Exploration with Limited Memory: Streaming Algorithms for Coin Tossing, Noisy Comparisons, and Multi-armed Bandits*

Sepehr Assadi
sepehr.assadi@rutgers.edu
Rutgers University – New Brunswick
Piscataway, NJ, USA

Chen Wang
chen.wang.cs@rutgers.edu
Rutgers University – New Brunswick
Piscataway, NJ, USA

ABSTRACT

Consider the following abstract coin tossing problem: Given a set of n coins with unknown biases, find the most biased coin using a minimal number of coin tosses. This is a common abstraction of various exploration problems in theoretical computer science and machine learning and has been studied extensively over the years. In particular, algorithms with optimal *sample complexity* (number of coin tosses) have been known for this problem for quite some time.

Motivated by applications to processing massive datasets, we study the *space complexity* of solving this problem with optimal number of coin tosses in the *streaming* model. In this model, the coins are arriving one by one and the algorithm is only allowed to store a limited number of coins at any point – any coin not present in the memory is lost and can no longer be tossed or compared to arriving coins. Prior algorithms for the coin tossing problem with optimal sample complexity are based on iterative elimination of coins which inherently require storing all the coins, leading to memory-inefficient streaming algorithms.

We remedy this state-of-affairs by presenting a series of improved streaming algorithms for this problem: we start with a simple algorithm which require storing only $O(\log n)$ coins and then iteratively refine it further and further, leading to algorithms with $O(\log\log(n))$ memory, $O(\log^*(n))$ memory, and finally a one *that only stores a single extra coin in memory* – the same exact space needed to just store the best coin throughout the stream.

Furthermore, we extend our algorithms to the problem of finding the k most biased coins as well as other exploration problems such as finding top-k elements using noisy comparisons or finding an ε-best arm in stochastic multi-armed bandits, and obtain efficient streaming algorithms for these problems.

*Full version available on: https://arxiv.org/pdf/2004.04666.pdf [5]

STOC '20, June 22–26, 2020, Chicago, IL, USA

ACM ISBN 978-1-4503-6979-4/20/06…$15.00
https://doi.org/10.1145/3357713.3384341

CCS CONCEPTS

• **Theory of computation → Streaming, sublinear and near linear time algorithms**; • **Computing methodologies → Machine learning**; • **Mathematics of computing**;

KEYWORDS

Streaming Algorithms, Pure Exploration, Memory-efficient Algorithms, Noisy Comparison, Multi-Armed Bandits

ACM Reference Format:

Sepehr Assadi and Chen Wang. 2020. Exploration with Limited Memory: Streaming Algorithms for Coin Tossing, Noisy Comparisons, and Multi-armed Bandits. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC '20), June 22–26, 2020, Chicago, IL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3357713.3384341

1 INTRODUCTION

Suppose you are given n coins with unknown biases; how many *samples* (coin tosses) are needed to find the most biased coin with a large (constant) probability of success? This basic problem captures the essence of various *(pure) exploration* problems in theoretical computer science and machine learning in which the general goal is to find a best option among a set of alternatives using a minimal number of stochastic/noisy trials. Examples include rank aggregation with noisy comparisons (e.g. [9, 12, 19, 20, 22, 24, 27, 47, 47]), best arm identification in multi-armed bandits (e.g. [6, 13, 16, 26, 33, 34, 36, 37, 41]), or computing with noisy decision trees (e.g. [27, 29, 44, 45]). These problems in turn have a wide range of applications in medical trials [46], networking [48, 50], web search [25], crowdsourcing [18, 51], and display advertising [2], among others.

This coin tossing problem admits a natural solution: sample/toss each coin "enough" number of times so that the empirical bias of each coin "closely" matches its true bias; then find the coin with the most empirical bias. Assuming there is some *constant known gap* between the bias of the most and the second most biased coins, a simple argument suggests that tossing each coin $O(\log n)$ times is enough for this purpose, leading to an algorithm with $O(n\log n)$ coin tosses overall.

It turns out that one can beat this natural approach and solve the problem with $O(n)$ samples [26] (see also [27]) which is the (asymptotically) optimal *sample complexity* of this problem [41]. Sample-optimal algorithms for this problem has since been studied extensively in various directions: finding multiple coins (e.g. [34, 35]),

with combinatorial constraints (e.g. [15, 17]), instance-optimal algorithms (e.g. [22, 33]), fixed-budget algorithms (e.g. [10, 11, 13]), limited adaptivity algorithms (e.g. [1, 22, 29]), or collaborative learning algorithms (e.g. [8, 32, 49]), to mention a few.

Alas, the sample-efficiency of these algorithms comes at a certain cost: unlike the basic approach that processes the coins "on the fly" by storing the current candidate coin, these more complicated algorithms need to store all coins and revisit them frequently before making a decision. As such, these solutions can be prohibitively expensive in their memory requirement in applications with a massive number of coins/options (including several of above examples). In such scenarios, the *space complexity*, in addition to the sample complexity, plays a major role in the efficiency of algorithms.

The *streaming model* of computation, pioneered by [4, 28, 31], precisely captures these scenarios. In this model, the coins are arriving one by one and the algorithm is only allowed to store a limited number of coins at any point – any coin not present in the memory is lost and can no longer be tossed or compared to arriving coins. We refer to the maximum number of coins stored by the algorithm at any point during the stream as the space complexity or memory cost of the algorithm (see Section 2 for details). We can now ask the following fundamental question:

> *What is the memory cost of achieving (asymptotically) optimal sample complexity for the coin tossing problem in the streaming model?*

Our main (conceptual) finding in this paper is that, surprisingly, there is almost *no tradeoff* between sample-efficiency and space-efficiency for coin tossing: one can achieve the sharpest possible bound on the space complexity, namely a memory of *a single extra coin*, without having to settle for an asymptotically sub-optimal sample complexity!

We further build on this result to design streaming algorithms for finding multiple coins with largest biases and for other related problems such as partitioning totally ordered elements using noisy comparisons or finding approximate best arms in stochastic multi-armed bandits. The extension of our coin tossing results to noisy comparisons is particularly interesting as there is no black-box reduction between the two models and indeed these models are often considered conceptually related but disjoint technique-wise (see, e.g. [9, 20, 22]).

1.1 Our Contributions on the Coin Tossing Problem

Most Biased Coin. Our first main result is a complete resolution of the aforementioned question:

> Result 1. *There exists a streaming algorithm that achieves the (asymptotically) optimal sample complexity for the coin tossing problem by storing only* ***a single extra coin*** *in its memory.*

We formalize Result 1 in Theorem 1. We emphasize that in Result 1 and throughout the paper, we assume the algorithm knows the gap between the bias of the most and the second most biased coins. Extending our results to unknown gaps is an interesting open question.

An interesting byproduct of using just a single-coin memory in Result 1 is that the algorithm necessarily maintains the most biased coin as its only candidate once this coin is observed in the stream, namely, it is also an *online* algorithm (this corresponds to the notion of *streaming online algorithms* proposed in [40].)

En route to proving Result 1, we design a series of streaming algorithms with optimal sample complexity for coin tossing (see the Appendix A in the full version [5]). We start with a simple algorithm that uses $O(\log n)$ memory by giving a streaming friendly implementation of the median-elimination algorithm of [26] using the "merge-and-reduce" technique from the streaming literature (see, e.g. [3, 30]). We then show that one can further improve the memory down to $O(\log\log n)$ coins by designing a variant of merge-and-reduce tailored directly to the coin tossing problem. This adaptation in turn allows us to use the more recent aggressive-elimination algorithm of [1] in place of the original median-elimination and reduce the space down to $O(\log^*(n))$ coins[1]. The final leap from $O(\log^*(n))$ memory algorithm to our single-coin memory algorithm however is the key step as explained below.

The memory bound of our intermediate streaming algorithms is heavily tailored to the number of elimination rounds of base algorithms in [1, 26] and it is known that $\Theta(\log^*(n))$ bound on number of elimination rounds is *tight* [1, 29]. As such, to obtain our final algorithm, we almost entirely forego the elimination approach and devise a new *budgeting strategy* for the problem: we maintain a candidate coin, called the "king", throughout the stream and assign it a certain budget which is increased per each new arriving coin and decreased whenever we toss any coin. Each arriving coin then "challenges" the king by tossing both the king and arriving coin, according to a carefully chosen rule, until either king wins against the new coin (by having a higher empirical bias at any of these challenges) or the budget of the king is depleted in which case we replace the king with the new coin and restart the process with this new king on the remainder of the stream.

This budgeting allows us to use a basic amortized analysis and argue that the total number of coin tosses by the algorithm is still $O(n)$ (albeit with a much more chaotic pattern of samples per coin compared to elimination-based algorithms). The key challenge is however to ensure that once the most biased coin becomes the king, it will not exhaust its budget throughout the remaining length of the stream which can be $\Theta(n)$-long. This requires proving that the random variable corresponding to the remaining budget of the king does not have any significant deviation from its expectation *throughout* the entire length of the stream and not only at *any fixed* point. This is similar-in-spirit to the fact that a length n symmetric (± 1)-random walk on a line does not deviate from the $\Theta(\sqrt{n})$ bound implied by the variance not only at the end, but throughout the entire walk (the proofs are however different since our version of "random walk" includes unbounded step sizes and so we first prove that these step sizes form a sub-exponential distribution and then use Bernstein's inequality to prove the desired concentration bound).

[1] None of these algorithms follow as a black-box from prior work and several new ingredients are still needed to make these parts work in the streaming model which can be of their own independent interest. To this end, these algorithms and their analysis are presented in the full version [5].

Top-k Most Biased Coins. A standard generalization of the coin problem we discussed so far is to find the top-k most biased coins assuming a gap between the bias of the k-th and $(k+1)$-th most biased coin. This problem has also been studied extensively in the literature and it is known that the (asymptotically) optimal sample complexity for this problem is $\Theta(n \log k)$ [34, 35]. We show that this optimal sample complexity can be achieved by memory-efficient streaming algorithms.

Result 2. *There exists a streaming algorithm that achieves the (asymptotically) optimal sample complexity for finding the top-k most biased coins by storing only $O(k)$ coins in the memory.*

We formalize Result 2 in Theorem 2. It is clear that any streaming algorithm for this problem requires memory of k coins to simply store the answer. As such, Result 2 implies that one can simultaneously achieve the *asymptotic* optimal memory and sample complexity for this problem.

The starting point of this algorithm is our budgeting approach in Result 1. However, there are two main challenges that need to be addressed: (1) we now need to maintain k "kings" but can no longer compare each arriving coin with (or assign a unit of budget to) every king (otherwise, there will be $\Omega(nk)$ coin tosses); more importantly (2) we need to collect all the top-k coins and still cannot guarantee any suitable (probabilistic) outcome while comparing any of these two coins to each other (as there may not be any gap between their biases in general). We elaborate on these challenges and how we address them in the high level overview of our algorithm in Section 4 and only mention here that addressing these challenges turn out to be a highly non-trivial task and in fact our algorithm in Result 2 is the main *technical* contribution of our work.

1.2 Application to Noisy Comparison Model

An interesting application of our results is to the following noisy comparison problem: we have a collection of n elements with an *unknown total order* and we can compare any two element i and j according to a *noisy* version of this ordering: when comparing i, j, with probability $2/3$ we receive the true answer whether $i < j$ or $j < i$, and with the remaining probability, the answer is arbitrarily. The goal is to partition the input into the set of k largest element and $(n-k)$ remaining smaller elements. This problem, often referred to as the *partition* problem, has received a burst of interest in recent years (see, e.g. [9, 19, 20, 22] and references therein). The streaming version of this problem, when the elements are arriving one by one in the stream and only the elements stored in the memory can be compared, is equally well-motivated (see [9] for related applications).

It is easy to spot a fundamental difference between the partition problem and coin tossing: the first one uses *ordinal* information between the elements while the latter concerns *cardinal* information. Due to this difference, the algorithms in one model do not carry over to another and the research on these two problems has been mostly disjoint (see, e.g. [9, 22] – see also [1] that gives a black-box reduction from coin tossing to a *different* noisy model of comparison and [22] that shows this, or any other, reduction cannot work in the model studied in our paper).

Interestingly, our algorithms in Result 1 and Result 2 operate by only comparing empirical biases of coins directly with each other (through the notion of "challenging" described above), which is an ordinal information. Rather more formally, our algorithms work even if instead of sampling the coins and observing their empirical biases, they can sample two coins and observe which one has the higher empirical bias. Owing to this property, we can indeed extend our algorithms in these results to the partition problem in the noisy model and obtain the following result.

Result 3. *There exists a streaming algorithm for the partition problem that uses $O(n \log k)$ noisy comparisons and a memory of $O(k)$ elements (the memory is a single extra element when $k = 1$).*

Result 3 is formalized in Theorem 3, presented in Section 5. Considering that the (asymptotically) optimal number of samples for the partition problem is $O(n \log k)$ [22], Result 3 achieves the asymptotically optimal sample complexity and space complexity simultaneously.

1.3 Application to Stochastic Multi-Armed Bandits

The ε-best arm identification (or PAC-learning) in the stochastic multi-armed bandit (MAB) games is defined as follows: we have a collection of n arms with unknown reward distributions in $[0, 1]$; the algorithm can pull (sample) each arm and receive a reward from the corresponding distribution. The goal is to, given a parameter $\varepsilon \in (0, 1)$, find any arm with expected reward at most ε less that the expected reward of the best arm, referred to as an *ε-best arm*. This problem is a (pure) exploration variant of the more general *regret minimization* problem in MABs introduced more than half a century ago [46] and has been studied extensively on its own (see, e.g. [6, 13, 16, 26, 33–37, 41] and references therein). Again, the streaming model for this problem, in which the arms are arriving one by one and can only be pulled if they are stored explicitly in the memory, is highly motivated; see, e.g., the recent work of [14, 39] on a related model to streaming and the classical work of [23] (we will elaborate on the connection between our work and the first two below).

It is easy to see that the coin tossing problem is a special case of this problem when the reward distributions are Bernoulli and more importantly, there is a *gap* of ε between the expected reward of the best arm and any other arm (making the ε-best arm unique). In general, these differences do not matter much and most algorithms for the coin tossing problem appear to extend directly to the ε-best arm problem as well. Unfortunately however, this is *not* the case for our algorithm in Result 1 (the brief intuition is that our algorithm only considers ordinal information between the empirical biases and a set of arms with gradually decreasing expected reward can "fool" the algorithm – we discuss this in detail in Section 6). Nevertheless, we are still able to extend our $O(\log^*(n))$ memory algorithm for coin tossing to this problem and prove the following result.

Result 4. *There exists a streaming algorithm for ε-best arm identification in stochastic multi-armed bandits that uses $O(n/\varepsilon^2)$ arm pulls and a memory of $O(\log^*(n))$ arms.*

Result 4 is formalized in Theorem 4, presented in Section 6. The sample complexity of this algorithm is asymptotically optimal [41] but its memory is within a *non-constant* (albeit extremely small) factor of the (best known) bounds; closing this gap remains a fascinating open problem.

We conclude this section by comparing our work with two very recent results of [14, 39]. Both papers design algorithms with a memory of only $O(1)$ arms for *regret minimization* in multi-armed bandits. Under such a setup, the algorithms should solve the problems of *exploration* and *exploitation* simultaneously and the exploration in their algorithms will pay an $O(\log(T))$ factor where T is the time horizon. This bound is not directly comparable with ours, and under the pure exploration scenario our algorithm will have asymptotically better sample efficiency. More importantly, since both of the papers adopted the strategy of confidence-bound estimation, in the context of streaming algorithms, these algorithms require making *multiple passes* over the input which may not be desirable in many settings (the algorithm of [14] additionally requires randomly permuting the arms which is infeasible unless one makes the random-order arrival assumption). It will be interesting to see if using our Result 4 in these algorithms can help with the performance.

2 PROBLEM DEFINITION: STREAMING COIN TOSSING

In the coin tossing problem that we study, there is a collection of n coins $\{\text{coin}_i\}_{i=1}^n$ with unknown biases $\{p_i\}_{i=1}^n$ and our goal is to identify the most biased coin, denoted by coin^*, via tosses of the coins. We refer to the number of coin tosses by the algorithm as its *sample complexity*. An important parameter that governs the sample complexity of the algorithms is the *gap parameter* Δ which denotes the difference between the bias of the most and the second most biased coins. We assume $\Delta > 0$ and is given to the algorithm – both assumptions are common in the literature [21, 26, 34, 47]. Indeed, the first assumption can be easily lifted by simply re-defining this value to be the gap between bias of the most biased coin and the next *distinct* bias. As for the second assumption, in both applications of our results, this parameter corresponds to the standard *input* parameters of the problem, namely the noise factor γ and the approximation factor ε.

We study this problem in the streaming model: The coins are arriving one by one in a stream and the algorithm needs to store each coin explicitly if it wants to toss it at some later point in the stream as well. In other words, the algorithm only has access to a coin if this is the current coin arriving in the stream, or the coin is currently stored in the memory of the algorithm. Moreover, once a coin is no longer in the memory (because it was either not stored in the first place or was later replaced by another coin), the algorithm has no further access to this coin (i.e., can neither toss it nor bring it back to the memory). We refer to the maximum number of coins stored by the algorithm at any point during the stream as the *space complexity* of the algorithm.

Remark 2.1. We stated the space complexity of the streaming algorithms in terms of number of stored arms and ignored the other information stored by them. This is the standard definition for streaming problems that assume *oracle* access to input (the coin tossing oracle for our purpose) such as streaming algorithms for submodular optimization (see, e.g. [7, 38, 42]). *All* our algorithms only require to store additional $\Theta(\log n + \log(1/\varepsilon))$ bits ($O(1)$ words of space in the word-RAM model) per each coin *in their memory*. We also remark that our $O(\log^*(n))$ space algorithm appears to be even implementable with only $\Theta(\log\log n + \log(1/\varepsilon))$ bit overhead per each memory coin by using the classical noisy counter of [43]; however, we do not pursue this direction in this paper.

3 MOST BIASED COIN: A SINGLE-COIN MEMORY ALGORITHM

We describe our main algorithm for the most biased coin problem in this section.

Theorem 1 (Formalization of Result 1). *There exists a streaming algorithm that given n coins arriving in a stream with the gap parameter Δ and confidence parameter δ, finds the most biased coin with probability at least $1-\delta$ using $O(\frac{n}{\Delta^2}\cdot\log(1/\delta))$ coin tosses and a memory of a single coin.*

Note that the sample complexity of our algorithm in Theorem 1 is asymptotically optimal in all three parameters and its space is minimum possible. We start with a high level overview of our algorithm, followed by its description, and then its analysis.

3.1 High Level Overview

The high level strategy of our algorithm is quite intuitive: The algorithm maintains a *single* coin in its memory, referred to as king. The goal is to ensure that at the end of the stream king is the most biased coin. Once a new coin arrives in the stream, we toss both the king and the new coin a certain number of times and based on the empirical bias, we may decide to overthrow the king and let the arriving coin become the new king. The challenge is of course to implement this intuitive strategy without using a large number of coin tosses.

In the intermediate algorithms, we introduced several multi-level challenge rules to ensure the number of overall coin tosses is small. Stemming from the same idea, a naive thought to address the single-coin challenge is to introduce another variation to the multi-level challenge with some fixed rules at each level. Unfortunately, this strategy can be proven impossible by the round lower bound in [1]. The original lower bound stated that for any round-based algorithm, finding the most biased coin necessarily takes $O(\log^*(n))$ rounds. As such, we can perform a simple reduction from 'rounds' to 'levels' and conclude such scheme will not work with only a single coin memory.

Indeed, a key step in ensuring the sample efficiency is a *lazy* challenging rule (as opposed to the fixed rules at each level) implemented in multiple levels: to compare king and the newly arrived coin, we first toss both coins a certain constant number of times; if the empirical bias of king is already larger than that of coin, we consider king the winner and move on; otherwise, we go to the next level and repeat this process with a larger number of coin tosses, and continue the same way – we only overthrow the king if it loses

to coin for a "large" number of times (we elaborate more on this below). We choose the number of samples in each level to ensure that the following two properties: (1) when the best coin arrives in the stream, it has a large probability of winning against any king at this point (no matter the budget of the king), and (2) when king becomes the best coin, it has a small probability of losing to *any* coin afterwards.

The approach above allows us to argue that with large probability, king is equal to the best coin at the end of the stream. However, it is still not enough to ensure the sample efficiency of the algorithm, because the lazy challenging rule allows for a large number of coin tosses per challenge (this is particularly problematic when king is *not* the most biased coin). We address this using an amortized analysis by allocating certain *budget* to the king: each king starts with some fixed (constant) budget and any new coin that arrives in the stream will increase the budget of king by some fixed (constant) number; the budget is reduced by one whenever we sample the king and its challenger. This way, we will simply overthrow the king once it has exhausted its *entire* budget accumulated so far. In that case we let the current challenger become the new king. The budget is then restarted for the new king and we continue as before.

Introduction of this budget ensures the sample efficiency of the algorithm (deterministically). However, we now need to make sure that the most biased coin will not exhaust its budget as the king and get overthrown. The lazy challenging rule we defined can be used to ensure that once the best coin becomes king, any remaining coin in the stream can only challenge the king *in expectation* with $O(1)$ samples, hence, by the time we visit the m-th next coin, we have used only $O(m)$ coin tosses *in expectation*, which fits the budget for king. But the worry is that *during* a $\Theta(n)$-length stream, there will be times that for which this random variable (the budget used) takes values $\gg O(m)$ (specially consider the *unboundedness* of tosses per each trial which is necessary to ensure correctness). It turns out however this cannot happen and we can prove that with high (constant) probability, throughout the *entire stream*, the number of times king is challenged is linear in the number of challengers. In order to do this, we need to ensure that our challenging rule is 'conservative' enough (the exact opposite of our $O(\log^*(n))$ space algorithm which utilizes a tower number-based increment of coin tosses per level) so that even though coin tosses per each challenge may be unbounded, they still form a sub-exponential distribution and hence we can apply Bernstein's inequality to prove the desired concentration bound.

3.2 The Algorithm: Game-Of-Coins

We now present our algorithm Game-Of-Coins. The input to the algorithm is the set of n coins $\{\mathsf{coin}_i\}_{i=1}^n$ arriving in an arbitrary order in a stream, the gap parameter $\Delta > 0$, and the confidence parameter $\delta \in (0,1)$ (the algorithm does *not* need to know the value of n in advance). Let us first set up the following parameters:

$$\{r_\ell\}_{\ell=1}^{\infty}: \quad r_\ell = 3^\ell;$$

(multiplicative factor of samples per each level of the challenge)

$$\{s_\ell\}_{\ell=1}^{\infty}: \quad s_\ell := \frac{4}{\Delta^2} \cdot \ln(1/\delta) \cdot r_\ell;$$

(the number of samples per each level of the challenge)

$$b := \frac{4}{\Delta^2} \cdot C \cdot \ln(1/\delta) + s_1.$$

(the budget given to the king per each new coin)

($C > 0$ is a constant to be determined later)

We are now ready to present the algorithm:

Algorithm 1: Game-Of-Coins

(1) Let king be the first available coin and set its budget $\Phi := \Phi(\mathsf{king}) = 0$.
(2) For each arriving coin_i in the stream do:
  (*a*) Increase the budget $\Phi(\mathsf{king})$ by b.
  (*b*) **Challenge subroutine:** For level $\ell = 1$ to $+\infty$ do:
    (i) If $\Phi(\mathsf{king}) < s_\ell$: we declare king defeated and go to Line (1).
    (ii) Otherwise, we decrease $\Phi(\mathsf{king})$ by s_ℓ and toss both king and coin_i for s_ℓ times.
    (iii) Let $\widehat{p}_{\mathsf{king}}$ and $\widehat{p}_i$ denote the empirical biases of king and coin_i in this trial.
    (iv) If $\widehat{p}_{\mathsf{king}} > \widehat{p}_i$, we declare king winner and go to the next coin in the stream; otherwise, we go to the next level of the challenge (increment ℓ by one).
(3) Return king as the best coin in the stream.

This concludes the description of our algorithm. The sample complexity of this algorithm can be bounded easily using an amortized analysis.

Claim 3.1. *The total number of coin tosses by the algorithm is at most* $4n \cdot b = O(\frac{n}{\Delta^2} \cdot \log(1/\delta))$.

Proof. The proof is a straightforward amortized analysis. Each arriving coin in the stream can increase the budget by b and each time we make a new king we allocate another b budget to it so over all we increase the budget by at most $2n \cdot b$ in total. On the other hand, each unit of budget is responsible for two coin tosses (for the king and its challenger) and so the total number of coin tosses is at most $4n \cdot b$ implying the claim as $b = O(\frac{\ln(1/\delta)}{\Delta^2})$. ∎

We prove the correctness of the algorithm in the next subsection.

3.3 The Analysis

The analysis consists of the following two main parts. Firstly, when we visit the most biased coin in the stream, it will defeat the king with a large probability and become the next king itself.

Lemma 3.2. *The probability that the most biased coin does* not *defeat the* king *is at most* $(\delta/2)$.

Secondly, after the most biased coin become the king, it will remain the king for the remainder of the stream with a large probability.

Lemma 3.3. *The probability that the most biased coin is ever defeated as the* king *is at most* $(\delta/2)$.

The proof of these key lemmas are postponed to the next two parts. Theorem 1 now follows easily from these and Claim 3.1.

Proof of Theorem 1. Claim 3.1 ensures the bound on the sample complexity of the algorithm, and Lemmas 3.2 and 3.3 together with a union bound ensure that with probability at least $1-\delta$, we return the most biased coin as the answer. ∎

3.3.1 ***Proof of Lemma 3.2.*** Let king be any coin other than the most biased coin and suppose the next arriving coin is the most biased one (denoted by coin^*). Define $\mathcal{E}_\ell$ as the event that coin^* wins against the king until level ℓ, we can therefore write the probability that coin^* defeats king as follows:

$$\begin{aligned}\Pr\left(\text{coin}^* \text{ loses to king}\right) &\leq \sum_{\ell=1}^{\infty} \Pr\left(\neg\mathcal{E}_\ell \mid \mathcal{E}_{\ell-1}\right)\\ &\leq \sum_{\ell=1}^{\infty} 2\cdot\exp\left(-\ln(1/\delta)\cdot r_\ell\right)\\ &\qquad\text{(by Chernoff-Hoeffding bound)}\\ &\qquad\text{(s_ℓ is the number of samples done in level ℓ)}\\ &< 2\delta\cdot\sum_{\ell=1}^{\infty}\exp\left(-3^\ell\right)\\ &\qquad\text{(by definition of $r_\ell = 3^\ell$ and since $\ln(1/\delta)\cdot r_\ell \geq \ln(1/\delta)+r_\ell$)}\\ &< (\delta/2)\\ &\qquad\text{(as this series converges to $< 1/10$)}\end{aligned}$$

Since the budget is finite, king will lose to coin^* in finite time with probability $1-(\delta/2)$.

3.3.2 ***Proof of Lemma 3.3.*** We first need to set up some notation. Let $T \in [n]$ denote the time step at which the most biased coin arrives in the stream (i.e., coin_T is the most biased coin coin^*). We define the following random variables $\{X_{ij}\}$ for $i, \ell \geq 1$ as the number of coin tosses when comparing king with coin_{T+i} at level ℓ of their challenge (note that index i refers to the i-th coin that arrives *after* the most biased coin, not from the beginning of the stream):

$$X_{i\ell} = \begin{cases} 0 & \text{If the challenge of } \text{coin}^* \text{ and } \text{coin}_{T+i} \\ & \quad \text{did } \textit{not} \text{ reach level } \ell \\ s_\ell & \text{Otherwise}\end{cases}.$$

For any $i \geq 1$, we further define $X_i = \sum_{\ell=1}^{\infty} X_{i\ell}$ which is the number of coin tosses when challenging coin_{T+i} with the king. Finally, define $Y_i := \sum_{j=1}^{i} X_j$. We prove that with probability $\geq 1-(\delta/2)$,

$$\text{for every } i \geq 1: \qquad Y_i < i\cdot b. \tag{1}$$

This proves Lemma 3.3 since: (1) the *total* number of samples from the time the coin^* is chosen as king till the i-th next coin arrives in the stream is Y_i and (2) the king receives $b\cdot i$ budget by the time we reach the i-th coin; hence, having $Y_i < i\cdot b$ *for all* i simultaneously, implies that the king never exhausted its budget and hence was not overthrown till the end of the stream.

In proving Eq (1), working directly with random variables defined above is rather tricky (as it will become evident from our proof). Hence, we instead define the following random variables:

$$\left\{X'_{i,\ell}\right\}: \qquad X'_{i\ell} = \begin{cases} 0 & \text{If the challenge of } \text{coin}^* \text{ and } \text{coin}_{T+i} \\ & \quad \text{did } \textit{not} \text{ reach level } \ell \\ r_\ell & \text{Otherwise}\end{cases};$$

(the difference with $X_{i\ell}$ is that we are setting $X'_{i\ell}$ to r_ℓ not s_ℓ)

$$\left\{X'_i\right\}: \qquad X'_i = \sum_{\ell=2}^{\infty} X'_{i,\ell}, \qquad \left\{Y'_i\right\}: \qquad Y'_i = \sum_{j=1}^{i} X'_j.$$

(note that in defining X'_i we are starting ℓ from 2 and *not* 1)

By these definitions, for every $i \geq 1$,

$$X_i \leq \left(\frac{4}{\Delta^2}\cdot\ln(1/\delta)\right)\cdot X'_i + s_1, \qquad Y_i \leq \left(\frac{4}{\Delta^2}\cdot\ln(1/\delta)\right)\cdot Y'_i + i\cdot s_1.$$

Hence, by the choice of budget increment b, to prove Eq (1), it suffices to prove the following:

$$\text{for every } i \geq 1: \qquad Y'_i < C\cdot i. \tag{2}$$

We now prove Eq (2). The approach is to bound the expected value of each Y'_i, prove that it is concentrated (by showing X'_j is a sub-exponential variable and apply Bernstein's inequality to Y'_i), and show that this concentration is enough to do a union bound over a $\Theta(n)$-length stream.

Claim 3.4. *For all* $i > 0$, $\mathbb{E}\left[Y'_i\right] \leq i$.

Proof. We prove that $\mathbb{E}[X'_j] \leq 1$ for every $j \in [i]$ which implies the claim by linearity of expectation. For every level $\ell > 1$ of the challenge, defining the event $\mathcal{H}_\ell$ as the event for the challenge to reach level ℓ, we have,

$$\begin{aligned}\Pr\left(\mathcal{H}_\ell\right) &\leq \Pr\left(\mathcal{H}_\ell \mid \mathcal{H}_{\ell-1}\right)\\ &\leq 2\exp\left(-\ln(1/\delta)\cdot r_{\ell-1}\right).\end{aligned} \tag{3}$$

where the inequality is by Chernoff-Hoeffding bound (for the event of coin^* losing) and $s_{\ell-1}$ number of samples done in level $\ell-1$. For the random variable X'_j, we have,

$$\begin{aligned}\mathbb{E}\left[X'_j\right] &\leq \sum_{\ell>1}\Pr\left(\text{challenge gets to level } \ell\right)\cdot r_\ell\\ &\leq \sum_{\ell>1} 2\exp\left(-\ln(1/\delta)+r_{\ell-1}\right)\cdot r_\ell\\ &\qquad\text{(by Eq (3) and since $\ln(1/\delta)\cdot r_\ell \geq \ln(1/\delta)+r_\ell$)}\\ &\leq 2\delta\cdot\sum_{\ell>1}\frac{3^\ell}{\exp\left(3^{\ell-1}\right)} \leq \delta.\\ &\qquad\text{($r_\ell = 3^\ell$, $r_{\ell-1} = 3^{\ell-1}$, and series converges to $< 1/2$)}\end{aligned}$$

Noting that $\delta < 1$ concludes the proof of the claim. ∎

Claim 3.4 suggests that $\left\{Y'_i\right\}$ behave as we require in Eq (1) in expectation. To prove a concentration bound, we prove that each $(X'_i - \mathbb{E}\left[X'_i\right])$ is a sub-exponential variable with small κ.

Claim 3.5. *For all* $i > 0$, $(X'_i - \mathbb{E}\left[X'_i\right])$ *is a sub-exponential random variable with* $\kappa = \frac{15}{\ln(1/\delta)}$.

Proof. Fix any $t > 0$ and let ℓ be the *largest* level where $\sum_{j=2}^{\ell} r_j \leq t$. Note that since $\{r_j\}_{j=1}^{\infty}$ forms a geometric series, we have $t \leq 5 \cdot r_\ell$. We thus have,

$$\begin{aligned}\Pr\left(\left|X'_i - \mathbb{E}\left[X'_i\right]\right| > t\right) &\leq \Pr(\text{challenge gets to level } \ell)\\ &\leq 2\exp\left(-\ln(1/\delta)\cdot r_{\ell-1}\right) \qquad \text{(by Eq (3))}\\ &\leq 2\exp\left(-\ln(1/\delta)\cdot\frac{t}{15}\right).\\ &\qquad\qquad \text{(as } t \leq 5\cdot r_\ell = 15\cdot r_{\ell-1})\end{aligned}$$

This implies the proof by definition of sub-exponential variables in Bernstein's inequality. ∎

We can now apply Bernstein's inequality to $(Y'_i - \mathbb{E}\left[Y'_i\right]) = \sum_{j=1}^{i}(X'_j - \mathbb{E}[X'_j])$ (since by Claim 3.5, variables $\left\{X'_j\right\}$ are independent and sub-exponential with $\kappa = \frac{15}{\ln(1/\delta)}$):

$$\begin{aligned}\Pr\left(Y'_i \geq C\cdot i\right) &\leq \Pr\left(\left|Y'_i - \mathbb{E}\left[Y'_i\right]\right| \geq (C-1)\cdot i\right)\\ &\leq 2\cdot\exp\left(-c\cdot\min\left(\frac{(C-1)^2\cdot i^2}{\kappa^2\cdot i}, \frac{(C-1)\cdot i}{\kappa}\right)\right)\\ &\qquad \text{(by Claim 3.4 to bound the expectation)}\end{aligned}$$

(and by Claim 3.5 and Bernstein's inequality, $c > 0$ is a constant)

$$\begin{aligned}&\leq 2\cdot\exp\left(-c\cdot\frac{(C-1)\cdot i\cdot\ln(1/\delta)}{15}\right)\\ &\qquad \text{(by the value of } \kappa = \tfrac{15}{\ln(1/\delta)} \text{ in Claim 3.5)}\\ &\leq (\delta/2)\cdot\exp(-i).\\ &\qquad \text{(by picking } C \text{ to be a sufficiently large constant)}\end{aligned}$$

Finally, by this and a union bound for all choices of i, we have,

$$\begin{aligned}\Pr\left(\exists i : Y'_i \geq C\cdot i\right) &\leq (\delta/2)\cdot\sum_{i=1}^{n}\exp(-i) < (\delta/2).\\ &\qquad \text{(as this series converges to } \tfrac{1}{e-1} < 1)\end{aligned}$$

This proves that with probability $\geq 1-(\delta/2)$, Eq (2) holds, finalizing the proof of Lemma 3.3.

Remark 3.6. The proof of Lemma 3.3 implies a bound of a random walk with flexible step size (rather than −1 and +1). As the analysis of such type of random walk may be useful in other settings as well, we abstract out this problem and its corresponding analysis in the Appendix of the full version [5] .

4 TOP k MOST BIASED COINS: AN $O(k)$-COIN MEMORY ALGORITHM

We now consider the more general problem of finding the k most biased coin for any integer $k \geq 1$. In this problem, we have a collection of coins $\{\text{coin}_i\}_{i=1}^{n}$ arriving in a stream; for simplicity of notation, we use $\text{coin}_{[i]}$ to denote the i-th most biased coin among these. Our goal is then to find the k coins with largest biases, namely, $\left\{\text{coin}_{[1]}, \ldots, \text{coin}_{[k]}\right\}$ (in no particular order) for a given integer $k \geq 1$. The gap parameter for this problem, denoted by Δ_k, is now defined as the gap between the bias of the k-th most biased coin and $(k+1)$-th one, namely $\text{coin}_{[k]}$ and $\text{coin}_{[k+1]}$.

We present a streaming algorithm for this problem with asymptotically optimal space complexity as well as sample complexity (by the lower bound of [35]).

Theorem 2 (Formalization of Result 2). *There exists a streaming algorithm that given an integer $k \geq 1$, n coins arriving in a stream with gap parameter Δ_k (between k-th and $(k+1)$-th most biased coins) and confidence parameter $\delta \in (0, 1/2)$, finds the k most biased coins with probability at least $1-\delta$ using $O(\frac{n}{\Delta^2}\cdot\log(k/\delta))$ coin tosses and a memory of $O(k)$ coins.*

4.1 High Level Overview

We follow the same "budgeting" strategy as our algorithm in Section 4. However, as stated in Section 1, there are two main challenges that we need to address: (1) we now need to maintain k "kings", namely, KINGS $= \left\{\text{king}_1, \ldots, \text{king}_k\right\}$ but can no longer compare each arriving coin with (or assign a unit of budget to) every king (otherwise there will be $\Omega(nk)$ samples); and (2) we need to collect all the top-k coins and still cannot guarantee any suitable (probabilistic) outcome while comparing any two of these coins to each other (as there may be no gap between their biases).

There is a natural way for addressing the first challenge: instead of comparing each arriving coin with the k king-coins using $O(k)$ coin tosses, delay processing of arriving coins, by storing them in a *buffer* B, until we collect roughly k of them; then handle all these coins using $O(k\log k)$ coin tosses in total by running the following *trial*: pick a *pivot* coin from B, compare this pivot with every king and every coin in B, and *prune* the buffer by discarding any coin with empirical bias less than the pivot in this trial. Assuming we prune a *constant fraction* of the buffer per each trial (which seems doable, at least in *expectation*, by picking the pivot *randomly*), we can spend $O(k\log k)$ coin tosses per trial and sample $O(n\log k)$ coins in total. Finally, to compare a king with a pivot, we can use the challenge subroutine (in our algorithm in Section 3): allow any king to use its budget and only consider it lost in a challenge when it exhausts its budget entirely (the coins in the buffer will not collect any budget). We can also allocate $O(k\log k)$ budget per each trial (and *not* per each arriving coin) and hope that this should allow us, similar to Section 3, to argue that any top-k pivot will win against any non-top-k king and will later remain in KINGS till the end.

Except that this actually would *no longer work*, which brings us to the second (and the main) challenge raised above. The problem with the above reasoning is that it does not take into account the outcome of challenging a top-k coin as a pivot with another top-k coin as a king. In such a challenge, the previous probabilistic guarantees in Section 3 no longer hold as we have no control on the gap between the biases of these coins. For instance, it is entirely possible that a top-k pivot completely depletes the budget of a top-k king and the troublesome part is that this is the same exact behavior we would also except from a top-k pivot when challenging a non-top-k king (with no apparent way of distinguishing between the two cases). At the same time, it is also completely possible that the bias of two top-k coins is almost the same and hence their challenges would be completely noisy. The choice of a top-k pivot also highlights another problem: we need to be very "cautious" in the pruning step as when choosing a top-k pivot, we may inadvertently discard

other top-k coins (either in the buffer or among KINGS) when they lose to this top-k coin – note that this goes exactly opposite of our goal of pruning a constant factor of the buffer per each trial.

We address the latter challenge by relaxing the requirement of the algorithm (and the analysis) in maintaining the top-k coins among KINGS throughout the entire length of the stream (after their arrival). In other words, in the course of our algorithm, the top-k coins may float between KINGS (and having a budget) and the buffer B (with no budget). This in turn requires us to relax our pruning rule so that the top-k coins in the buffer do not get discarded in a trial: this is done by limiting the cases when a discard can happen (for instance not doing any pruning when the pivot joins the KINGS), while still ensuring the constant fraction pruning (in expectation) per trial. Finally, the analysis now needs to take into account that a top-k coin may *repeatedly* exhausts its budget and there will be periods of trials in the stream when a top-k coin resides in B with no budget (which we refer to as *risky* trials). Fortunately, by modifying the algorithm appropriately, we can limit the *length* and the *frequency* of such periods throughout the stream and show that with high (constant) probability, any top-k coin will indeed remain among KINGS $\cup$ B till the end.

4.2 The Algorithm

We now present our algorithm in this section. The input to our algorithm is a set of n coins $\{\mathsf{coin}_i\}_{i=1}^{n}$ arriving in an arbitrary order in the stream, the gap parameter Δ_k (the gap between the bias of $\mathsf{coin}_{[k]}$ and $\mathsf{coin}_{[k+1]}$), and the confidence parameter $\delta \in (0,1)$ (the algorithm does not need to know the value of n in advance). We use the following parameters:

$$\{r_\ell\}_{\ell=1}^{\infty}: \quad r_\ell = 3^\ell;$$

(multiplicative factor of coin tosses per each level of the challenge)

$$\{s_\ell\}_{\ell=1}^{\infty}: \quad s_\ell := 16 \cdot \frac{4}{\Delta_k^2} \cdot \ln(k/\delta) \cdot r_\ell;$$

(the number of coin tosses per each level of the challenge)

$$b := 16 \cdot \frac{4}{\Delta_k^2} \cdot C \cdot \ln(k/\delta) + s_1;$$

(the budget given to each king once the buffer is full)

$$K := 10 \cdot k.$$

(the limit on the size of the buffer)

The description of the algorithm can be shown as Algorithm 2 (Federated-Game-of-Coin). We note that at this point, the bound on the sample complexity of this algorithm is *in expectation* and not deterministically. For simplicity of exposition, we analyze this variant of the algorithm first and then point out, in Remark 4.6, how to change this slightly so that the algorithm *never* (deterministically) uses more than a fixed certain number of coin tosses bounded by $O(\frac{n}{\Delta_k^2} \cdot \log(k/\delta))$ (this extension is straightforward). We present the analysis of the algorithm in the next section.

Algorithm 2: Federated-Game-of-Coin

(1) Initialize KINGS $= \{\mathsf{king}_1, \ldots, \mathsf{king}_k\}$ by the first k arriving coins and let B be the buffer.
(2) For any $\mathsf{king}_i \in$ KINGS, define the budget $\Phi_i := \Phi(\mathsf{king}_i)$ which is initialized to 0.
(3) While number of coins in B is less than K, add the next coin in the stream to B.
(4) **Trial subroutine:** Otherwise, run the following *trial*:
  (a) Pick a *pivot* $\overline{\mathsf{coin}}$ uniformly at random from B. Increase the budget Φ_i of king_i by b.
  (b) **Buffer-challenge:** For each $\mathsf{coin}_i \in B$: toss both coin_i and $\overline{\mathsf{coin}}$ for s_1 times and record which one had a higher empirical bias.
  (c) **King-challenge:** For each $\mathsf{king}_i \in$ KINGS: run the **challenge subroutine** of Game-Of-Coins between $\overline{\mathsf{coin}}$ and king_i (with new $\{s_\ell\}$ and budget $\Phi(\mathsf{king}_i)$) and record which coin won the challenge (but do not discard any coin).
  (d) Let D denote the recorded number of times $\overline{\mathsf{coin}}$ was defeated in the trial.
  (e) *Discard case:* If $D \geq k$: discard $\overline{\mathsf{coin}}$, any coin in KINGS$\cup$ B that lost to $\overline{\mathsf{coin}}$. Then fill up the remainder of KINGS with the arriving coins of the stream and go to (3).
  (f) *Swap case:* If $D < k$: pick $\overline{\mathsf{king}}$ uniformly at random coins in KINGS that were defeated by $\overline{\mathsf{coin}}$ (such a coin should exists) and swap $\overline{\mathsf{coin}}$ and $\overline{\mathsf{king}}$, i.e., make $\overline{\mathsf{coin}}$ a new king (with zero budget) and add $\overline{\mathsf{king}}$ to B. Then repeat the trial by going to (a).
(5) At the end, toss each of the coins in KINGS$\cup B$ for s_1 times and return the top-k ones according to their empirical bias as the answer.

4.3 The Analysis

There are two main parts in the analysis, which guarantee the sample complexity and the correctness of the algorithm, respectively:

Lemma 4.1. *The expected number of coin tosses by the algorithm is* $O(\frac{n}{\Delta_k^2} \cdot \log(\frac{k}{\delta}))$.

Lemma 4.2. *The probability that even a single* $\mathsf{coin}_{[j]}$ *for* $j \in [k]$ *is discarded before the end of the stream (before Line (5)) is is at most* $\frac{\delta}{2}$.

Given the above two lemmas, we will be ready to prove Theorem 2. The only remaining part is to show the last step of the algorithm is also correct, which is given as

Lemma 4.3. *With probability at least* $1-\delta$*, the algorithm will return the top-k coins in line (5) of algorithm* Federated-Game-of-Coin.

Lemma 4.3 is easy to show as it follows from our earlier results (and also from known results in the literature since we can simply run any standard algorithm for finding top-k coins on these set of $O(k)$ coins at the end).

Proof of Theorem 2. The number of coin tosses for the algorithm immediately follows from conclusion of Lemma 4.1. Moreover, Lemma 4.2 ensures that with probability at least $1-\frac{\delta}{2}$, all the top-k coins will be maintained in KINGS $\cup B$ by the end of the stream. Finally, by Lemma 4.3, the very final step of the algorithm also correctly returns the set of top-k coins. ∎

In the following, we present the proofs for Lemma 4.1 and Lemma 4.2.

4.3.1 ***Proof of Lemma 4.1 (Sample Complexity).*** Let us recall that in the algorithm, coin tossing happens only during a *trial* in the trial subroutine (ignoring the last $O(k \cdot s_1)$ samples in (5) which are clearly within the desired sbounds on sample complexity by definition of s_1), namely, when the buffer is full and we pick a pivot for challenging the other coins. Let N_{trial} denote the *number of trials* in the algorithm. We have the following claim based on a similar amortized analysis as in our Game-Of-Coins algorithm.

Claim 4.4. *The total number of coin tosses in the algorithm is* $O(k \cdot b \cdot N_{\text{trial}})$.

Claim 4.4 implies that we can bound the sample complexity of the algorithm by bounding N_{trial} which is the content of the next claim.

Claim 4.5. *The expected number of trials is* $\mathbb{E}[N_{\text{trial}}] = O(\frac{n}{k})$.

Proof. Consider the following event:

- $\mathcal{E}_{\text{pivot}}$: the pivot coin $\overline{\text{coin}}$ loses to at least k coins and wins over at least k other coins.

Whenever $\mathcal{E}_{\text{pivot}}$ happens, we discard at least k coins from the buffer. By lower bounding the probability of this event by a constant, we can then argue that the expected number of coins discarded in each trial is $\Omega(k)$. As the next trial can only happen when the buffer again becomes full (thus after $\Omega(k)$ new coins are visited), this will allow us to argue that the expected number of trials before we process the entire stream is $O(n/k)$.

We now lower bound the probability that $\mathcal{E}_{\text{pivot}}$ happens by considering a simpler case that ensures $\mathcal{E}_{\text{pivot}}$. The total number of coins in KINGS$\cup B$ is $K+k = 11k$. Let us sort these coins in decreasing order of their biases as $\text{coin}_{(1)}, \text{coin}_{(2)}, \ldots, \text{coin}_{(K+k)}$. We further partition these coins into the *top* part $Top := \{\text{coin}_{(1)}, \ldots, \text{coin}_{(5k)}\}$, the *middle* part $Mid := \{\text{coin}_{(5k+1)}, \ldots, \text{coin}_{(7k)}\}$, and the *bottom* part $Bot := \{\text{coin}_{(7k+1)}, \cdots, \text{coin}_{(11k)}\}$. See Figure 1 for an illustration.

Now firstly note that since we only have k coins, the probability that the pivot is chosen from Mid is at least $\frac{2k-k}{K} = \frac{1}{10}$. In the following, we condition on this event. Note that conditioned on this event, any coin in Top would lose to $\overline{\text{coin}}$ with probability at most $1/2$, and any coin in Bot which is *not* in KINGS would win against $\overline{\text{coin}}$ with probability at most $1/2$ (a coin in Bot which is a king may have collected a lot of budget and thus still have a more chance of winning against $\overline{\text{coin}}$ even though its bias is less than it). We define the following random variables.

- X_{lose}: number of coins in Top that lose to $\overline{\text{coin}}$ – let $X_{win} = |Top| - X_{lose}$.
- Y_{win}: number of coins in $Bot \setminus$ KINGS that win against $\overline{\text{coin}}$ – let $Y_{lose} = |Bot \setminus \text{KINGS}| - Y_{win}$.

Therefore, we will have $\mathbb{E}\left[X_{lose} \mid \overline{\text{coin}} \in Mid\right] \le 5k/2$ and $\mathbb{E}\left[Y_{win} \mid \overline{\text{coin}} \in Mid\right] \le 3k/2$. As such,

$$\Pr\left(X_{win} < k \mid \overline{\text{coin}} \in Mid\right) \le \Pr\left(X_{lose} \ge 4k \mid \overline{\text{coin}} \in Mid\right) \le \frac{5}{8},$$

$$\Pr\left(Y_{lose} < k \mid \overline{\text{coin}} \in Mid\right) \le \Pr\left(Y_{win} \ge 2k \mid \overline{\text{coin}} \in Mid\right) \le \frac{3}{4},$$

where both inequalities are by Markov bound. Moreover, we have,

$$\Pr\left(X_{win} \ge k \wedge Y_{lose} \ge k \mid \overline{\text{coin}} \in Mid\right) \ge \left(1-\frac{5}{8}\right)\cdot\left(1-\frac{3}{4}\right) = \frac{3}{32},$$

since these events are independent of each other. Define the above event as $\mathcal{E}_{\text{discard}}$, notice that whenever the event above happens, we would be in the 'discard case' of the algorithm (since $\overline{\text{coin}}$ has lost to at least k coins in Top) and we would discard at least k coins (all the coins in Y_{lose} that belong to Bot). Hence,

$$\begin{aligned}\Pr\left(\mathcal{E}_{\text{pivot}}\right) &\ge \Pr\left(X_{win} \ge k \wedge Y_{lose} \ge k \wedge \overline{\text{coin}} \in Mid\right)\\ &= \Pr\left(\overline{\text{coin}} \in Mid\right)\cdot \Pr\left(\mathcal{E}_{\text{discard}}\right)\\ &\ge \frac{1}{10}\cdot\frac{3}{32} = \frac{3}{320} \ge \frac{1}{200}.\end{aligned}$$

This implies that the expected number of coins that are discarded in each trial is at least $k/100$. Moreover, note that this lower bound holds in every trial *independent* of the outcome of the past trials (event hough the events between the two trials may not necessarily be independent). This means that the distribution of N_{trial} stochastically dominates the distribution of number of times we see a head by tossing a biased coin with probability $1/200$ of showing a head. For the latter distribution we know that the expected number of tries before we see n heads is $200 \cdot n/k$ and hence we also have $\mathbb{E}[N_{\text{trial}}] \le 200 \cdot n/k$ (as after seeing n coins the trials are finished). ∎

We now formally conclude the proof of Lemma 4.1. Combine Claim 4.4 and Claim 4.5, one can observe that the expected number of coin tosses will be $O(k \cdot b \cdot N_{\text{trial}}) = O(k \cdot b \cdot \frac{n}{k}) = O(b \cdot n)$. And according to the definition, this is $O(\frac{n}{\Delta_k^2} \cdot \log(\frac{k}{\delta}))$ as desired.

Remark 4.6. We remark that the probability that N_{trial} is more than twice its expectation is exponentially small in $\Theta(n/k)$ (which we can assume k is at most $\sqrt{n}$ since whenever $k \ge \sqrt{n}$, we can simply toss each coin $O(\log n)$ times to obtain its 'almost true' bias and still be within the correct budget – but in this case, we can simply run a deterministic algorithm for finding top-k coins in the stream over the empirical biases). As such, we can simply modify the algorithm by terminating with an arbitrary answer whenever the N_{trial} reaches twice its expected value – this can only decrease the probability of success by $\exp\left(-\Theta(\sqrt{n})\right)$ (which again can be assumed to be always $o(\delta)$ by a similar argument as why assuming $k \le \sqrt{n}$ is without loss of generality). This means that the sample complexity of our algorithm can be bounded *deterministically* also.

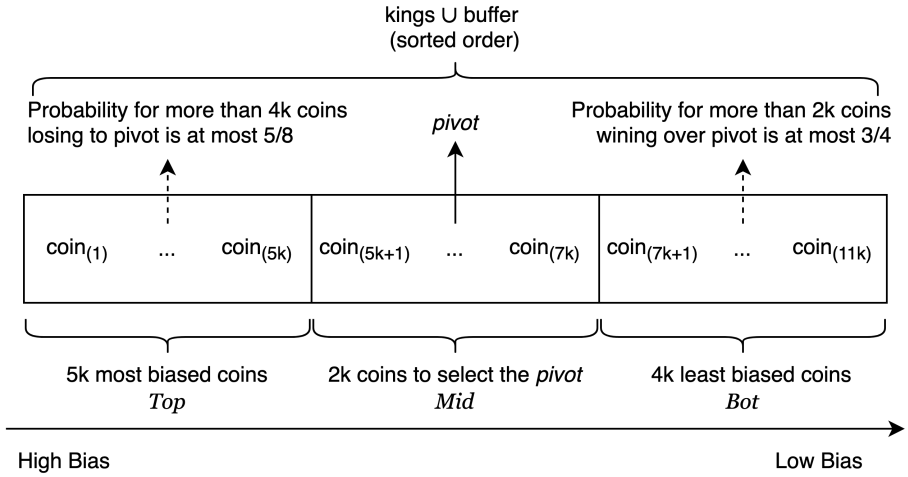

Figure 1: When picking the pivot between the $(5k+1)$-th and $(7k)$-th most biased coins (namely, from *Mid*) from the current KINGS and buffer, the probability for the *pivot* to win over at least k coins *and* lose against at least k coins is at least a constant.

4.3.2 ***Proof of Lemma 4.2 (Correctness of the Algorithm).*** Let us start by giving some intuition about the proof before diving into the technical details. The ideal scenario for the algorithm is if we start with all the top-k coins appearing at the beginning of the stream and so from the get go, they all belong to KINGS. In such a scenario, we can invoke Lemma 3.3 from Section 3 in an almost black-box way and argue that the budgeting scheme allows for all these coins to remain in king till the very end of the stream with probability at least $1-\delta$. The reason this works is that in this case, we never need to consider comparing two top-k coins with each other (as the pivots are sampled from the buffer alone).

Of course, in general, we will not have all top-k coins as KINGS in the beginning. The first thing we need to worry is when a top-k coin enters the buffer (and for now let us assume there is no other top-k coin the buffer for the next foreseeable streaming steps): since this top-k coin does not have any budget, can we still hope to have it around for multiple trials before it is chosen as the pivot and even have a chance of joining the KINGS? Since the pivot is chosen uniformly at random, we would expect this top-k coin to become a pivot itself within the next $O(k)$ trials. Thus, we only need this coin to remain in the buffer for the next $O(k)$ trials; as the coins are sampled $O(\log k)$ times in each trial, we can guarantee this event. Moreover, once this coin is chosen as the pivot, we can also guarantee that it will join the KINGS by the same argument as Lemma 3.2 in Section 3.

Already at this point, we encounter a problem: What if this top-k coin swaps one of the top-k coins in KINGS? Indeed, our pruning rule allows us to argue that with high probability we will *not* have a *discard* step when this coin joins KINGS but inevitably a swap needs to happen and we may very well swap a top-k coin with with another top-k coin. This can become even more challenging when multiple top-k coins all join the buffer.

Our main argument here is to show that it is possible to partition the execution of the algorithm over the stream into *long* sequences of "relative safety" in which no top-k coin belongs to the buffer and the top-k coins in KINGS start to accumulate budget (which allows us to do union bound over these long sequences), and *short* outbursts of "risky" trials in which the budget of every king may be depleted and the only thing that saves us through these risky trials is that their numbers are small (so we can directly use a union bound over them). The final step is to use a simple potential function argument to prove that the total number of such risky outbursts is small and most of the stream involves the long non-risky trials (so even though the budgets of the top-k coins in KINGS may get restarted after each risky outbursts, we can still expect them to survive all these outbursts and not get discarded by the end of the stream). We now formalize this intuition.

We start by setting up our notation. Let us define:

- *Risky trial*: A trial with at least one of the top-k coins present in the buffer B;
- *Non-risky trial*: A trial without any coin from the top-k coins present in the buffer B;
- *(Non-risky) Chunk*: A maximal sequence of consecutive non-risky trials.

What we intend to prove is as follows (see Figure 2 for an illustration of these definitions):

(i) During any single non-risky chunks, with large probability of $1-\frac{\text{poly}(\delta)}{\text{poly}(k)}$, we will not encounter any 'swap case' or 'discard case' of the algorithm that *removes a top-k coin from* KINGS. In other words, we only enter a risky trial on the condition of a new arriving top-k coin joining the buffer from the stream (Claim 4.7).

(ii) For a single risky trial, with large probability of $1-\frac{\text{poly}(\delta)}{\text{poly}(k)}$, no coin among the top-$k$ will be discarded, even though we may encounter many 'swap case' or 'discard case' in the algorithm (Claim 4.8).

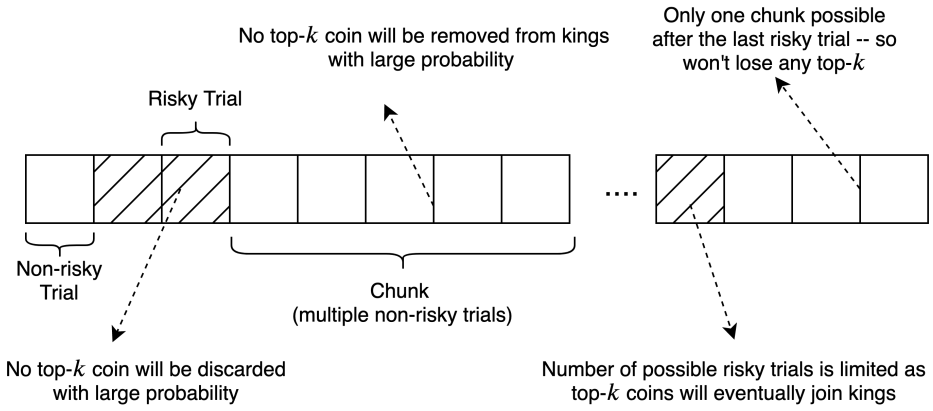

Figure 2: An illustration of the notation, events and arguments adopted in the proof of Lemma 4.2.

(iii) The expected number of risky trials as well as (non-risky) chunks is poly(k)/poly(δ) where the bound is small enough to do a union bound over all occurrences of the above cases (Claim 4.9).

The proof of the following claim is analogous to Lemma 3.3.

Claim 4.7. *With probability at least* $1 - \frac{\delta^2}{k^{15}}$*, any top-*$k$ *coin in* KINGS *will not be defeated during a fixed (non-risky) chunk.*

The proof is similar to the proof of Lemma 3.3, with the only difference as we will toss each coin $O(\frac{1}{\Delta_k} \cdot \log(\frac{k}{\delta}))$ times on average, which gives us the $1/\text{poly}(k)$ term on the denominator.

Claim 4.8. *With probability at least* $1 - \frac{\delta^{16}}{k^9}$*, in a single risky trial, no top-*k *coin will be discarded.*

Proof. We first argue that the only way for any top-k coin to get discarded is if one of the following two events happens:

- $\mathcal{E}_{\text{defeated-top}}$: $\overline{\text{coin}}$ is *not* a top-k coin and defeats a top-k coin in KINGS $\cup$ B.
- $\mathcal{E}_{\text{pivot-top}}$: $\overline{\text{coin}}$ is a top-k coin that loses at least k times (namely, have $D \geq k$).

This is the case because of the following: if $\overline{\text{coin}}$ is not a top-k coin, the only way for it to be able to discard a top-k coin is if it wins against it (which is captured by$\mathcal{E}_{\text{defeated-top}}$). On the other hand, if $\overline{\text{coin}}$ is a top-k coin, the only for it to be able to discard *any* other coin, is if it enters a 'discard case' that only happens if it loses at least k times (which is captured by $\mathcal{E}_{\text{pivot-top}}$).

We now bound the probability of each of these two events. Fix any top-k coin$^* \in$ KINGS $\cup$ B and the pivot $\overline{\text{coin}}$. Note that $\overline{\text{coin}}$ and coin* are tossed *at least* s_1 times before we decide which one is the winner (coin* may have a budget if it belongs to KINGS on top of the $b \geq s_1$ provided to it at the beginning of this trial but we may and will ignore that for this argument). By the choices of the parameters, we should have $\Pr\left(\text{coin}^* \text{ loses to } \overline{\text{coin}}\right) \leq \frac{2\delta^{16}}{k^{16}}$. Now applying the fact that $k \geq 2$ and $\delta \leq< 1/2$, we have

$$\Pr\left(\mathcal{E}_{\text{defeated-top}}\right) \leq \frac{\delta^{16}}{k^{14}}.$$

The trickier part is to upper-bound the probability of $\mathcal{E}_{\text{pivot-top}}$: when we should compare $\overline{\text{coin}}$ with some king$_i \in$ KINGS which is not a top-k coin itself. Here, we can no longer ignore the fact that king$_i$ may have collected some budget. So $\overline{\text{coin}}$ needs to win against king$_i$ despite king$_i$ having some budget (that we cannot necessarily bound beyond saying it is finite). However, we already proved an analogous statement like this in Lemma 3.2 and the argument here is identical to that. Indeed, by similarly defining $\mathcal{E}_{i,\ell}$ as the event that $\overline{\text{coin}}$ wins against the king$_i$ until level ℓ, we have,

$$\begin{aligned}\Pr\left(\overline{\text{coin}} \text{ loses to king}_i\right) &\leq \sum_{\ell=1}^{\infty} \Pr\left(\neg\mathcal{E}_{i,\ell} \mid \mathcal{E}_{i,\ell-1}\right) \\ &< \frac{\delta^{16}}{k^{16}}.\end{aligned}$$

By a union bound over the at most $11k$ non-top-k coins in KINGS$\cup B$, we have that

$$\Pr\left(\mathcal{E}_{\text{pivot-top}}\right) \leq \frac{\delta^{16}}{k^{10}}.$$

A union bound over these two events (and a very loose upper bound) finalizes the proof. ∎

Claim 4.9. *Assuming the events of Claim 4.8 for every upcoming risky trial, with probability at least* $1 - \frac{\delta}{k^3}$*, the number of risky trials is at most* $10 \cdot \frac{k^6}{\delta}$.

Proof. Let us fix any risky trial. By definition, there must exists at least one top-k coin, denoted by coin*, in the buffer B. By the random choice of the pivot, we will pick coin* as the pivot with probability $\frac{1}{10k}$. Let us condition on this event.

Moreover, note that since not all KINGS are top-k coins, there exists at least one non-top-k king, denoted by king$_i$, in KINGS. By conditioning on the event of Claim 4.8, coin* will beat king$_i$ and also enters a 'swap case'. However, there is no guarantee that

coin^* did not win against some other coins, some of which may actually be top-k coin themselves. In that case, one of those may get swapped with coin^* instead of king_i. Still, considering we pick $\overline{\text{king}}$ to swap with coin^* uniformly at random, there is at least a $\frac{1}{k}$ chance that we pick to swap king_i with coin^*. This means that, assuming the event in Claim 4.8, with probability at least $\frac{1}{10k^2}$, we will swap coin^* with king_i.

An important observation here is that as long as the event in Claim 4.8 continues to happen, we will never *decrease* the number of top-k coins in KINGS (this actually follows from the event $\mathcal{E}_{\text{defeated-top}}$ bounded in Claim 4.8 and not the exact statement of the claim itself). This, plus the above fact implies that in each trial, we have a probability of $\geq \frac{1}{10k^2}$ to increase the number of top-k coins among the KINGS (and we will not decrease it conditioned on Claim 4.8). As the number of top-k coins in KINGS can be increased to k only, we can conclude that the expected number of risky trials before we increase the top-k coins in KINGS to become k is $10k^3$. Hence, by Markov bound, with probability $1 - \frac{\delta}{k^3}$, we can only have $10 \cdot \frac{k^6}{\delta}$ risky trials. ∎

We can now use Claim 4.8 and Claim 4.9 and do a union bound (step by step on each upcoming risky trial) to present two arguments: (1). With probability at least $1 - \frac{2\delta}{k^3}$, we will keep all the top-k coins and have no more than $10 \cdot \frac{k^6}{\delta}$ risky trials; (2). Under the condition of (1), the number of consecutive non-risky trials ('non-risky chunks') is at most $10 \cdot \frac{k^6}{\delta}$, and therefore the probability of lossing any top-k coins is at most $\frac{\delta}{k^5}$. Taking together, it means with probability $\left(1 - \left(\frac{2\delta}{k^3} + \frac{\delta}{k^5}\right)\right) \geq 1 - \frac{\delta}{2}$ we will not lose any top-k coin throughout the stream, which proves Lemma 4.2.

5 PARTITION WITH NOISY COMPARISONS

In this section, we consider one applications of our techniques to the problem of top-k recovery from noisy comparisons.

5.1 Problem Definition

In this problem, we have a collection of n elements, denoted by $\{\text{element}_i\}_{i=1}^n$, with an *unknown total order* over these elements. The algorithm has a 'noisy' access to this ordering: for any pairs of elements, the algorithm can query the order between the elements of this pair; with probability $1/2 + \gamma$, the answer is according to the underlying total ordering, and with the remaining probability, the answer is arbitrary. The goal in the top-k problem is to, given $\{\text{element}_i\}_{i=1}^n$, parameters k and γ, and query access to the underlying ordering, output the top largest k elements according to this ordering, using a minimal number of queries. This problem is also sometimes referred to as the *select* problem and its special case of $k = 1$ is called the *MAX* problem in the literature.

We can model this problem in the streaming setting as before: the elements in $\{\text{element}_i\}_{i=1}^n$ are arriving one by one in the stream and the algorithm is only allowed to store a limited number of these elements – to query a pair of elements at any point, both elements are required to be in the memory of the algorithm.

5.2 Our Results for the Top-k Recovery Problem

We obtain the following algorithms for this problem.

Theorem 3. *(Formalization of Result 3) There exists streaming algorithms that given n elements arriving in a stream, parameters k and γ, and the confidence parameter δ, with probability at least $1 - \delta$, find the top k largest element in the underlying ordering in the noisy comparison model, using $O(k)$ memory and $O(\frac{n}{\gamma^2} \cdot \log(k/\delta))$ (noisy) comparisons.*

We shall note that the number of comparisons done by all our algorithms are *optimal* (even in the absence of any memory restriction). The algorithms can be directly obtained by showing that the top-k recovery problem is mathematically equivalent to finding the k most biased coin with gap at least γ. In this sense, one can directly apply our algorithms in Theorem 1 and Theorem 2 (depending on whether $k \geq 2$) to get the results.

Specifically, we show the mathematical equivalence of the two problems by the following lemma:

Lemma 5.1. *Let element_1 and element_2 be a pair of elements with true order $\text{element}_1 > \text{element}_2$ ('$>$' here means 'has a higher order than'). Suppose the noisy comparison will return a correct answer with probability $\frac{1}{2} + \gamma$, and we query the comparison $\frac{K}{\gamma^2}$ times and determine the element that wins the most times as the higher order element. Then,*

$$\Pr(\text{element}_2 \text{ is considered higher order}) \leq 2 \cdot \exp\left(-\frac{1}{4} \cdot K\right).$$

The lemma can be straightforwardly proved by defining the random variables as an analogy of the coin tossing problem.

6 EXPLORATION IN STOCHASTIC MULTI-ARMED BANDITS

We consider another application of our algorithms, this time to the exploration problem in stochastic multi-armed bandits.

6.1 Problem Definition

In the (stochastic) multi-armed bandit (MAB) problem, we have a collection of n arms $\{\text{arm}_i\}_{i=1}^n$. Each sample (or pull) of any arm_i results in a reward in $[0, 1]$ sampled from an unknown distribution with mean $\mu_i \in [0, 1]$[2]. For a parameter $\varepsilon \in (0, 1)$, we say that an arm_i is an *ε-best arm* if its expected reward is at most ε smaller than the expected reward of the maximum (the best arm), or alternatively $\mu_i \geq \max_j \mu_j - \varepsilon$. In the exploration problem, our goal is to, given the arms $\{arm_i\}_{i=1}^n$ and a parameter $\varepsilon > 0$, return any ε-best arm using a minimal number of arm pulls.

We study this problem in the streaming model as follows: The arms are arriving one by one in a stream and the algorithm needs

[2]Our results extend verbatim to any Sub-Gaussian reward distribution with no assumption on range of the rewards. This, to the best our knowledge, is the common characteristic of all prior work on exploration in MAB as well, and simply follows from the fact that the Chernoff-Hoeffding inequality used in the proofs extends directly to these distributions. As such, we omit the details.

to store each arm explicitly if it wants to pull it at some later point in the stream as well.

6.2 Our Results for the ε-Best Arm Problem

We design the following streaming algorithms for this problem.

Theorem 4. *There exist streaming algorithms that given n arms arriving in a stream, the approximation parameter $\varepsilon \in (0,1)$, and the confidence parameter δ, with probability at least $1-\delta$, finds an ε-best arm using:*

- *a memory of a single arm and $O(\frac{n}{\varepsilon^2} \cdot \log(1/\delta))$ arm pulls assuming at least ε gap between the largest (expected) reward and the second largest reward;*
- *a memory of $O(\log^*(n))$ arms and $O(\frac{n}{\varepsilon^2} \cdot \log(1/\delta))$ arm pulls in general.*

Intuitively, if a gap of at least ε exists between the best and second-best arms, then the problem can essentially be solved by our main algorithm Game-Of-Coins – nothing needs to be changed except the notations. However, unfortunately, for the problem of finding the ε-best arm without the gap guarantee, our main algorithm does not work in general. The issue here is that if a bunch of arms with gaps far smaller than ε arrive in a consecutive manner, the less stronger arms will have concrete probabilities to replace the stronger ones. And if this type of event happens over time, arms with gap larger than ε will be able to be selected.

We tackle the above problem by iteratively refining the gap of selecting arms. Specifically, we leverage the framework of the $\log^*(n)$ space algorithm (one of the intermediate algorithms), and repetitively narrowing the gap of $\varepsilon_l = O(\frac{\varepsilon}{2^{l-1}})$ at each layer l. Since the number of arms with the $\log^*(n)$ space algorithm will ruling out arms by a tower factor, we will have enough additional budget to pay for the up-sampling factor. The algorithm can be shown as follows:

Algorithm 3:
Parameters:
$\{r_\ell\}_{\ell=1}^{\infty} : r_1 := 4, r_{\ell+1} = 2^{r_\ell}, \quad \varepsilon_\ell = \frac{\varepsilon}{10 \cdot 2^{\ell-1}} \quad \beta_\ell = \frac{1}{\varepsilon_\ell^2}$
$s_\ell = 4\beta_\ell(\ln(\frac{1}{\delta}) + 3r_\ell) \quad c_1 = 2^{r_1},\ c_\ell = \frac{2^{r_\ell}}{2^{\ell-1}} (\ell \geq 2)$
Counters: $C_1, C_2, ..., C_t \quad t = \lceil \log^*(n) \rceil + 1$
Stored arms: $\text{arm}_1^*, \text{arm}_2^*, ..., \text{arm}_t^*$ the most bias coin of ℓ-th level

- For each arriving arm_i in the stream do:
  (1) Read arm_i to memory.
  (2) **Aggressive Selective Promotion:** Starting from level $\ell = 1$:
    (a) Sample both arm_i and arm_ℓ^* for s_ℓ times. Drop arm_i if $\widehat{p}_{\text{arm}_i} < \widehat{p}_{\text{arm}_\ell^*}$, otherwise replace arm_ℓ^* with arm_i;
    (b) Increase C_ℓ by 1.
    (c) If $C_\ell = c_\ell$, send arm_ℓ^* to the next level by calling Line (2)a with $(\ell = \ell + 1)$.
  (3) Return arm_t^* as the selected most bias coin.

Lemma 6.1. *The sample complexity of the algorithm is $O(n \cdot \beta \cdot \log(\frac{1}{\delta})) = O(\frac{n}{\Delta^2} \cdot \log(1/\delta))$.*

The proof is almost identical to the proof of the $\log^*(n)$-space intermediate algorithm with minor parameter substitutions.

Lemma 6.2. *With probability at least $1-\delta$, the coin selected by the algorithm is an ε-best arm.*

We can prove the lemma by showing the 'boundedness' of the accumulated gap over each levels. Formally, we present:

Claim 6.3. *With probability at least $1-\delta$, at any level ℓ, there will be at least one arm with at most $\sum_{i=1}^{\ell} \varepsilon_i$ reward gap between the best arm.*

With Claim 6.3, for the event happens with probability at least $(1-\delta)$, the gap between every two layers are bounded. Accumulating the gap among every level and summing up will give us $\sum_{\ell=1}^{\lceil \log^*(n) \rceil + 1} \varepsilon_\ell \leq \frac{\varepsilon}{10} \sum_{\ell=1}^{\infty} \frac{1}{2^{\ell-1}} < \varepsilon$, which means the returned arm will have a cumulative gap of at most ε from the best arm.

ACKNOWLEDGMENTS

We would like to thank Arpit Agarwal, Sanjeev Khanna, Shay Moran, and David Woodruff for helpful discussions. We are also grateful to the anonymous reviewers of STOC 2020 for many helpful comments that helped with the presentation of this paper.

REFERENCES

[1] Arpit Agarwal, Shivani Agarwal, Sepehr Assadi, and Sanjeev Khanna. 2017. Learning with Limited Rounds of Adaptivity: Coin Tossing, Multi-Armed Bandits, and Ranking from Pairwise Comparisons. In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017.* 39–75.

[2] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, Nitin Motgi, Seung-Taek Park, Raghu Ramakrishnan, Scott Roy, and Joe Zachariah. 2008. Online Models for Content Optimization. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008.* 17–24.

[3] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. 2004. Approximating extent measures of points. *J. ACM* 51, 4 (2004), 606–635.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The space complexity of approximating the frequency moments. In *STOC*. ACM, 20–29.

[5] Sepehr Assadi and Chen Wang. 2020. Exploration with Limited Memory: Streaming Algorithms for Coin Tossing, Noisy Comparisons, and Multi-Armed Bandits. arXiv:2004.04666

[6] Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. 2010. Best Arm Identification in Multi-Armed Bandits. In *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010.* 41–53.

[7] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 2014. Streaming submodular maximization: massive data summarization on the fly. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014.* 671–680.

[8] Avrim Blum, Nika Haghtalab, Ariel D. Procaccia, and Mingda Qiao. 2017. Collaborative PAC Learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA.* 2392–2401.

[9] Mark Braverman, Jieming Mao, and S. Matthew Weinberg. 2016. Parallel algorithms for select and partition with noisy comparisons. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016.* 851–862.

[10] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. 2011. Pure exploration in finitely-armed and continuous-armed bandits. *Theor. Comput. Sci.* 412, 19 (2011), 1832–1852.

[11] Sébastien Bubeck, Tengyao Wang, and Nitin Viswanathan. 2013. Multiple Identifications in Multi-Armed Bandits. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013.* 258–265.

[12] Róbert Busa-Fekete, Balázs Szörényi, Weiwei Cheng, Paul Weng, and Eyke Hüllermeier. 2013. Top-k Selection based on Adaptive Sampling of Noisy Preferences. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013.* 1094–1102.

[13] Alexandra Carpentier and Andrea Locatelli. 2016. Tight (Lower) Bounds for the Fixed Budget Best Arm Identification Bandit Problem. In *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23-26, 2016*. 590–604.

[14] Arghya Roy Chaudhuri and Shivaram Kalyanakrishnan. 2019. Regret Minimisation in Multi-Armed Bandits Using Bounded Arm Memory. *CoRR* abs/1901.08387 (2019).

[15] Lijie Chen, Anupam Gupta, and Jian Li. 2016. Pure Exploration of Multi-armed Bandit Under Matroid Constraints. In *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23-26, 2016*. 647–669.

[16] Lijie Chen, Jian Li, and Mingda Qiao. 2017. Towards Instance Optimal Bounds for Best Arm Identification. In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*. 535–592.

[17] Shouyuan Chen, Tian Lin, Irwin King, Michael R. Lyu, and Wei Chen. 2014. Combinatorial Pure Exploration of Multi-Armed Bandits. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 379–387.

[18] Xi Chen, Paul N. Bennett, Kevyn Collins-Thompson, and Eric Horvitz. 2013. Pairwise ranking aggregation in a crowdsourced setting. In *Sixth ACM International Conference on Web Search and Data Mining, WSDM 2013, Rome, Italy, February 4-8, 2013*. 193–202.

[19] Xi Chen, Sivakanth Gopi, Jieming Mao, and Jon Schneider. 2017. Competitive analysis of the top-K ranking problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. 1245–1264.

[20] Xi Chen, Yuanzhi Li, and Jieming Mao. 2018. A Nearly Instance Optimal Algorithm for Top-k Ranking under the Multinomial Logit Model. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. 2504–2522.

[21] Yuxin Chen and Changho Suh. 2015. Spectral MLE: Top-K Rank Aggregation from Pairwise Comparisons. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. 371–380.

[22] Vincent Cohen-Addad, Frederik Mallmann-Trenn, and Claire Mathieu. 2020. Instance-Optimality in the Noisy Value-and Comparison-Model. In *In SODA 2020 (to appear)*.

[23] Thomas M Cover. 1968. A note on the two-armed bandit problem with finite memory. *Information and Control* 12, 5 (1968), 371–377.

[24] Susan B. Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. 2014. Top-k and Clustering with Noisy Comparisons. *ACM Trans. Database Syst.* 39, 4 (2014), 35:1–35:39.

[25] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. 2001. Rank aggregation methods for the Web. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*. 613–622.

[26] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. 2002. PAC Bounds for Multi-armed Bandit and Markov Decision Processes. In *Computational Learning Theory, 15th Annual Conference on Computational Learning Theory, COLT 2002, Sydney, Australia, July 8-10, 2002, Proceedings*. 255–270.

[27] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. 1994. Computing with Noisy Information. *SIAM J. Comput.* 23, 5 (1994), 1001–1018.

[28] Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. 1999. An Approximate L^1-Difference Algorithm for Massive Data Streams. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*. 501–511.

[29] Navin Goyal and Michael E. Saks. 2010. Rounds vs. Queries Tradeoff in Noisy Computation. *Theory of Computing* 6, 1 (2010), 113–134.

[30] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. 2000. Clustering Data Streams. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*. 359–366.

[31] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. 1998. Computing on data streams. In *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*. 107–118.

[32] Eshcar Hillel, Zohar Shay Karnin, Tomer Koren, Ronny Lempel, and Oren Somekh. 2013. Distributed Exploration in Multi-Armed Bandits. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 854–862.

[33] Kevin G. Jamieson, Matthew Malloy, Robert D. Nowak, and Sébastien Bubeck. 2014. lil' UCB : An Optimal Exploration Algorithm for Multi-Armed Bandits. In *Proceedings of The 27th Conference on Learning Theory, COLT 2014, Barcelona, Spain, June 13-15, 2014*. 423–439.

[34] Shivaram Kalyanakrishnan and Peter Stone. 2010. Efficient Selection of Multiple Bandit Arms: Theory and Practice. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*. 511–518.

[35] Shivaram Kalyanakrishnan, Ambuj Tewari, Peter Auer, and Peter Stone. 2012. PAC Subset Selection in Stochastic Multi-armed Bandits. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*.

[36] Zohar Shay Karnin, Tomer Koren, and Oren Somekh. 2013. Almost Optimal Exploration in Multi-Armed Bandits. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. 1238–1246.

[37] Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. 2016. On the Complexity of Best-Arm Identification in Multi-Armed Bandit Models. *J. Mach. Learn. Res.* 17 (2016), 1:1–1:42.

[38] Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. 2019. Submodular Streaming in All Its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 3311–3320.

[39] David Liau, Zhao Song, Eric Price, and Ger Yang. 2018. Stochastic Multi-armed Bandits in Constant Space. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*. 386–394.

[40] List of Open Problems [n.d.]. List of Open Problems in Sublinear Algorithms. Problem 73: Streaming Online Algorithms. https://sublinear.info/73.

[41] Shie Mannor and John N. Tsitsiklis. 2003. Lower Bounds on the Sample Complexity of Exploration in the Multi-armed Bandit Problem. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*. 418–432.

[42] Slobodan Mitrovic, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Volkan Cevher. 2017. Streaming Robust Submodular Maximization: A Partitioned Thresholding Approach. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 4557–4566.

[43] Robert H. Morris. 1978. Counting Large Numbers of Events in Small Registers. *Commun. ACM* 21, 10 (1978), 840–842.

[44] Ilan Newman. 2004. Computing in Fault Tolerance Broadcast Networks. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*. 113–122.

[45] Rüdiger Reischuk and Bernd Schmeltz. 1991. Reliable Computation with Noisy Circuits and Decision Trees-A General n log n Lower Bound. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*. 602–611.

[46] Herbert Robbins. 1952. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.* 58, 5 (1952), 527–535.

[47] Nihar B. Shah and Martin J. Wainwright. 2017. Simple, Robust and Optimal Ranking from Pairwise Comparisons. *J. Mach. Learn. Res.* 18 (2017), 199:1–199:38.

[48] Mohammad Sadegh Talebi, Zhenhua Zou, Richard Combes, Alexandre Proutière, and Mikael Johansson. 2018. Stochastic Online Shortest Path Routing: The Value of Feedback. *IEEE Trans. Automat. Contr.* 63, 4 (2018), 915–930.

[49] Chao Tao, Qin Zhang, and Yuan Zhou. 2019. Collaborative Learning with Limited Interaction: Tight Bounds for Distributed Exploration in Multi-Armed Bandits. In *In FOCS 2019 (to appear)*.

[50] Yuan Xue, Pan Zhou, Tao Jiang, Shiwen Mao, and Xiaolei Huang. 2016. Distributed learning for multi-channel selection in wireless network monitoring. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 1–9.

[51] Yuan Zhou, Xi Chen, and Jian Li. 2014. Optimal PAC Multiple Arm Identification with Applications to Crowdsourcing. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. 217–225.