathbb{P}(|X - \mu| \geq \delta\mu) \leq e^{-\lfloor \gamma/2 \rfloor}$.*

Notice that $|C'| = \sum_{i=1}^n X_i$ where $X_i = \mathbb{1}_{i \in C} \mathbb{1}_{h(i)=1} \in [0,1]$; since $p = \lambda/v$ is the probability of subsampling an element, $\mathbb{P}(h(i) = 1)$, we have the following corollary:

▶ **Corollary 5.** *If $\gamma \leq \lfloor \frac{c}{3} k \log m \rfloor$, with I, C, and C' defined accordingly, then:*

$$\mathbb{P}\left(\big|\, |C'| - p|C| \,\big| \geq \varepsilon v p \right) \leq e^{-\lfloor \gamma/2 \rfloor} .$$

Consequently, by setting $\gamma = \lfloor \frac{c}{3} k \log m \rfloor = \mathcal{O}(\varepsilon^2 \lambda)$, compared to the original $\mathcal{O}(\lambda)$, we keep the same approximation guarantees as McGregor and Vu [17]:

$$\mathbb{P}(\overline{E_I}) \leq e^{-\left\lfloor \lfloor \frac{c}{3} k \log m \rfloor / 2 \right\rfloor} = e^{-\left\lfloor \frac{c}{6} k \log m \right\rfloor} \leq e^{-\frac{c}{6} k \log m + 1} = \frac{e}{m^{ck/6}} .$$

Algorithm 2 Procedure `FindGuess`.

$s \leftarrow |V| - 1$
while $|C'_s| < (1-\varepsilon)(1-1/e-\varepsilon)\lambda$ ***or*** $\neg\mathtt{active}_s$ **do**
  $s \leftarrow s - 1$

Removing F_0-sketching. First, it should be noted that McGregor and Vu showed that MACH_* produces a $1-1/e-\delta(\varepsilon)$ approximation of the optimal coverage where $\delta(\varepsilon) = \varepsilon(3-1/e-\varepsilon) \leq 2.6\varepsilon$. Asymptotically, the statement of McGregor and Vu is right because the algorithm can simply start by dividing ε by 3 and it would indeed produce a $1-1/e-\varepsilon$ approximation. Nevertheless, such a modification would result in a significant slowdown ($\times 3$ to $\times 27$ depending on the independence factor). In the §5 experiments, we assess the approximation quality relatively to the actual theoretical bound $1-1/e-\delta(\varepsilon)$. Finally, thanks to the following result, we adapt MACH_γ so that F_0-sketching is not needed.

Let MACH'_γ be the algorithm that replaces line 24 in Algorithm 1 with Algorithm 2. The selected guess is the biggest active guess, s, such that $|C'_s| \geq (1-\varepsilon)(1-1/e-\varepsilon)\lambda$. We thus conclude MACH'_γ is correct for $\gamma \geq \lfloor \frac{c}{3} k \log m \rfloor$.

▶ **Lemma 6.** *Let v be some guess in MACH'_γ and let C' be the final subsampled coverage associated with guess v. If*

$$\big|\, |C'| - p|C| \,\big| < \varepsilon v p; \text{ and } v > OPT; \text{ and } (1-\varepsilon)(1-1/e-\varepsilon)\lambda \leq |C'|\,,$$

then $|C| > (1-1/e-\delta(\varepsilon))\,OPT$.

▶ **Proposition 7.** *For $\gamma \geq \lfloor \frac{c}{3} k \log m \rfloor$, MACH'_γ finds a $1-1/e-\delta(\varepsilon)$ approximation of the Maximum-k-Coverage problem with probability at least $1-2e/m^{ck/6}$.*

Since the F_0-sketch is omitted, the time complexity is $T'_\gamma = \mathcal{O}(\varepsilon^{-1}\gamma m|\overline{\mathcal{S}}|\log k)$, while the space complexity is unchanged. With $\gamma = \lfloor \frac{c}{3} k \log m \rfloor$, MACH'_γ has a time complexity of $T'_{\lfloor \varepsilon^2\lambda/3 \rfloor} = \mathcal{O}(\varepsilon^{-1} k m |\overline{\mathcal{S}}| \log k \log m)$, which is at least ε^{-2} faster than expression (4).

4.2 Pairwise Independence

We now consider the smallest independence factor possible, $\gamma = 2$.

▶ **Proposition 8.** *Let h be a 2-independent hash function, and I, C, C', be defined accordingly. We have $\mathbb{P}\left(\, \big|\, |C'| - p|C| \,\big| \geq \varepsilon v p \,\right) \leq 2/(ck \log m)$.*

Substituting the bound of Proposition 8 into Proposition 7, we conclude:

▶ **Corollary 9.** *With probability at least $1-4/(ck\log m)$, MACH'_2 returns a $1-1/e-\delta(\varepsilon)$ approximation to the Maximum-k-Coverage problem.*

The decrease in probabilistic guarantee is compensated by a significant speed-up. The time complexity of MACH'_2 is $T'_2 = \mathcal{O}(\varepsilon^{-1} m |\overline{\mathcal{S}}| \log k)$, which grows only logarithmically in k.

5 Experimentation

In this section, we assess the performance of the MACH'_γ algorithm family on real-world datasets. We focus on four datasets, summarized in Table 2:

- *SocialNet*[7] represents a collection of individuals linked by a friendship relation.
- *UKUnion* [5] combines snapshots of webpages in the .uk domain taken over a 12-month period between June 2006 and May 2007.
- *Webbase* [6] and *Webdocs* [13] each represent a collection of interlinked websites.

Table 2 Real-world datasets. Hapax Legomena (HL) refers to the number of sets that contains an element which appears only in this set. The minimum set size and element frequency is 1.

Dataset	n $\times 10^6$	m $\times 10^6$	Set size Max	Med	Avg	Element frequency Max	Med	Avg	HL
SocialNet	65.0	37.6	3,615	12	48.10	4,223	6	27.80	24.0%
UKUnion	126.5	74.1	22,429	25	45.56	4,714,511	2	26.71	16.7%
Webbase	112.2	57.0	3,841	6	11.81	618,957	2	6.00	23.5%
Webdocs	5.3	1.7	71,472	98	177.20	1,429,525	1	56.93	21.2%

MACH'_γ is implemented[8] in C++20 and executed on *Spartan* the high performance computing system of The University of Melbourne. The CPU model is the Intel(R) Xeon(R) Gold 6254 CPU @ 3.10GHz with a maximum frequency of 4GHz. MACH'_γ naturally implies a parallel algorithm that consists of performing the computation related to each guess in parallel. Nonetheless, we do not implement an actual parallel algorithm as it would require substantial effort in order to fine tune. Also, compared to the original algorithm, this approach does not change the number of guesses. Therefore, the potential speed-up of a parallel implementation would be the same for our algorithm MACH'_γ and the original algorithm MACH_*.

Assessing coverage. With original independence factor $\gamma = \lceil 2\lambda \rceil$, MACH'_γ can still take tens of hours on the biggest datasets. We thus introduce a new variant of the algorithm, the *full sampling* variant, $\mathsf{MACH}'_{\text{fs}}$. *Full sampling* means there is no subsampling so $\mathsf{MACH}'_{\text{fs}}$ is a deterministic algorithm where $\mathbb{P}(E_I) = 1$. It means that $\mathsf{MACH}'_{\text{fs}}$ is fast and produces particularly good solutions (Figure 3). However, it has a space complexity of $\tilde{\mathcal{O}}(n)$ so $\mathsf{MACH}'_{\text{fs}}$ is just seen as a tool to assess the approximation quality of MACH'_2.

Setting c. To bound the failure probability of $\mathsf{MACH}'_{\lceil 2\lambda \rceil}$ and $\mathsf{MACH}'_{\lfloor \varepsilon^2 \lambda/3 \rfloor}$, we set $c \leftarrow 6$. With $c < 6$, $\mathsf{MACH}'_{\lfloor \varepsilon^2 \lambda/3 \rfloor}$ would be even more space efficient, while maintaining a high probability of success: $\mathbb{P}(E_I) \geq 1 - e/m^{ck/3}$, for $c \geq 1$, where m is expected to exceed several million. We therefore run $\mathsf{MACH}'_{\text{fs}}$ and MACH'_γ with $c = 1$.

Suite of experiments. Algorithms $\mathsf{MACH}'_{\text{fs}}$, $\mathsf{MACH}'_{\lceil 2\lambda \rceil}$, $\mathsf{MACH}'_{\lfloor \varepsilon^2 \lambda/3 \rfloor}$ and MACH'_2 are executed on the four datasets, for $\varepsilon \in \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}\}$ and $k \in \{1, 2, 4, 8, 16, 32, 64, 128, 256\}$. Although $\varepsilon = 1/2$ is out of the theoretical consideration, because $1 - 1/e - \delta(1/2) < 0$, it presents an opportunity to observe how the algorithm behaves outside its theoretical scope: $\varepsilon < 0.267$. For each Figure, in the main text, we only show the datasets representative of the variety of behaviors. The remaining components of each figure are in the Appendix.

[7] `https://snap.stanford.edu/data/com-Friendster.html`
[8] `https://github.com/caesiumCode/streaming-maximum-cover`

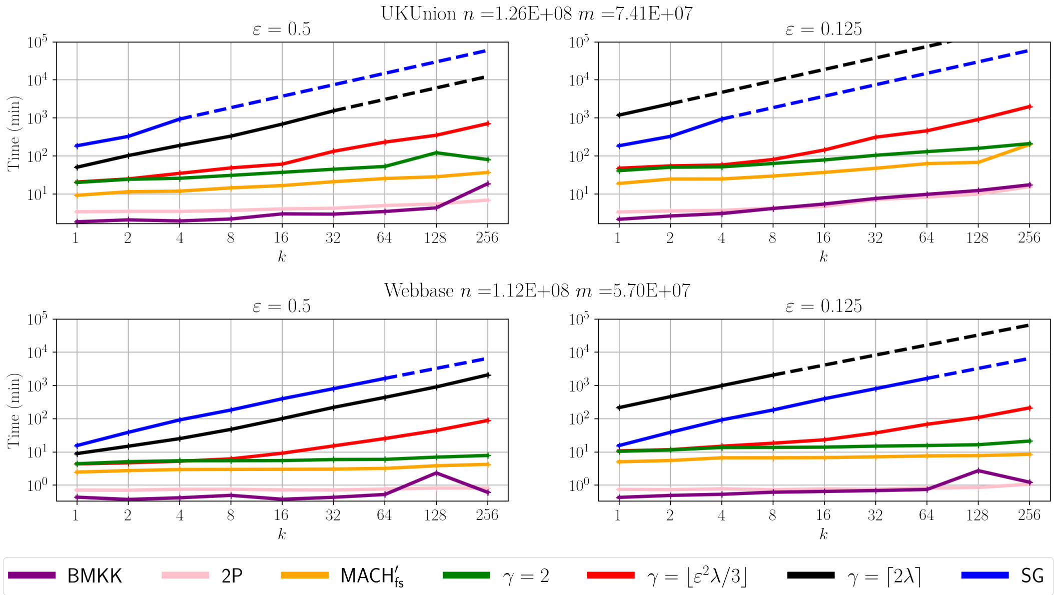

Figure 1 Running times of the algorithms, demonstrated on *Webbase*. Observe that for $\varepsilon = 1/8$, sticking with $\gamma = \lceil 2\lambda \rceil$ leads to a particularly slow algorithm. On the other hand, up to $k = 8$, both $\gamma = \lceil \varepsilon^2 \lambda/3 \rceil$ and $\gamma = 2$ are only 2–3 times slower than full sampling. For larger values of k, $\gamma = \lceil \varepsilon^2 \lambda/3 \rceil$ becomes 8–10 times slower. The missing values (dashed line) for $\gamma = \lceil 2\lambda \rceil$ and SG are extrapolated, they refer to a running time that exceeds the time limit of 48 hours.

Comparator algorithms (refer to Table 1). The goal is to assess the trade-off between time, space and approximation quality, therefore we aim to compare MACH'_γ with algorithms that perform relatively well in all three categories. For that reason, we implemented the comparator algorithms SG, BMKK, and 2P. The algorithm of Yu and Yuan [22] takes too much time and space, as it solves for all possible values of k at once. Also, as illustrated and explained in the Appendix (Figure 7), the $\tilde{O}(\varepsilon^{-d} m)$-space algorithms consume too much space in comparison to the $\tilde{O}(\varepsilon^{-d} n)$ space algorithms. We run Algorithm 2P instead of its $(1 - 1/e - \varepsilon)$-approximation cousin [18]: the latter is less effective than 2P, empirically, while consuming the same space and taking more passes.

5.1 Runtime Evaluation

Figure 1 demonstrates the time saved by reducing the independence factor. $\mathsf{MACH}'_{\lfloor \varepsilon^2\lambda/3 \rfloor}$ is consistently faster than $\mathsf{MACH}'_{\lceil 2\lambda \rceil}$ by an order of magnitude, while, as k increases, MACH'_2 widens its gap over $\mathsf{MACH}'_{\lfloor \varepsilon^2\lambda/3 \rfloor}$. In contrast to $\mathsf{MACH}'_{\mathsf{fs}}$, the time spent calculating hash function outputs for subsampling is clear. About half the running time of MACH'_2 is about subsampling, while this proportion easily exceeds 99% of the running time for $\mathsf{MACH}'_{\lceil 2\lambda \rceil}$. Considering comparators, BMKK and 2P are equally the fastest algorithms by a wide margin. Despite only one pass, SG is one of the slowest algorithms, along with $\mathsf{MACH}'_{\lceil 2\lambda \rceil}$. The precise running times can be consulted in Table 3.

5.2 Space Efficiency

To measure the space complexity of the different algorithms, we simply count the number of element instances stored by each algorithm. Figure 2 demonstrates how space efficient MACH'_γ is compared with alternatives, as predicted by the sublinear asymptotic bound: $\tilde{O}(\varepsilon^{-2} k)$, seemingly independent of the coverage size, in practice as well as theory.

Table 3 Summary of the running times in minutes of the algorithms (Figure 1). An empty cell means the time exceeds 48 hours (2880 minutes).

Dataset	*Webbase*				*UKUnion*			
k	4	16	64	256	4	16	64	256
				$\varepsilon = 0.5$				
$\mathsf{MACH}'_{\text{fs}}$	4.5	4.4	4.7	9.2	17.1	22.3	38.1	59.7
MACH'_2	8.5	8.8	9.3	10.8	35.7	48.3	77.2	134.5
$\mathsf{MACH}'_{\lfloor \varepsilon^2\lambda/3 \rfloor}$	8.0	14.5	38.3	135.9	40.8	90.3	338.3	1145.3
$\mathsf{MACH}'_{\lceil 2\lambda \rceil}$	143.4	588.6	2609.1	–	849.0	–	–	–
SG	92.1	399.3	1632.4	–	936.7	–	–	–
BMKK	0.5	0.5	0.6	0.8	2.9	3.8	5.5	11.6
2P	0.8	0.8	0.8	0.8	3.6	4.4	6.6	8.8
				$\varepsilon = 0.125$				
$\mathsf{MACH}'_{\text{fs}}$	6.7	6.8	7.6	8.5	24.9	37.1	63.3	201.3
MACH'_2	13.6	13.9	15.6	21.3	52.0	79.0	130.6	212.4
$\mathsf{MACH}'_{\lfloor \varepsilon^2\lambda/3 \rfloor}$	14.9	23.2	68.0	212.7	58.3	144.3	461.3	2002.5
$\mathsf{MACH}'_{\lceil 2\lambda \rceil}$	987.4	–	–	–	–	–	–	–
SG	92.1	399.3	1632.4	–	936.7	–	–	–
BMKK	0.5	0.6	0.7	1.2	3.1	5.5	9.9	17.5
2P	0.8	0.8	0.8	1.1	3.7	4.7	8.3	15.7

As stated earlier, the space complexity of $\mathsf{MACH}'_{\text{fs}}$, SG and BMKK scales linearly with the coverage size of the solution. So when MACH'_γ does not look so advantageous for *SocialNet* when $\varepsilon = 0.125$, it is simply because the coverage is almost as small as the space budget of MACH'_γ. The coverage of *UKUnion* is about 10 times bigger than *SocialNet*, but the space consumption is about the same as *SocialNet*.

5.3 Estimating Approximation Quality

The maximum set coverage problem is NP-Hard. Comparing the coverage size produced by MACH'_γ with the optimal solution is infeasible at the scale of our datasets. Since the greedy algorithm guarantees a $1 - 1/e$ approximation, and can be implemented, its coverage is our *reference*. Moreover, even if the optimal solution, OPT, is unknown, the $1 - 1/e - \delta(\varepsilon)$ approximation of MACH'_γ can be verified using the greedy algorithm thanks to the following implication: $|C|/|G| \geq \beta \implies |C|/\text{OPT} \geq \beta(1-1/e)$, where G returned by greedy and C an arbitrary coverage. In particular, if $|C|/|G| \geq 1-\delta(\varepsilon)/(1-1/e)$ then $|C|/\text{OPT} \geq 1-1/e-\delta(\varepsilon)$.

Figure 3 demonstrates that for theoretically *admissible* values of ε, MACH'_γ produces a coverage close to greedy coverage, and always within the $1 - \delta(\varepsilon)/(1 - 1/e)$ limit. Regardless of γ value, it remains very close to the coverage produced by $\mathsf{MACH}'_{\text{fs}}$. Even for pairwise independence, which is expected to produce worse solutions, there is no clear performance effectiveness difference compared higher independence. Additionally, even though MACH'_γ is a randomized algorithm, no coverage has been observed to beat greedy.

We observe some rare events ($< 1\%$) where the coverage is particularly small compared to $\mathsf{MACH}'_{\text{fs}}$. Investigating these cases reveals that such solutions typically contain fewer than k sets. If MACH'_γ does not select the right guess, it tends to select a guess slightly bigger than the right one, which increases the threshold, therefore it does not have enough opportunity to select k sets in $\mathcal{O}(\varepsilon^{-1})$ passes.

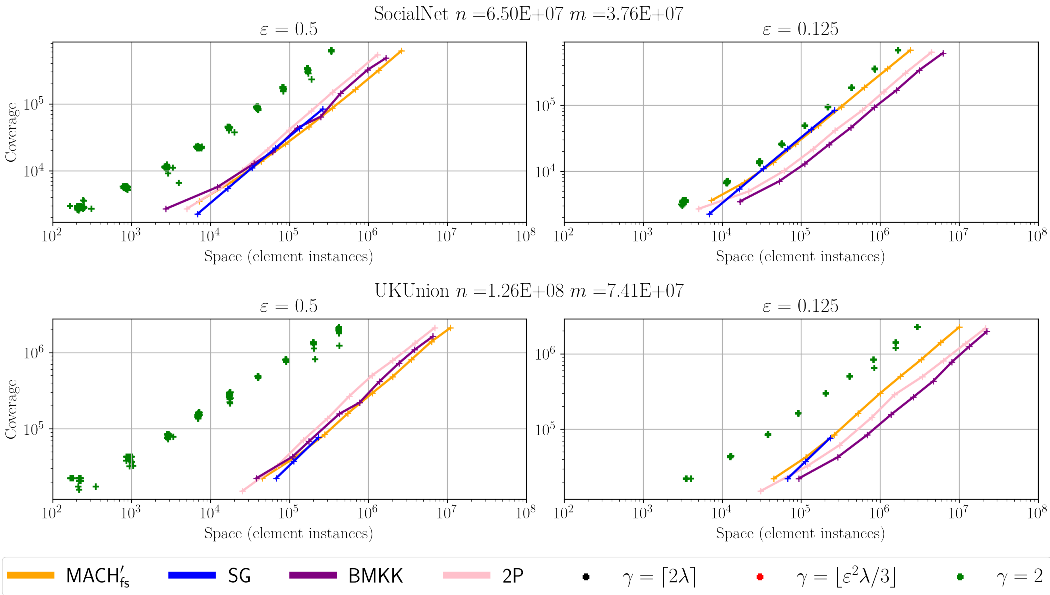

Figure 2 Coverage versus space, for $k \in \{1, 2, 4, 8, 16, 32, 64, 128, 256\}$. Every panel shows the advantage of the MACH′ family for max coverage in streams. For $\varepsilon = 1/2$, the space advantage is at least ten-fold. Interestingly, as ε decreases, the space advantage drops for some datasets, but the coverage does not improve significantly, suggesting that a *lightweight* MACH′ approach, i.e., smaller ε^{-1}, might be the most effective time-space-performance trade-off.

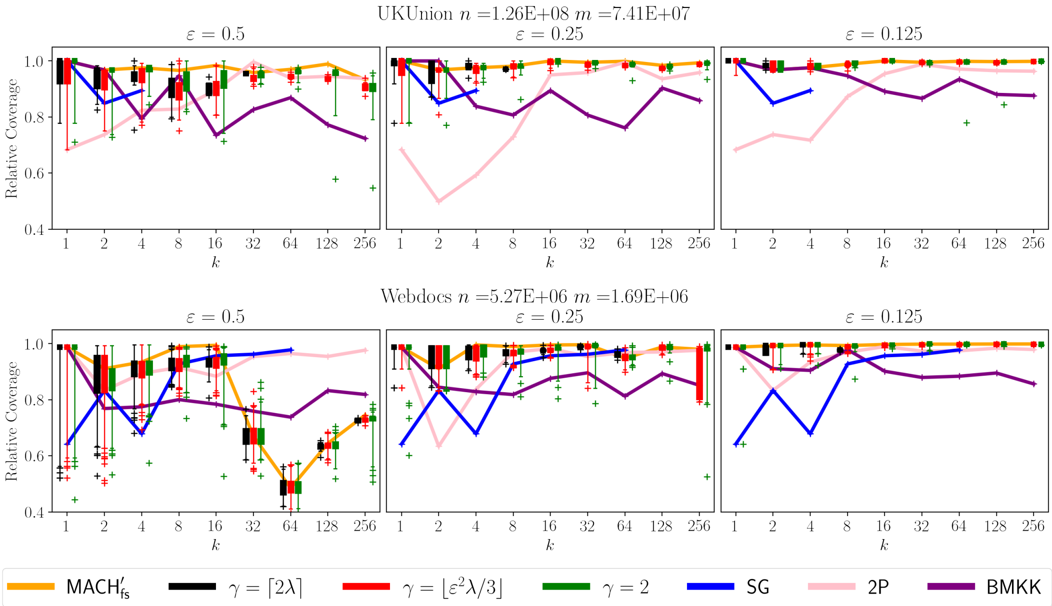

Figure 3 Coverage of the algorithms relative to greedy coverage. Since they are randomized, there are box plots of coverage produced by MACH$'_\gamma$. The box plots show the 1%, 25%, 75% and 99% quantiles, hence the points below and above the box plots are in the first and last 1%. For $\gamma = 2$ and $\gamma = \lfloor \varepsilon^2 \lambda/3 \rfloor$, each boxplot gathers 200 data points on average whereas for $\gamma = \lceil 2\lambda \rceil$, each boxplot gathers 90 data points on average. Observe that the MACH′ methods return excellent coverage except for values of k around 64 on the dataset *Webdocs* when $\varepsilon = 1/2$.

6 Conclusions

In this paper, we accelerate the sublinear-space approach to solving Maximum Coverage. The algorithm MACH_* of McGregor and Vu is hampered by a high-independence hash function. We generalize their approach to produce MACH_γ, so that $\mathsf{MACH}_* = \mathsf{MACH}_{\lceil 2\lambda \rceil}$ and then avoid F_0-sketches to obtain MACH'_γ. The space consumption is in $\tilde{\mathcal{O}}(\varepsilon^{-2}k)$ and the approximation factor is $1 - 1/e - \delta(\varepsilon)$.

For reasonable values of ε (≤ 0.25), our algorithm, MACH'_γ, maintains the space efficiency and approximation quality of $\mathsf{MACH}_* = \mathsf{MACH}_{\lceil 2\lambda \rceil}$. In experiments, it is several orders of magnitude faster. In practice, we find MACH'_2 presents the best trade-off between space complexity, time complexity and approximation quality. Since MACH'_2 is so efficient, we can run it several times with fresh randomness. This approach is more effective than executing MACH'_γ with a high independence factor. Although we avoided F_0-sketching in MACH'_γ, they could help compare independent instances of the fast MACH'_2.

We obtained several key results by carefully analyzing upper bounds on algorithm failure probability. We expect this idea accelerates other lower-space streaming algorithms.

References

1 Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. In *10th ITCS*, pages 1:1–1:19, 2018. doi:10.4230/LIPIcs.ITCS.2019.1.

2 Sepehr Assadi. Tight space-approximation tradeoff for the multi-pass streaming set cover problem. In *36th ACM PODS*, pages 321–335, 2017. doi:10.1145/3034786.3056116.

3 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: massive data summarization on the fly. In *20th ACM SIGKDD*, pages 671–680, August 2014. doi:10.1145/2623330.2623637.

4 MohammadHossein Bateni, Hossein Esfandiari, and Vahab Mirrokni. Almost optimal streaming algorithms for coverage problems. In *29th ACM SPAA*, pages 13–23, 2017. doi:10.1145/3087556.3087585.

5 Paolo Boldi, Massimo Santini, and Sebastiano Vigna. A large time-aware web graph. *SIGIR Forum*, 42(2):33–38, 2008. doi:10.1145/1480506.1480511.

6 Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *13th WWW*, pages 595–601, 2004. doi:10.1145/988672.988752.

7 Amit Chakrabarti and Anthony Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. In *27th ACM-SIAM SODA*, pages 1365–1373, 2016. doi:10.1137/1.9781611974331.ch94.

8 G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *IEEE Transactions on Knowledge and Data Engineering*, 15(3):529–540, 2003. doi:10.1109/tkde.2003.1198388.

9 Graham Cormode and Donatella Firmani. A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2013. doi:10.1007/s10619-013-7131-9.

10 Graham Cormode, Howard Karloff, and Anthony Wirth. Set cover algorithms for very large datasets. In *19th ACM CIKM*, pages 479–488, 2010. doi:10.1145/1871437.1871501.

11 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

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 *52nd ACM STOC*, pages 1363–1374, 2020. doi:10.1145/3357713.3384286.

13 Bart Goethals and Mohammed J Zaki. Fimi'03: Workshop on frequent itemset mining implementations. In *3rd IEEE Data Mining Workshop on Frequent Itemset Mining Implementations*, pages 1–13, 2003.

14 Tal Grossman and Avishai Wool. Computational experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, 101(1):81–92, 1997. doi:10.1016/s0377-2217(96)00161-0.

15 Piotr Indyk and Ali Vakilian. Tight trade-offs for the maximum k-coverage problem in the general streaming model. In *38th ACM PODS*, pages 200–217, 2019. doi:10.1145/3294052.3319691.

16 Ching Lih Lim, Alistair Moffat, and Anthony Wirth. Lazy and eager approaches for the set cover problem. In *37th ACSC*, pages 19–27, 2014. doi:10.5555/2667473.2667476.

17 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory of Computing Systems*, 63(7):1595–1619, 2018. doi:10.1007/s00224-018-9878-x.

18 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *35th ICML*, pages 3829–3838, 2018. URL: https://proceedings.mlr.press/v80/norouzi-fard18a.html.

19 Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):1–50, 2012. doi:10.1145/2220357.2220361.

20 Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *9th SDM*, pages 697–708, 2009. doi:10.1137/1.9781611972795.60.

21 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995. doi:10.1137/s089548019223872x.

22 Huiwen Yu and Dayu Yuan. Set coverage problems in a one-pass data stream. In *13th SDM*, pages 758–766, 2013. doi:10.1137/1.9781611972832.84.

A Definition of γ-independent hash functions family

▶ **Definition 10.** *The family of hash functions $\mathcal{H} \subset \{f : \mathcal{U} \to Y\}$ is γ-independent iff for every γ distinct keys $x_1, \dots, x_\gamma \in \mathcal{U}$ and γ values $y_1, \dots, y_\gamma \in Y$, if we draw f uniformly at random from $\mathcal{H}$, then the $f(x_i)$ are independent uniform random variables, and* $\mathbb{P}\left[\bigcap_{i=1}^{\gamma}(f(x_i) = y_i)\ \right] = 1/|Y|^\gamma$.

B Corollary 5 (proof)

▶ **Corollary 5.** *If $\gamma \le \lfloor \frac{c}{3} k \log m \rfloor$, with I, C, and C' defined accordingly, then:*

$$\mathbb{P}\left(\ \big|\ |C'| - p|C|\ \big| \ge \varepsilon vp\ \right) \le e^{-\lfloor \gamma/2 \rfloor}\,.$$

Proof. Consider I to be fixed, so the $\mathbb{1}_{i \in C}$ factor is just a constant, with no randomness.

$$\mathbb{E}(|C'|) = \sum_{i=1}^{n} \mathbb{1}_{i\in C}\mathbb{E}(\mathbb{1}_{h(i)=1}) = \sum_{i\in C} \mathbb{P}(h(i) = 1) = p|C|$$

Let $\delta = \varepsilon v/|C|$ and $\mu = \mathbb{E}(|C'|)$, then $\mathbb{P}\left(\ \big|\ |C'| - p|C|\ \big| \ge \varepsilon vp\ \right) = \mathbb{P}\left(\ \big|\ |C'| - p|C|\ \big| \ge \delta\mu\ \right)$. Now, recalling the definition (2) of λ, we verify the condition on the independence factor, γ:

$$\begin{aligned} \frac{c}{3} k \log m &= \frac{\varepsilon^2}{3}\lambda = \frac{\varepsilon}{3}\delta\mu \le e^{-1/3}\frac{\varepsilon}{2}\delta\mu && e^{-1/3} > 2/3 \\ &\le e^{-1/3}\min(1,\delta)\delta\mu && \delta \ge \varepsilon/2 \text{ and } \varepsilon/2 \le 1\,, \end{aligned}$$

where $|C| \le \mathrm{OPT} \le 2v$ gives us the condition $\delta \ge \varepsilon/2$. The condition $\varepsilon/2 \le 1$ is arbitrary but recall that we want a $1 - 1/e - \varepsilon$ approximation so $\varepsilon < 1 - 1/e \le 0.7$. Therefore, $\gamma \le \lfloor \frac{c}{3} k \log m \rfloor \le \lfloor e^{-1/3} \min(\delta, \delta^2)\mu \rfloor$ and Theorem 4 gives the desired inequality. ◀

C Lemma 6 (proof)

▶ **Lemma 6.** *Let v be a guess in* MACH$_\gamma$ *and let C' be the final subsampled coverage associated with guess v. If*

1. $\big|\ |C'| - p|C|\ \big| < \varepsilon vp$*; and*
2. $v \geq OPT$*; and*
3. $(1-\varepsilon)(1-1/e-\varepsilon)\lambda \leq |C'|$*;*

then $|C| > (1 - 1/e - \delta(\varepsilon))\,OPT$.

Proof. Assuming condition 3, we have,

$$
\begin{aligned}
|C'| - \varepsilon vp &\geq (1-\varepsilon)(1-1/e-\varepsilon)vp - \varepsilon vp & \\
|C|p &\geq (1-\varepsilon)(1-1/e-\varepsilon)vp - \varepsilon vp & \text{Condition 1}\\
|C| &\geq (1-\varepsilon)(1-1/e-\varepsilon)v - \varepsilon v & p > 0\\
|C| &\geq (1-1/e-\delta(\varepsilon))v & \text{where } \delta(x) = x(3-1/e-x)\\
|C| &\geq (1-1/e-\delta(\varepsilon))\,\mathrm{OPT} & \text{Condition 2.}
\end{aligned}
$$

◀

D Proposition 7 (proof)

▶ **Proposition 7.** *For $\gamma \geq \lfloor \frac{c}{3} k \log m \rfloor$,* MACH$'_\gamma$ *finds a $1 - 1/e - \delta(\varepsilon)$ approximation of the Maximum-k-Coverage problem with probability at least $1 - 2e/m^{ck/6}$.*

Proof. Let v_s be the selected guess accordingly to procedure 2 and v_* the right guess, i.e. $\mathrm{OPT}/2 \leq v_* \leq \mathrm{OPT}$. Also, we denote by I_s the solution associated with guess v_s.

- If $v_s = v_*$, we already saw that the $1 - 1/e - \delta(\varepsilon)$ approximation is guaranteed if the event $E_{I_s} = \big||C'_s| - p|C_s|\big| < \varepsilon v_s p$ is met.
- If $v_s > v_*$, then $v_s \geq \mathrm{OPT}$ because each guess is of the form $2^g||\mathcal{S}||_\infty$, so v_s must be at least twice as big as v_*. Therefore, Lemma 6 ensures the $1-1/e-\delta(\varepsilon)$ approximation if the event E_{I_s} is met, because procedure 2 always takes a guess for which $|C'_s| \geq (1-\varepsilon)(1-1/e-\varepsilon)\lambda$.

To conclude, MACH$'_\gamma$ finds a $1-1/e-\delta(\varepsilon)$ approximation of *Maximum-k-Coverage* if $v_s \geq v_*$ and E_{I_s}. Furthermore, a consequence of Corollary 9 in §2.3 [17] is that, for the right guess v_*, if $\big||C'_*| - p|C_*|\big| < \varepsilon v_* p$ then $|C'_*| \geq (1-\varepsilon)(1-1/e-\varepsilon)\lambda$, which makes the right guess a possible choice for procedure 2. Therefore, $E_{I_*} \Rightarrow v_s \geq v_*$. Consequently:

$$
\begin{aligned}
\mathbb{P}\left(\{v_s \geq v_*\} \cap E_{I_s}\right) &= 1 - \mathbb{P}\left(\{v_s < v_*\} \cup \overline{E_{I_s}}\right) \geq 1 - \mathbb{P}\left(\overline{E_{I_*}} \cup \overline{E_{I_s}}\right)\\
&\geq 1 - \mathbb{P}\left(\overline{E_{I_*}}\right) - \mathbb{P}\left(\overline{E_{I_s}}\right) \geq 1 - 2\frac{e}{m^{ck/6}}
\end{aligned}
$$

◀

E Proposition 8 (proof)

▶ **Proposition 8.** *Let h be a 2-independent hash function, and I, C, C', be defined accordingly. We have* $\mathbb{P}\left(\ \big|\ |C'| - p|C|\ \big| \geq \varepsilon vp\ \right) \leq 2/(ck\log m)$.

Proof. As before, $|C'|$ can be represented as $|C'| = \sum_{i=1}^{n} X_i$, where $X_i = \mathbb{1}_{i \in C} \mathbb{1}_{h(i)=1}$. Letting $\mathbb{V}$ stand for *variance*, we have:

$$\begin{aligned}
\mathbb{V}\big(|C'|\big) &= \mathbb{E}\left(|C'|^2\right) - \mathbb{E}\big(|C'|\big)^2 = \mathbb{E}\left(\left(\sum_{i=1}^{n} X_i\right)^2\right) - p^2|C|^2 \\
&= \mathbb{E}\left(\sum_{i=1}^{n} X_i + \sum_{i \neq j} X_i X_j\right) - p^2|C|^2 && X_i^2 = X_i \\
&= \mathbb{E}\big(|C'|\big) + \sum_{i \neq j} \mathbb{E}\left(X_i X_j\right) - p^2|C|^2 \\
&= p|C| + \sum_{i \neq j} \mathbb{E}\left(X_i\right) \mathbb{E}\left(X_j\right) - p^2|C|^2 && \text{pairwise independence} \\
&= p|C| + \sum_{i \neq j} \mathbb{1}_{i,j \in C} \mathbb{E}\left(\mathbb{1}_{h(i)=1}\right) \mathbb{E}\left(\mathbb{1}_{h(j)=1}\right) - p^2|C|^2 \\
&= p|C| + p^2\left(|C|^2 - |C|\right) - p^2|C|^2 = p(1-p)|C|
\end{aligned}$$

Including this value for the variance in Chebyshev's inequality, we have:

$$\begin{aligned}
\mathbb{P}\left(\ \big|\ |C'| - p|C|\ \big| \geq \varepsilon v p\ \right) &\leq \frac{\mathbb{V}\big(|C'|\big)}{(\varepsilon v p)^2} = \frac{p(1-p)|C|}{\varepsilon^2 v^2 p^2} \\
&\leq \frac{p\,\mathrm{OPT}}{\varepsilon^2 v^2 p^2} = \frac{\mathrm{OPT}}{\varepsilon^2 v^2 p} \\
&\leq \frac{2v}{\varepsilon^2 v^2 p} && \mathrm{OPT}/2 \leq v \\
&= \frac{2}{\varepsilon^2 \lambda} && p = \lambda / v \\
&= \frac{2}{\varepsilon^2 c \varepsilon^{-2} k \log m} = \frac{2}{c k \log m}\,.
\end{aligned}$$

◀

F Experimentation

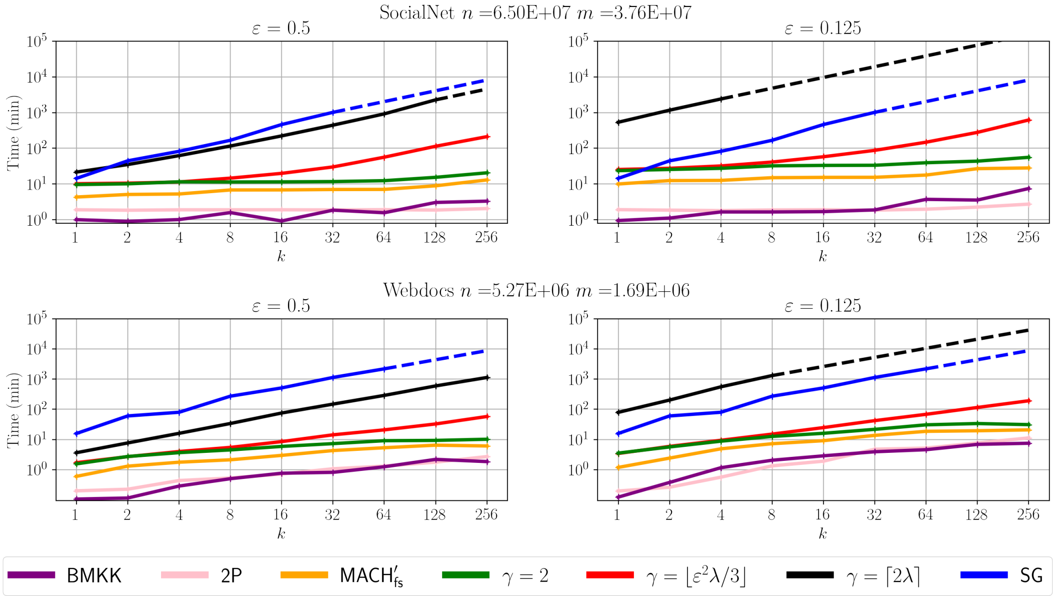

Figure 4 Running times of algorithms MACH$'_\gamma$, MACH$'_{\text{fs}}$, SG, and BMKK on datasets *SocialNet*, *UKUnion* and *Webdocs*.

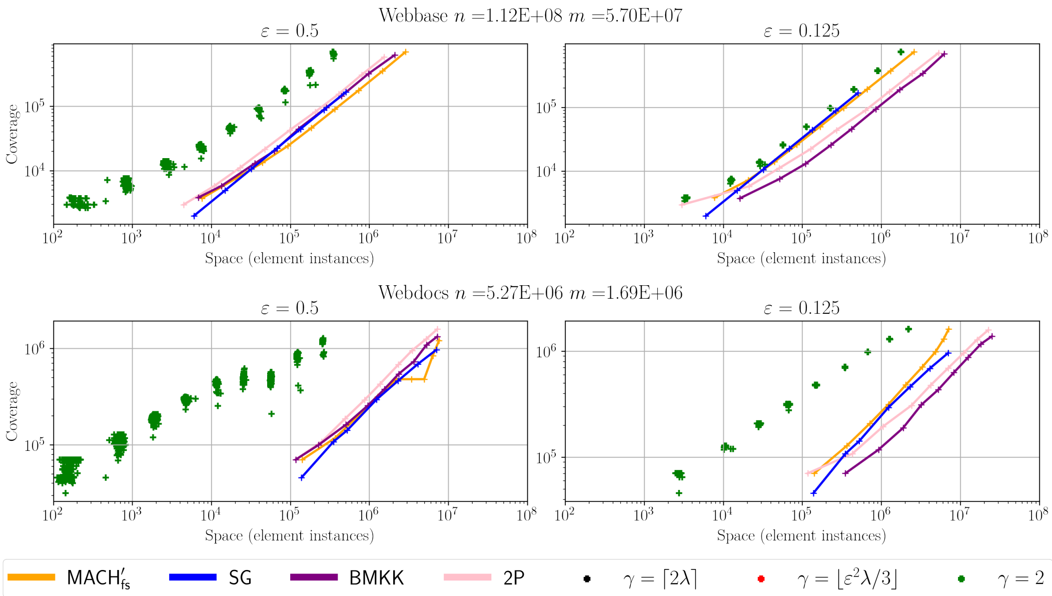

Figure 5 Coverage versus space, for $k \in \{1, 2, 4, 8, 16, 32, 64, 128, 256\}$ on datasets *Webbase* and *Webdocs*. The anomalies of MACH$'_{\text{fs}}$ and MACH$'_\gamma$ in *Webdocs* when $\varepsilon = 0.5$ coincide with the coverage quality drop in Figure 3.

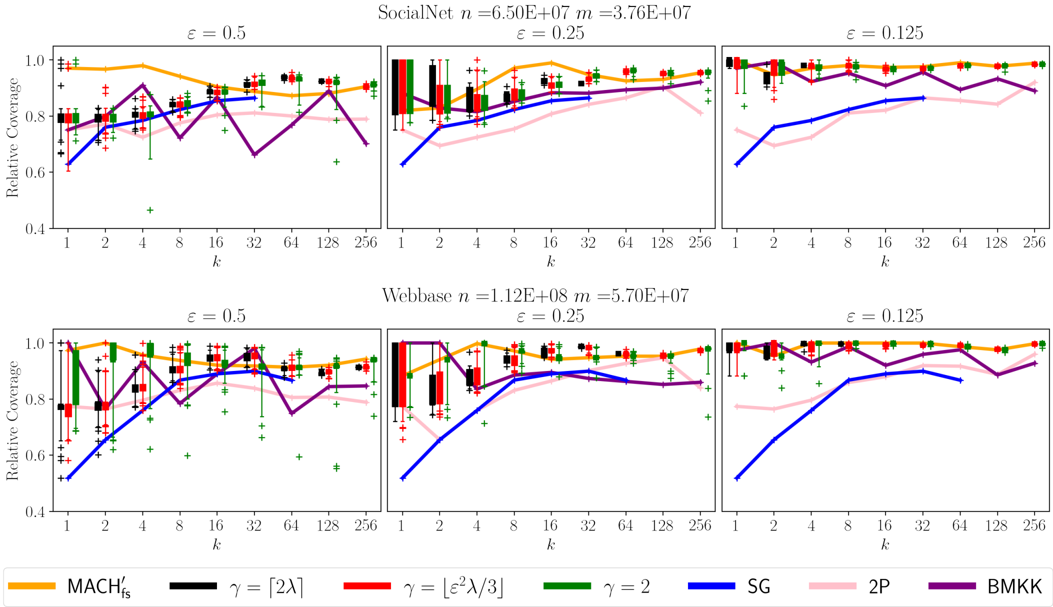

Figure 6 Box plot of coverage produced by MACH$'_\gamma$, SG and BMKK relative to the coverage produced by the greedy algorithm for the datasets *SocialNet* and *Webbase*.

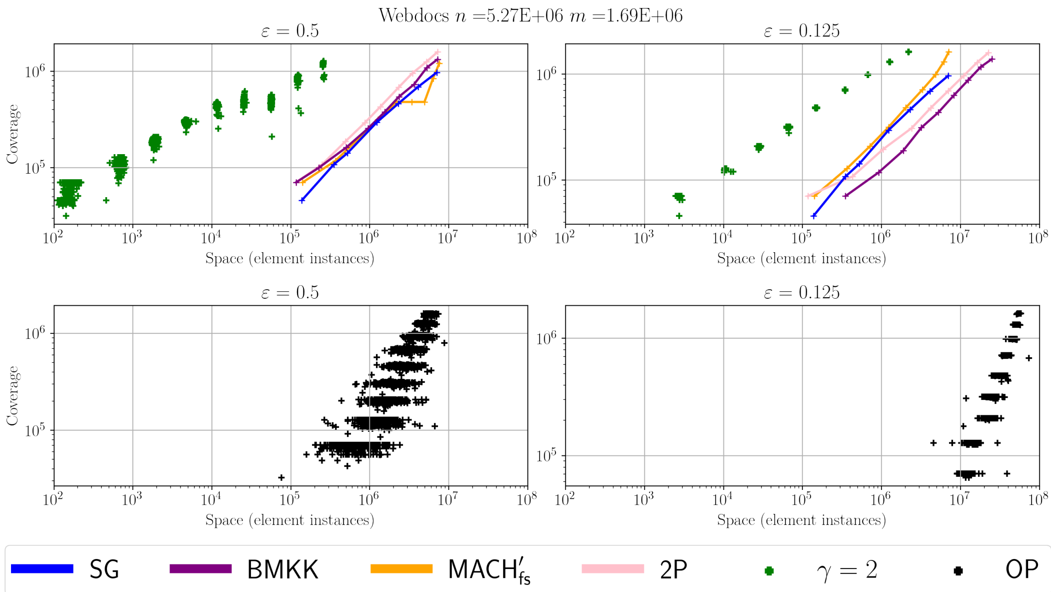

Figure 7 Coverage versus space, the $\tilde{\mathcal{O}}(\varepsilon^{-2}m)$ space algorithm OP [17] consumes systematically more space than all the other $\tilde{\mathcal{O}}(\varepsilon^{-d}n)$ space alternatives. This is because the $\tilde{\mathcal{O}}(\varepsilon^{-d}n)$ space algorithms actually scale linearly with respect to the returned coverage size with a hidden constant close to one. On the other hand, the $\tilde{\mathcal{O}}(\varepsilon^{-d}m)$ space algorithms, such as OP, have precisely a $\tilde{\Theta}(\varepsilon^{-d}m)$ space complexity with a much bigger hidden constant, storing a fraction of each set.