Better Sliding Window Algorithms to Maximize Subadditive and Diversity Objectives

Michele Borassi
borassi@google.com
Google Research
Zurich, Switzerland

Alessandro Epasto
aepasto@google.com
Google Research
New York, NY, USA

Silvio Lattanzi
silviol@google.com
Google Research
Zurich, Switzerland

Sergei Vassilvitskii
sergeiv@google.com
Google Research
New York, NY, USA

Morteza Zadimoghaddam
zadim@google.com
Google Research
Zurich, Switzerland

ABSTRACT

The streaming computation model is a standard model for large-scale data analysis: the input arrives one element at a time, and the goal is to maintain an approximately optimal solution using only a constant, or, at worst, polylogarithmic space.

In practice, however, recency plays a large role, and one often wishes to consider only the last w elements that have arrived, the so-called *sliding window* problem. A trivial approach is to simply store the last w elements in a buffer; our goal is to develop algorithms with space and update time sublinear in w. In this regime, there are two frameworks: *exponential histograms* and *smooth histograms*, which can be used to obtain sliding window algorithms for families of functions satisfying certain properties.

Unfortunately, these frameworks have limitations and cannot always be applied directly. A prominent example is the problem of maximizing submodular function with cardinality constraints. While some of these difficulties can be rectified on a case-by-case basis, here, we describe an alternative approach to designing efficient sliding window algorithms for maximization problems. Then we instantiate this approach on a wide range of problems, yielding better algorithms for submodular function optimization, diversity optimization and general subadditive optimization. In doing so, we *improve* state-of-the art results obtained using problem-specific algorithms.

PODS'19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6227-6/19/06.
https://doi.org/10.1145/3294052.3319701

CCS CONCEPTS

• **Theory of computation → Streaming models**; **Approximation algorithms analysis**; **Streaming, sublinear and near linear time algorithms**; • **Mathematics of computing → Discrete optimization**.

ACM Reference Format:
Michele Borassi, Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. 2019. Better Sliding Window Algorithms to Maximize Subadditive and Diversity Objectives. In *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3294052.3319701

1 INTRODUCTION

Since their introduction in a seminal work by Alon et al. [2], data streams have emerged as a tool for processing large volumes of data. In a standard data stream setting, the input arrives one element at a time. The goal of the algorithm designer is to compute (or approximate) some function of the entire input in space *sublinear* in the input size. Classical results include estimating frequency moments [2]; estimating the number of distinct values [7, 25]; approximating important graph properties, such as graph cuts [1]; and preserving an approximately maximum matching [33]. A survey of applications is provided in [34].

While data stream algorithms have received a lot of well-deserved attention, they treat all the elements equally: in particular, there is no difference between an element that arrived at the beginning of the stream and one that arrived at the end. In practice, however, recent data is often far more important; one way to model this assumption is to consider only the last w elements of the stream during the computation. While w is typically much smaller than the total size of the stream (since the latter can be arbitrarily large), the goal is still to design algorithms with both space and update time sublinear in w. Note that sliding window problems are

a subset of dynamic stream problems [29, 34], where any element can be deleted at any time. Nonetheless, they form a challenging special case and have been studied extensively, see, for instance, [3, 4, 11, 13, 15, 16, 18, 26, 28, 31, 32, 36, 38].

Sliding Window Frameworks. To aid in designing sliding window algorithms, Datar et. al [23], and later Braverman and Ostrovsky [15] introduced techniques for special families of instances. Both of these, as well as the approach we propose in our work, rely on the existence of an efficient algorithm for the insertion-only version of the problem. The frameworks are discussed in detail in the related work section, here we introduce them briefly.

Exponential Histograms. Datar et al. [23] introduced the *exponential historgram* framework for estimating a non-negative set function f over a fixed-size sliding window. Their approach assumes that the function is weakly additive and that there exists a small space sketch that can be computed in an insertion-only stream. They show that if the sketch is composable, that is, an estimate of $f(A \cup B)$ can be obtained from the sketches of A and B, then polylogarithmically many sketches are sufficient to approximate the function over a sliding window.

Smooth Histograms. The work of Braverman and Ostrovsky [15] further generalized this approach. Instead of looking at composability, the authors consider *smooth* functions. At a high level, these are functions that do not experience sharp transitions from the addition (or deletion) of a single element, see [15] for a precise definition. Their key idea is to prove that for such smooth functions, one only needs to remember the value of the function for logarithmically many substreams over the course of any window. Such a result can be achieved by imposing that the difference between the values of the function on consecutive substreams be sufficiently large.

This approach is strictly more general than the exponential histogram of [23]; however, it also comes with some strong limitations. To the best of our knowledge, the biggest downside of smooth histograms is that to be directly applied, they require the ability to compute the function almost exactly, as a $(1-\epsilon)$-approximation of the function f leads to a $1-5\epsilon$ approximation of the value of the function over the sliding window. Thus, anything worse than a 0.8-approximation results in a trivial solution. In itself, this rules out their method for many important problems in which providing such an approximation is NP-hard. A prominent example is that of submodular function optimization subject to a cardinality constraint, where a $1-1/e$ approximation is the best possible.

This limitation can be overcome on a case-by-case basis, (see Section 2 below for examples), but doing so requires intricate reasoning in every case. In this work, we introduce a different approach to designing sliding window algorithms for maximization problems, and we apply it to diversity maximization and subadditive optimization.

Our Results. In order to solve maximization problems in the sliding window setting, we take a different perspective on the problem: instead of identifying a property of *functions* that makes them amenable to sliding window computations, we ask for an *algorithmic primitive* and prove that it is sufficient to turn any insertion-only algorithm into a sliding window algorithm. Then we show that this algorithmic primitive is easy to design for our problems.

Our proof is entirely constructive, so we obtain efficient sliding window algorithms for our problems. Briefly, our main observation is that it is possible to design an algorithm that, given two consecutive parts of the stream A and B, reduces the problem of producing an approximate solution for the substream represented by any suffix of the first summary, A concatenated with B with the additional assumption that the substream A itself does not contain a good approximation of the solution. This additional assumption significantly simplifies the design of sliding window algorithms for maximization problems. For example, under this assumption, the solution of the problem for a submodular function is trivial; in fact, it is enough to simply return the solution contained in B.

Our reduction algorithm is also simple. It is based on the observation that for every stream and every sliding window size, one can easily construct three substreams A, A^+, and B, so that a good solution to the sliding window problem is contained either in A^+, in B, or in a suffix of the first summary, A, and B.

Furthermore, we show that for many of the problems, our approach improves upon the hand-tuned implementations (Table 1 contains a summary of our results). For example, we give the first sliding window algorithms for a number of problems, such as the many variants of selection with diversity constraints problem, submodular maximization under p-matchoid constraints and, more generally, subadditive function maximization.

Roadmap. We start by reviewing related work in Section 2. Then we provide a formal formulation of our approach in Section 3, followed by specific applications. We apply the result to subadditive function optimization in Section 4.1, and in Section 4.2, we apply our maximization methods/sketches for selection with diversity constraints. Finally, we provide an even stronger version of our approach for submodular function optimization in Section 5.

Problem	Space (prev.)	Space (current)	Approx. (prev.)	Approx. (current)
Maximizing a subadditive function	-	$O(s \log w)$	-	$\alpha^2/2 - \epsilon$
Submodular maximization (card. const.)	$O(k \log^2 w)$	$O(k \log w)$	$1/3 - \epsilon$	$1/3 - \epsilon$
Submodular maximization (p-matchoid const.)	-	$O(k \log w)$	-	$O(1/p)$
Diversity maximization (linear instances)	-	$O(k \log^2 w)$	-	$\zeta/5 - \epsilon$
Diversity maximization (other instances)	-	$O(k^2 \log^2 w)$	-	$\zeta/5 - \epsilon$

Table 1: A summary of the results obtained in this paper. For simplicity, we assume that the ratio between the maximum and the minimum values is polynomial in the size w of the sliding window. We use n for the size of the stream. In subadditive maximization, we denote by α (resp., s) the approximation factor (resp., the space used) by a streaming algorithm. In diversity maximization, we denote by ζ the best approximation factor that a centralized algorithm can achieve.

2 RELATED WORK

For brevity, we restrict our literature review mostly to the sliding window streaming model of computation; we first review in detail the two main frameworks used to solve problems in the sliding window streaming model, then we list algorithms known for specific problems.

Sliding window model. The sliding window model of computation [23] is a powerful computational paradigm that allows one to naturally model data recency constraints in streaming settings. In this model, the algorithm receives a potentially unlimited stream of data points $x_1, x_2, \ldots$, and, at any point in time t, it is required to provide the result of a function f over the *active-window* W, i.e., the set of the latest w data points received (the points x_i for $i \in [\min(t-w, t)]$. This model encompasses multiple challenging data streaming problems and has received significant attention in the past [3, 4, 11, 13, 15, 16, 18, 26, 28, 31, 32, 36, 38].

We now review some of the main algorithmic techniques in this model. In their pioneering work on the sliding window problem, Datar et al. [23] and later Braverman and Ostrovsky [15] introduced two techniques that have been used successfully for providing low-space complexity algorithms for a multitude of problems in the sliding window setting.

Exponential histograms. Datar et al. [23] jump-started the study of the sliding window model in their seminal paper. They introduced *exponential histograms*, a general framework for estimating non-negative set functions f over a fixed-size sliding window. The framework assumes that the function is weakly additive and that there exists a low-space, composable sketch that can be computed in an insertion-only stream. More precisely, the authors assume that the function f is polynomially bounded by the size of the stream $f(A) \leq \text{poly}(|A|)$ and has the following weak-additivity properties: $f(A \cup B) \geq f(A) + f(B)$ and $f(A \cup B) \leq c_f\,(f(A) + f(B))$ for a constant $c_f \geq 1$. Finally, they assume that the function f admits low-space sketches that can be computed efficiently in insertion-only streams and can be *composed*, i.e., an estimate of $f(A \cup B)$ can be obtained from the sketches of A and B. Under these assumptions, they show that polylogarithmically many sketches are sufficient to approximate the function.

Smooth Histograms. Braverman and Ostrovsky [15] introduced the smooth histogram technique. The key idea is to preserve a subset of indices $t_1, t_2, \ldots, t_s$ of time in the stream such that one only needs to keep the value of the function f for the substream $[t_i, n]$, starting at one of these indices and ending with the last inserted item. Such an exact (or approximate) value can be computed using an (insertion-only) streaming algorithm. Braverman and Ostrovsky show that for any function f that respects certain smoothness assumptions, it is possible to keep $\text{poly}\log(w)$ many such indices and obtain a good approximation of f over the latest w items. The main idea is to look at the values of the function in the intervals mentioned above and ensure that for any pair of indices at distance 2, t_{i-1}, t_{i+1} the value of $f([t_{i+1}, t]) \leq (1-\beta) g([t_{i-1}, t])$ for some constant β, where $f([a, b])$ is the value of function f on the interval $[a, b]$ of the stream. For this method to work, the function f to be estimated needs to be polynomially bounded $f(A) \leq \text{poly}(|A|)$, monotone non-decreasing, and, crucially, (α, β)-smooth, a definition that we now recall.

Definition 2.1. *A function f is (α, β)-smooth if for all times $a < b < c < d$, we have:* $(1-\beta) f([a, c]) \leq f([b, c]) \implies (1-\alpha) f([a, d]) \leq f([b, d])$.

Braverman and Ovstrovsky then show that it is enough to maintain $O(\log_{1+\beta} M/m)$ such histogram indices, where M/m is the ratio of the max and min value of the function, to obtain a $1-\alpha$ approximation for any (α, β) smooth function f that can be computed exactly in the streaming setting. They also extend their results to functions f for which a $(1-\epsilon)$-approximation algorithm exists, in which case Theorems 2 and 3 in [15] show a $1 - 5\epsilon$-approximation algorithm in a sliding window. Notice that the smooth histogram provides a non-trivial result only if the function admits a good enough approximation ($\epsilon < 1/5$). Hence, it is not applicable

to many problems addressed in the paper (e.g., maximizing submodular functions).

The authors use the smooth histogram technique to provide multiple improved results, including better approximations of weakly-additive functions from [23], memory-optimal algorithms for frequency moments in L^p for $p > 0$ and algorithms to find the longest increasing sequence.

Recent Developments. Very recently, and concurrently with our work, two new generalizations of the smooth histogram approach have been introduced [10, 12].

In the first of these [12] Braverman et al. introduce the composable histogram method, which is an adaptation of the smooth histogram framework that leads to near optimal results for the heavy hitters problem. Similar to the smooth histogram framework, this approach applies to function estimation and requires the underlying function to be smooth. Additionally, it needs a good approximation algorithm for obtaining estimates of all suffixes on insertion-only streams.

In the second, [10], Braverman et al. specialize the smooth histogram approach to linear algebra questions by defining a notion of smooth functions for positive semidefinite matrices. They use the framework to obtain improved results for many problems, including spectral and graph sparsification and low-rank approximation.

These developments are elegant modifications of the smooth histogram framework tailored to specific classes of problems and are in contrast to the work presented here, which gives a new algorithmic primitive that leads to sliding window results for a wide range of classic maximization problems.

Our work is orthogonal to the presented frameworks and presents a simple technique for designing sliding window algorithms for maximization problems.

2.1 Specific Sliding Window Algorithms

In addition to the frameworks presented, there have been a number of ad hoc solutions modifying these frameworks for specific problems. We give a full overview below.

Sampling from sliding windows. Several authors have addressed the problem of sampling k items from the sliding window [4, 16]. In this context, the problem is made non-trivial by the implicit nature of element deletions in the model: the algorithm is not notified when an element expires and naively storing all elements' expiration times would require $\Omega(w)$ space. Babcock et al. [4] introduced the chain-sampling algorithm, which extends the reservoir sampling method to sample k items with space $O(k \log(w))$ for a window of fixed size w. They also show results for sliding windows of varying lengths. These results were later improved by Braverman et al. [16] who showed how to do this in optimal space $O(k)$ in the fixed size window model.

Heavy hitters and statistics for the sliding window. Several authors [3, 11, 28, 38] studied the problem of finding heavy-hitters in sliding window streams under the L^p norm, where one wants to find elements that appear more than a certain constant fraction of the L^p norm of the frequency vector of the sliding window. It is known that algorithms with $\operatorname{poly}\log(w)$ space are possible only for $p \leq 2$ because of a lower bound on computing the L^p norm in streams [35]. Several authors have provided results for the L^1 norm [3, 28, 38].

Braverman et al. [11] provided the first results for the L^2 norm which do not allow the use of the smooth histogram technique. This result implies a $\operatorname{poly}\log(w)$ algorithm for all $0 < p \leq 2$, closing the gap with the impossibility results. The authors introduce a *semi-smooth* algorithm technique that borrows in part from the smooth histogram method. In a related problem, Homem et al. [27] provided a data structure for maintaining the top-k most frequent elements in a sliding window stream. Recently, Basat et al. [8] provided upper and lower bounds on the memory needed to approximate sums over binary sliding window streams.

Sliding window over graphs. Crouch et al. [22] presented the first results in the sliding window model for graphs. In particular, they study a model where edges arrive one at a time and in which one wants to preserve properties for the graph induced by the sliding window of the latest edges to arrive. They provide algorithms for graph sparsifiers and approximate spanners and estimators of the size of the maximum matching and minimum spanning tree. More precisely, they show how to preserve an approximate spanner that allows one to estimate the distance of nodes up to a multiplicative factor and a $(1+\epsilon)$-sparsifier that almost exactly preserves edge cuts. Applications of these methods include verifying k-edge-connectivity in a graph or that the graph is bipartite. The results on matchings show a constant factor approximation that is based on a method similar to the smooth histogram technique. The authors show, in fact, that while the matching function is not smooth, it holds a smooth-like property.

Submodular maximization. Epasto et al. [24] gave a $(1/3 - \epsilon)$-approximation algorithm for arbitrary monotone submodular functions subject to cardinality constraints that requires $\tilde{O}(k \operatorname{poly}\log(w))$ space (for polynomially bounded functions). Submodular functions are not in general smooth, and they do not admit approximations $\geq 1 - \frac{1}{5}$, as required by the smooth histogram framework; however, the authors design an algorithm based on a variation of the smooth histogram technique for the problem. They also show a $1/2 - \epsilon$-approximation algorithm but with significantly more

space. Concurrently, Chen et al. [20] showed a $(1/4 - \epsilon)$-approximation $\tilde{O}(k \operatorname{poly}\log(w))$ space for the same classes of functions. They also show that the same method can be extended to arbitrary hereditary constraints and provide a $(\frac{1}{4p+(1+\epsilon)16p^2})$-approximation for p-matroid of rank k using $\tilde{O}(k \operatorname{poly}\log(w))$ space in the sliding window.

Clustering. The problem of clustering in sliding windows has also been studied [5, 14]. Braverman [14] shows results for the fundamental k-median and k-means problem in the metric and Euclidean cases of the sliding window model. They provide the first constant factor approximation algorithm that uses polylogarithmic space for metric k-median and metric k-means problems in the sliding window. The algorithm uses $O(k^3 \log^6(w))$ space, where k is the number of centers to output. Their work improves and addresses an open problem posed by the work of Babcock et al. [5], which designed an algorithm with space poly(w) . The algorithm is based on a core-set method and an extension of the smooth histogram technique.

In a related line of work, Cohen-Addad et al. [21] studied the problem of approximating the diameter and the k-center clustering in the sliding window model. They gave a $(3+\epsilon)$-approximation for the diameter problem using space $\log(\frac{M}{m})$, where M, m are the minimum and maximum distances in the space, respectively. For the k-center problem, they show a $(6+\epsilon)$-approximation $O(k \log\left(\frac{M}{n}\right))$ space.

Other applications. Beame et al. [9] provide multiple results in the sliding window model, including an algorithm for the distinctness problem.

Wang et al. [37] study the problem related to influence maximization. They cast the problem as a stream mining problem in which user actions are observed in real time and one wants to select a set of users with maximum influence over the most recent actions. The authors observe that the problem does not respect the requirements of the exponential and smooth histogram techniques. Then they introduce a technique based on keeping $O(\log(w))$ snapshots of the problem using an insertion-only algorithm that allows one to obtain a constant factor approximation to the problem.

3 A GENERAL APPROACH TO SOLVE MAXIMIZATION PROBLEMS

Definition 3.1 (Evaluation functions and monotonicity). *Let $\mathcal{X}$ and $\mathcal{Y}$ be the set of all inputs and solutions for an optimization problem. A solution $Y \in \mathcal{Y}$ is feasible for input $X \in \mathcal{X}$ if it satisfies the constraints of the optimization problem. Let $f : \mathcal{Y} \to \mathbb{R}^{\geq 0}$ be a function that evaluates the quality of a solution $Y \in \mathcal{Y}$. We focus on maximization problems. We say that the maximization problem represented by the function f is monotone if, for every two inputs $X \subset X'$, if Y is a feasible solution for X, it is also a feasible solution for X'.*

Notice that the monotonicity implies that for $X \subset X'$, we have $\text{OPT}^*(X) \leq \text{OPT}^*(X')$, where $\text{OPT}^*(X)$ is the value of the optimal solution in X. To simplify the notation, we allow $\mathcal{Y}$ to include non-feasible solutions and assume that $f(Y) = 0$ when Y is not a feasible solution.

In the paper, we assume that $w \geq 2$, otherwise, the problem is trivial. To summarize the data, we use a pair of functions (Z, z). The first function, Z, takes an input X and encodes it into a small summary $Z(X)$, while the second function, z, decodes $Z(X)$ and produces a feasible solution Y. This means that $f(z(Z(X))$ is the value of the solution found by the summary algorithm. We allow the summaries to be general data structures (we do not restrict their representations to be subsets of the input). We only need to assume that Z can be computed in insertion-only streams (i.e., it supports an insert operation), and z is a (polynomial-time) computable function of the summary.

To simplify the notation, we assume that $\mathcal{X}$ contains all possible summaries.

Definition 3.2 (Approximate summaries). *We say (Z, z) is a* summary *for any pair of functions $Z : \mathcal{X} \to \mathcal{X}$ and $z : \mathcal{X} \to \mathcal{Y}$. For a maximization problem f, we say the summary is α-approximate for X if:*

$$f(z(Z(X))) \geq \alpha \, \text{OPT}^*(X) \tag{1}$$

We say a summary is α-approximate (without specifying any particular X), if it is α-approximate for every $X \in \mathcal{X}$.

To allow for randomization in the summary algorithm, we say (Z, z) is a *randomized summary* if the aforementioned conditions hold on any input with high probability, where the probability is defined on the space of the random choices of the algorithm. Whenever clear from the context, we will call a randomized summary simply a summary.

Definition 3.3 (Space of a summary). *We denote the size of a summary by s_Z.*

We now present a technique that uses approximate summaries to compute approximately optimal solutions for maximization problems in the sliding window setting. The general intuition is to keep two consecutive approximate summaries for every possible guess of the optimum value, and then use the summaries to obtain a sliding window algorithm.

For simplicity, we assume that we have access to a streaming α-approximation summary, (Z, z), as well as the minimum, m, and maximum, M, value of a function f. Fix $\delta > 0$ and let $\Lambda = \{\lambda_1, \lambda_2, \ldots\}$ be a set of thresholds such that $\lambda_{i+1} = \lambda_i(1+\delta)$; we will specify the bounds of Λ later. For each threshold $\lambda \in \Lambda$, we maintain a pair of summaries, A_λ and B_λ, computed as follows.

Initially, each $A_\lambda = B_\lambda = \emptyset$, and the corresponding summaries are $Z(\emptyset)$. When an element x arrives, we check if the value of the solution computed by the summary on $B_\lambda \cup \{x\}$ remains smaller than λ. If so, we simply extend B_λ to $B_\lambda \cup \{x\}$, and update the corresponding summary. Otherwise, we reset A_λ to the current value of B_λ, and set $B_\lambda = \{x\}$. We show the pseudocode in Algorithm 1.

Function `Summaries`(X, λ)
  $Z(A_\lambda) \leftarrow Z(\emptyset)$; $Z(B_\lambda) \leftarrow Z(\emptyset)$;
  foreach $x \in X$ **do**
    if $f(z(Z(B_\lambda \cup \{x\}))) \leq \lambda$ **then**
      $Z(B_\lambda) \leftarrow Z(B_\lambda \cup \{x\})$;
    else
      $Z(A_\lambda) \leftarrow Z(B_\lambda)$; $Z(B_\lambda) \leftarrow Z(\{x\})$;
    end
  end
  return $(Z(A_\lambda), Z(B_\lambda))$

Algorithm 1: The high level algorithm for maintaining approximate summaries.

Now we observe some basic properties of our summaries that will be useful to show our main result.

Lemma 3.4. *For a set A_λ, let A_λ^+ be the set A_λ together with the first element of the corresponding B_λ. The following are invariants maintained by the algorithm.*
(i) A_λ and B_λ are two disjoint consecutive substreams of X,
(ii) A_λ precedes B_λ, and
(iii) B_λ ends with the current time t and always contains the last element of the stream.
(iv) When $|A_\lambda| \geq 1$:

$$f(z(Z(A_\lambda^+))) > \lambda. \tag{2}$$

(v) When $|A_\lambda| \geq 2$:

$$f(z(Z(A_\lambda))) \leq \lambda \tag{3}$$

(vi) When $|B_\lambda| \geq 2$:

$$f(z(Z(B_\lambda))) \leq \lambda \tag{4}$$

Proof. The first three points follow from construction. Equation 2 follows from the fact that A_λ is set to a non-empty summary only when $B_\lambda \cup \{x\}$ has value $> \lambda$, and this property is maintained thereafter. Equation 3 follows from the fact that $|A_\lambda| \geq 2$ implies that A_λ has been set as a copy of a prior B_λ that was not a singleton, and this happens only if B_λ has value $\leq \lambda$. Equation 4 follows from the same property. □

So far we have described how to maintain two summaries, A_λ and B_λ, with the properties in Lemma 3.4 for every threshold λ. Now we need to show how to use these two summaries to provide a solution for an active window.

To do so, we assume to have access to an additional function h that takes two summaries and an index, and produces a good summary for the stream starting at the index. Clearly, if we require h to work in every setting, designing such h would be equivalent to solving the initial problem. Interestingly, we can show that we need to use such a function only when A_λ contains a solution of "small" value.

More formally, consider a stream $x_0, x_1, x_2, \ldots, x_a, x_{a+1}, \ldots, x_b$. Let A be the sequence $x_0, \ldots, x_a$ and let B be the sequence $x_{a+1}, \ldots, x_b$. Let A_λ, B_λ be corresponding summaries. Finally, for $t \in [0, a]$, let A^t be the sequence $x_t, x_{t+1}, \ldots, x_a$.

Definition 3.5 (Mergeable summary for maximization problems). *Let f be a monotone function. We say that an α-approximate maximization problem summary (Z, z) is (α', β)-mergeable for $\beta \leq 1$, if there exists a composer function $h : \mathcal{X} \times \mathcal{X} \times \mathbb{Z} \to \mathcal{X}$ such that for any $A, B \in \mathcal{X}$ and for any $0 \leq t \leq |A|$, $H = h(Z(A), Z(B), t)$ verifies*
(i) $z(H)$ is a valid solution for $A^t \cup B$;
(ii) if $f(z(Z(A))) \leq \beta\,\mathrm{OPT}^*(A^t \cup B)$, then: $(h(Z(A), Z(B), t), z)$ is an α'-approximate summary for $A^t \cup B$.

Note that in (ii), with a slight abuse of notation, we are denoting with $h(Z(A), Z(B), t)$ the function that computes the sketch. Similarly to the previous case to allow for randomization in the summary algorithm we define a (α', β)-mergeable *randomized summary* if the aforementioned conditions hold on any input with high probability.

We note that we are not providing a general method of designing the composer function h. For every maximization problem, one needs to design a tailored function.

To simplify exposition, we use intervals of timestamps such as $[a, b]$ for $a, b \in \mathbb{N}$ to refer to the stream of items appearing between time a and b (inclusive) in order. We refer to W as the active window of size w, (i.e., for $t \in \mathbb{N}$, $W = [\max(t - w, 1), t]$).

Remark 3.6. *For some of the maximization problems such as subadditive or submodular maximization, the composer function h defined in Definition 3.5 is very simple. If the solution of the summary for the first sequence A is small enough compared to the optimum of the active window $A^t \cup B$, we can just discard the summary of A and output only the solution for the summary of B.*

However, for the diversity maximization problem, this simple trick will not work, and we need to provide more involved mergeable summaries and an actual composer function to obtain the desired approximation algorithms for sliding window setting.

Before presenting our algorithmic result, we pause to analyze the requirements of Definition 3.5 in terms of extracting solutions for suffixes of the first stream. Notice that we never require in our algorithms to compose with an empty set

$B = \emptyset$, however, the Definition 3.5 allows that. This does not imply that a mergeable summary is able to obtain a solution for any suffix A^t of A. The key here is that if $B = \emptyset$, we need to output a good approximation solution only if $f(A, z(Z(A))) \leq \beta\,\mathrm{OPT}^*(A^t)$, which makes the problem significantly simpler (sometimes trivial). For example, note that if $\beta < \alpha$, by monotonicity this condition never happens.

We now present our first algorithmic result. We show that, given a family of mergeable summaries, it is possible to obtain a sliding window algorithm that uses only logarithmically many summaries.

Let the set of thresholds be $\Lambda := \{\alpha m, (1+\delta)\alpha m, (1+\delta)^2\alpha m, \ldots (1+\delta)M\}$.

We run the algorithm presented in Algorithm 2.

Input The stream X, upper and lower bounds m, M; $\delta > 0$; the size of the window w.
Output Solution for the last active window W.
Phase 1: Consume the stream X in order, maintaining, in parallel, all the summaries.
$\Lambda \leftarrow \{\alpha m, (1+\delta)\alpha m, \ldots, (1+\delta)M\}$;
foreach $\lambda \in \Lambda$ **do**
  $Z(A_\lambda), Z(B_\lambda) \leftarrow \texttt{Summaries}(X, \lambda)$;
end
Phase 2: Extract the solution for the window W at the end of X.
$t_0 \leftarrow \max(|X| - w, 1)$;
$\lambda^* \leftarrow \max(\{\lambda : A_\lambda \subsetneq W \text{and } |A_\lambda| \geq 1\})$;
$S_1 \leftarrow z(Z(A^+_{\lambda^*}))$
$\lambda' \leftarrow \lambda^*(1+\delta)$;
if $W \cap A_{\lambda'} \neq \emptyset$ **then**
  $H \leftarrow h(Z(A_{\lambda'}), Z(B_{\lambda'}), t_0)$; $S_2 \leftarrow z(H)$;
else if $W = B_{\lambda'}$ **then**
  $S_2 \leftarrow z(Z(B_{\lambda'}))$;
else
  return S_1
end
return $\mathrm{argmax}_{S \in \{S_1, S_2\}} f(S)$;

Algorithm 2: The pseudocode of our main algorithm.

When asked for a solution for the current active window $\mathrm{W} = [t_0, t]$, where $t_0 = \max(t - w, 1)$, we find the maximum $\lambda^* \in \Lambda$ such that A_{λ^*} is a proper subset of the current active window and it is non-empty (we show that such a λ^* exists in Lemma 3.7). Let, $A^+_{\lambda^*}$ be A_{λ^*} combined with the first element of B_{λ^*}, and let $S_1 = z(Z(A^+_{\lambda^*}))$, be the candidate solution computed for λ^*.

Now consider $\lambda' = (1+\delta)\lambda^*$ (we show in Lemma 3.8 that $\lambda' \in \Lambda$): by construction, either $A_{\lambda'}$ is not initialized or $A_{\lambda'} \not\subset \mathrm{W}$ (the case $A_{\lambda'} = \mathrm{W}$ is excluded by construction). In either case, this implies that $\mathrm{W} \subseteq A_{\lambda'} \cup B_{\lambda'}$.

We now construct a solution S_2 as follows, depending on the particular position of W:
(i) If W intersects with $A_{\lambda'}$. Then, we have $\mathrm{W} = A^{t_0}_{\lambda'} \cup B_{\lambda'}$. We set $S_2 = h(Z(A_{\lambda'}), Z(B_{\lambda'}), t_0)$.
(ii) $\mathrm{W} = B_{\lambda'}$. Then we set $S_2 = z(Z(B_{\lambda'}))$.
(iii) W is a suffix of $B_{\lambda'}$. In this case we show that S_1 is a good solution, so we do not use S_2.
Finally, the algorithm outputs the solution of maximum value between S_1 and S_2.

Before proving our main result, we show that the algorithm above is well defined.

Lemma 3.7. *There exists a $\lambda^* \in \Lambda$ such that A_{λ^*} is non-empty and $A_{\lambda^*} \subsetneq W$.*

Proof. This statement is true for $\lambda = \alpha m$. Observe that the function has a value $\geq m$ on any non-empty subset of the stream by assumption (as m is the minimum value of the function); therefore, any α-approximate summary computed over any subset of the stream has value at least αm, so the value returned by the summary B_λ is at least αm every time the summary consumes one element of the stream. As a result, the algorithm resets A_λ and B_λ at every step, such that A_λ contains the second to last item observed, which is part of the sliding window by assumption of $w \geq 2$. Notice that this property continues to hold any other time the summary is reset. Finally, notice that the last element of the stream is always in B_λ, so A_λ is a proper subset of W. □

Lemma 3.8. *Let λ^* defined as in Algorithm 2, and $\lambda' = (1+\delta)\lambda^*$. Then $\lambda' \in \Lambda$.*

Proof. Clearly if $A_{\lambda^*} \subset \mathrm{W}$ and $|A_{\lambda^*}| \geq 1$, then $A^+_{\lambda^*} \subseteq \mathrm{W}$ as well. Moreover, by Equation 2 and monotonicity of f:

$$\lambda^* < f(z(Z(A^+_{\lambda^*}))) \tag{5}$$

Notice that this property cannot hold for $\lambda^* = M(1+\delta)$, as this would imply that $f(z(Z(A^+_{\lambda^*}))) > M(1+\delta) > \mathrm{OPT}^*(\mathrm{W})$. Therefore $\lambda' \in \Lambda$. □

We are now ready for the main result.

Theorem 3.9 (Sliding window algorithm for maximization problems). *Let (Z, z) be an α-approximate maximization problem summary and let it be (α', β)-mergeable for a monotone function f. Let h be its composer function. The algorithm described above computes a $\min(\alpha', \frac{\alpha\beta}{1+\delta})$-approximation to the maximization f in space $O\left(s_Z \log_{1+\delta} \frac{M}{\alpha m}\right)$.*

Proof. First we consider the quality of the solution obtained by S_1. Suppose that $\lambda^* \geq \frac{\alpha\beta}{1+\delta}\,\mathrm{OPT}^*(\mathrm{W})$. Then, Equation 5 implies that $S_1 = z(Z(A^+_{\lambda^*}))$ is an $\frac{\alpha\beta}{1+\delta}$ approximation to the optimum solution.

For the remainder of the proof, assume otherwise that $\lambda^* < \frac{\alpha\beta}{1+\delta}\,\mathrm{OPT}^*(\mathrm{W})$. This is equivalent to saying that $\lambda' <$

$\alpha\beta\,\mathrm{OPT}^*(\mathrm{W})$. We will show that in this case S_2 gives an $\min(\alpha, \alpha')$-approximate solution. We have to consider 3 cases:

(1) W intersects with $A_{\lambda'}$ and hence $\mathrm{W} \subseteq A_{\lambda'} \cup B_{\lambda'}$. Note that $|A'_\lambda| \geq 2$.[1] In this case we use the function h guaranteed by the mergeable summaries. We have: $\beta\,\mathrm{OPT}^*(\mathrm{W}) > \alpha\beta\,\mathrm{OPT}^*(\mathrm{W}) > \lambda' \geq f(z(Z(A_{\lambda'})))$ where the third inequality follows by the invariant of the algorithm. Therefore, the precondition for mergeable summaries holds, and the result is an α'-approximation.
(2) $\mathrm{W} = B_{\lambda'}$. In this case, considering $z(Z(B_{\lambda'}))$ will lead to an $\alpha \geq \frac{\alpha\beta}{1+\delta}$-approximation since Z is an α-approximate summary.
(3) $\mathrm{W} \subsetneq B_{\lambda'}$ Finally, we show that this case is not possible. Since $\lambda' < \alpha\beta\,\mathrm{OPT}^*(\mathrm{W})$, it must be that $\lambda' < \alpha\,\mathrm{OPT}^*(\mathrm{W})$. However, in this case, since Z is an α-approximate summary, we know that: $\alpha\,\mathrm{OPT}^*(\mathrm{W}) \leq f(z(Z(B_{\lambda'}))) \leq \lambda' < \alpha\,\mathrm{OPT}^*(\mathrm{W})$, which is a contradiction.

Therefore, the best of the two solutions is guaranteed to give at least a $\min(\alpha', \frac{\alpha\beta}{1+\delta})$-approximation.

To obtain the space bound, observe that there are $\log_{1+\delta}(\frac{M}{\alpha m})$ values of λ to consider, and for each of them our algorithm computes three sketches each of size less than s_Z. Hence, the total space complexity is $O(s_Z \log_{1+\delta} \frac{M}{\alpha m})$. □

4 EXAMPLES OF APPLICATIONS

We now apply our new technique to some prototypical problems.

4.1 Subadditive Function Maximization

In this subsection, we apply our new technique to subadditive optimization under downward closed constraints. Let V be a ground set: a set function $g : 2^V \to \mathfrak{R}$ is subadditive if $g(Y_1 \cup Y_2) \leq g(Y_1) + g(Y_2)$, $\forall Y_1, Y_2 \subseteq V$. A set system $\mathcal{I} \subset P(V)$, where $P(V)$ is the powerset of V, is downward closed iff $I \in \mathcal{I} \implies I' \in \mathcal{I}\ \forall I' \subset I$. We seek to maximize the function g subject to a downward closed constraint (i.e., we seek to find $\max_{I \in \mathcal{I}} g(I)$). A special and widely studied case of a downward closed set system is the cardinality constraint (i.e., $\mathcal{I} = \{I \subset V \,|\, |I| \leq k\}$ for some $k \in \mathcal{Z}^+$). Let g be a non-negative and subadditive set function whose non-zero co-domain is in $[m, M]$. We show that if there exists a α-approximate summary using space s_Z for maximizing g subject to a downward closed constraint then there exists a sliding window algorithm for maximizing g that gets a $\frac{\alpha^2}{2(1+\delta)}$-approximation using space $O(s_Z \log_{1+\delta} \frac{M}{m\alpha})$ for every constant $\delta > 0$.

[1]Note that $|A_{\lambda'}| \neq 1$, otherwise $A_{\lambda'} \subsetneq W$ which would contradict the definition of λ.

We reformulate the problem in the following natural way. We assume that the points from V arrive one at a time, and that at time t, we are allowed to output sets in $\mathcal{I} \cap P(\{x_1, \ldots, x_t\})$. Given a solution Y to the stream $X = \{x_1, \ldots, x_t\}$, we let $f(Y) = g(Y)$, if $Y \in \mathcal{I} \cap P(X)$, and $f(Y) = 0$ otherwise. Notice that f is non-negative and bounded in $[m, M] \cup \{0\}$. Notice also that this implies that the function f satisfies our monotonicity property, $\mathrm{OPT}^*(X') \leq \mathrm{OPT}^*(X')$ whenever $X \subseteq X'$, even if g is not monotone, simply by the monotonicity of $P(\{x_1, \ldots, x_t\})$.

Note that, for general subadditive functions, it is not clear how to compute an α-approximate summary, even in the specific case of cardinality constraints; however, whenever such a summary exists, we show that it is possible to modify it to get a $(\frac{\alpha}{2}, \frac{\alpha}{2})$ mergeable summary. We actually prove a slightly stronger lemma that works for any C-"weakly" subadditive function g; that is, any function such that $g(Y_1 \cup Y_2) \leq C(g(Y_1) + g(Y_2))$, $\forall Y_1, Y_2$ (the subadditive case is the specific case of $C = 1$). We say that f is C-weakly subadditive if it models a maximization problem for a function g that is C-weakly subadditive.

Lemma 4.1. *Let f be a bounded and C-weakly subadditive function subject to downward closed constraints. If there exists an α-approximate summary using space s_Z for maximizing f, then for the same problem, there exists also a $(\frac{\alpha}{2C}, \frac{\alpha}{2C})$-mergeable summary using the same space.*

Proof. Recall that function f has an α-approximate summary so we know that there exist (z, Z) such that $f(z(Z(X))) \geq \alpha\,\mathrm{OPT}^*(X)$, $\forall X \in \mathcal{X}$.

Now, fix $\beta = \frac{\alpha}{2C}$: we need to show that if $f(z(Z(A))) \leq \frac{\alpha}{2C}\,\mathrm{OPT}^*(A^t \cup B)$, then there exist h such that $(h(Z(A), Z(B), t), z)$ is an α'-approximate summary for $A^t \cup B$. We show that it is enough to define $h(Z(A), Z(B), t) = Z(B)$. In fact, $f(z(Z(B))) \geq \alpha\,\mathrm{OPT}^*(B) \geq \frac{\alpha}{C}(\mathrm{OPT}^*(A^t \cup B) - C\,\mathrm{OPT}^*(A^t)) \geq \frac{\alpha}{C}(\mathrm{OPT}^*(A^t \cup B) - C\,\mathrm{OPT}^*(A)) \geq \frac{\alpha}{C}\left(\mathrm{OPT}^*(A^t \cup B) - \frac{Cf(z(Z(A)))}{\alpha}\right)$ which is $\geq \frac{\alpha}{C}(\mathrm{OPT}^*(A^t \cup B) - \frac{1}{2}\,\mathrm{OPT}^*(A^t \cup B)) \geq \frac{\alpha}{2C}\,\mathrm{OPT}^*(A^t \cup B)$.

Furthermore, note that $(h(Z(A), Z(B), t), z)$ is a valid solution of $A^t \cup B$ for the downward closed property of the constraints.

So by fixing $h(Z(A), Z(B), t) = Z(B)$ we get a $(\frac{\alpha}{2C}, \frac{\alpha}{2C})$-mergeable summary. □

From Lemma 4.1 and Theorem 3.9 we obtain the following result:

Corollary 4.2. *Let f be a bounded and C-weakly subadditive function, if there exists an α-good summary using space s_Z for maximizing f under downward closed constraints, then for the same problem, there exists a sliding window algorithm that computes a $\frac{\alpha^2}{2C(1+\delta)}$-approximation for f that uses space $O(s_Z \log_{(1+\delta)} \frac{M}{m})$, for any constant $\delta > 0$.*

By using the streaming algorithm of Badanidiyuru et al. [6] that gives a $(\frac{1}{2}-\epsilon)$-approximate summary using space $O(k \log \frac{M}{m})$, we get the following corollary.

Corollary 4.3. *Let f be a monotone, bounded and submodular function subject to cardinality constraints. Then there exists a sliding window algorithm that computes a $\frac{1}{8(1+\delta)}$-approximation for f that uses space $O(k(\log_{(1+\delta)} \frac{M}{m})^2)$, for any constant $\delta > 0$.*

Similarly, it is possible to show a sliding window $O(1/p)$-approximation algorithm for non-monotone submodular optimization under p-Matroid constraint as in [20]. Using the streaming algorithm of Chekuri et al. [19], it is also possible to get a sliding window $O(1/p)$-approximation algorithm for non-monotone submodular optimization under the more general class of p-Matchoid constraints.

We note that the previous bound does not improve on previously known results [20, 24]. Nevertheless, in Section 5, we show how to strengthen the approach to get tighter bounds on a few maximization problems, including submodular optimization[2].

4.2 Diversity Maximization

In diversity maximization problems, we are given a stream X of points in a metric space, and we are asked to find a subset $Y \subset X$ of k points with maximum diversity. The diversity of a set Y, div(Y), depends on the distances between the points in Y. Indyk et al. [30] studied multiple variants of diversity each with its own div function, as explained in the second column of Table 2. We also list the approximation ratio and space of our new streaming and sliding window algorithms for each of these problems in Table 2 (note that all previously known streaming algorithms use space $\Omega(\sqrt{W})$ [30] or are limited to certain special cases like spaces with bounded doubling dimension [17]). In constructing the summaries, we sometimes need a centralized algorithm for solving the diversity maximization problems. Various constant approximation centralized algorithms exist for each of these problems in the literature; any of the algorithms can be used into our construction. The parameter ζ denotes the approximation factor of the centralized algorithm for the relevant diversity objective. The solution evaluation function fY) is defined, as before, to be div(Y) for a feasible solution Y of at most k points and 0 otherwise.

Diversity of a size k subset Y is always the sum of some number of distances, (i.e., $k(k-1)$ pairwise distances in Remote-clique, $k-1$ distances in Remote-tree, 1 distance in remote-edge, etc). To simplify the proofs, we normalize the diversity function, and instead consider the average distance.

Definition 4.4. *Let* $\text{div}'(Y)$ *be* $\frac{1}{n_k}\text{div}(X)$*, where n_k is the number of pairwise distances that are summed in* div*. In this section, we use $f(Y) = \text{div}'(Y)$ as our objective function for a feasible solution Y.*

Remark 4.5. *Any α-approximate solution Y for* div′ *is also an α-approximate solution for* div*, because the rescaling parameter is the same for Y and for the optimum. Hence, from now on, we only work with* div′*.*

We start by stating two useful properties and prove that our problems satisfy them.

(1) Given two sets of points $Y = \{y_1, \ldots, y_k\}$, and $Y' = \{y'_1, \ldots, y'_k\}$, if $\text{d}(y_i, y'_i) \leq \mu$ for each i, then $|f(Y) - f(Y')| \leq 2\mu$;
(2) Given $Y = \{y_1, \ldots, y_k\}$, if $\text{d}(y_i, y_j) \geq \mu$ for each $i < j$, then $f(Y) \geq \mu$;

Lemma 4.6. *All the diversity problems in Table 2 satisfy Properties 1 and 2.*

Proof. We focus on the mapping from Y to Y' that transforms each y_i to y'_i. Every point in this transformation moves by at most μ distance; therefore, by triangle inequality, the distance between any pair of points in Y is at most the distance of their projections in Y' minus 2μ. So the average distance of any subset of points in Y is also at most the average distance of the projected subset in Y' minus 2μ. Therefore, $f(Y') \geq f(Y) - 2\mu$. Similarly, one can show that $f(Y) \geq f(Y') - 2\mu$ which concludes the proof of Property 1.

The value of $f(Y)$ is the average of some subset of distances, and each distance is at least μ. Therefore, $f(Y)$ is also at least μ, which proves Property 2. □

4.2.1 *Solving remote-edge, remote-tree, remote-cycle, remote t-trees, remote t-cycles.*

Constructing approximate summary. Given lower and upper bounds m and M, we maintain a set of I_μ points with minimum pairwise distance of at least μ for each μ in $\{m, (1+\epsilon)m, \ldots, M\}$. We initialize each set I_μ to $\emptyset$ and process the stream X. For every new arriving point x and every μ, if there are fewer than k points in I_μ and the minimum distance of x to points in I_μ is at least μ, we insert x to I_μ.

This way, for each μ, either we have k points at distance μ from each other, and by Property 2 we find a solution with value at least μ, or we can move all points by at most μ so that they coincide with points of the sketch, then, by Property 1, we find a solution with value at least $\text{OPT}^*(X) - 2\mu$. By choosing the proper value of μ (as we will show in Lemma 4.7), in both cases we find a constant approximation of $\text{OPT}^*(X)$. So outputting the set I_μ with the highest diversity yields a

[2]Note that the algorithm presented in Section 5 is not problem-specific, but it is as general as the method presented in the previous section. We prefer to present the simpler method in the main body of the paper simply for clarity of exposition.

Name	Diversity Function	App. str	Space str	App. SW	Space SW
Remote-edge	$\min_{u,v\in Y} \mathrm{d}(u,v)$	$1/5-\epsilon$	$O(k\log\Delta)$	$\zeta/5-\epsilon$	$O(k\log^2\Delta)$
Remote-clique	$\sum_{u\neq v\in Y} \mathrm{d}(u,v)$	$\zeta/5-\epsilon$	$O(k^2\log\Delta)$	$\zeta/5-\epsilon$	$O(k^2\log^2\Delta)$
Remote-tree	weight of minimum spanning tree of Y	$1/5-\epsilon$	$O(k\log\Delta)$	$\zeta/5-\epsilon$	$O(k\log^2\Delta)$
Remote-cycle	weight of minimum TSP tour of Y	$1/5-\epsilon$	$O(k\log\Delta)$	$\zeta/5-\epsilon$	$O(k\log^2\Delta)$
Remote t-trees	minimum weight of t trees spanning Y	$1/5-\epsilon$	$O(k\log\Delta)$	$\zeta/5-\epsilon$	$O(k\log^2\Delta)$
Remote t-cycles	minimum weight of t cycles spanning Y	$1/5-\epsilon$	$O(k\log\Delta)$	$\zeta/5-\epsilon$	$O(k\log^2\Delta)$
Remote-star	$\min_{u\in Y}\sum_{v\in Y}\mathrm{d}(u,v)$	$\zeta/5-\epsilon$	$O(k^2\log\Delta)$	$\zeta/5-\epsilon$	$O(k^2\log^2\Delta)$
Remote-bipartition	$\min_{Y=Y_1\cup Y_2,\lvert Y_1\rvert=\lfloor\frac{k}{2}\rfloor}\sum_{u\in Y_1}\sum_{v\in Y_2}\mathrm{d}(u,v)$	$\zeta/5-\epsilon$	$O(k^2\log\Delta)$	$\zeta/5-\epsilon$	$O(k^2\log^2\Delta)$
Remote-pseudoforest	$\sum_{u\in Y}\min_{v\in Y-\{u\}}\mathrm{d}(u,v)$	$\zeta/5-\epsilon$	$O(k^2\log\Delta)$	$\zeta/5-\epsilon$	$O(k^2\log^2\Delta)$
Remote-matching	min weight of a perfect matching (for even k)	$\zeta/5-\epsilon$	$O(k^2\log\Delta)$	$\zeta/5-\epsilon$	$O(k^2\log^2\Delta)$

Table 2: The summary of results for Diversity Maximization problems under different metrics for diversity. Parameter ζ denotes the approximation factor of the offline algorithm for the relevant diversity objective and δ is the ratio between the minimum and the maximum distance in the dataset.

constant factor approximation for the diversity maximization problem in the streaming setting. Therefore, we can define the summary $Z(X)$ to be the union of all I_μ sets and the solution finder function $z(Z(X))$ to be the maximizer set I_μ to be $\mathrm{argmax}_{I_\mu} f(I_\mu)$.

Mergeable summary. We need to supplement our summary with a set of backup points to recover selected points in I_μ in case they expire (fall out of sliding window). We denote the ith point in I_μ by $P_{\mu,i}$. We aim to find a representative point $P'_{\mu,i}$ for $P_{\mu,i}$ such that the distance between them is at most μ. We also want the representative points of set I_μ to be distinct, (i.e., we need $|I_\mu|$ distinct representative points for points in I_μ). However, it is possible for a point P' to represent two points from two different sets, I_μ and $I_{\mu'}$.

We find and maintain representative points as follows. Whenever a new point x is added to set I_μ, we make x its own representative point. Otherwise, we check to see whether x is at distance of at most μ to any point in I_μ. If such a point $P_{\mu,i}$ exists, we make x the new representative point of $P_{\mu,i}$; i.e., $P'_{\mu,i}$ is set to x (if there are multiple options, we select one arbitrarily).

Now, let us define the function h used to compose two summaries $Z(A)$ and $Z(B)$ of two streams given time t (start of the active window). For each μ, the summaries of A and B each have a I_μ set. We call them $I_\mu(A)$ and $I_\mu(B)$. Let $I'_\mu(A)$ and $I'_\mu(B)$ be the set of representative points for these two sets. We define H_μ to be the union of $I_\mu(B)$ and the representative points of $I'_\mu(A)$ that arrive at or after t. Clearly, all points in H_μ are in the active window. Let ALG be some centralized ζ-approximation algorithm for our diversity problem. Function h returns $\mathsf{ALG}(H_{\mu^*})$, where $\mathsf{ALG}(X)$ is the solution of ALG on input X, and $\mu^* = \mathrm{argmax}_\mu f(\mathsf{ALG}(H_\mu))$.

Lemma 4.7. *The summary (Z,z) defined above is $(\frac{1}{5}-\epsilon)$-approximate. Moreover, it is $(\frac{\zeta}{5}-\epsilon,\frac{1}{5}-\epsilon)$-mergeable when accompanied by the composer function h defined above. Here, ζ is the approximation factor of the centralized algorithm* ALG *used to construct the function h.*

Proof. We start by proving the $(\frac{1}{5}-\epsilon)$-approximation for an arbitrary stream X. Recall that we refer by $\mathrm{OPT}^*(X)$ to the value of the optimum on X and by $Y(X)$ to the optimal solution. The set of μ values that we try ensures that there exists a μ' such that $\frac{1}{5}\mathrm{OPT}^*(X) \le \mu' \le \frac{1}{5}(1+\epsilon)\mathrm{OPT}^*(X)$, for which we maintain set $I_{\mu'}$. If there are k points in $I_{\mu'}$, this set itself has k points, and its diversity is at least μ' by Property 2. Therefore, we have a $\frac{1}{5}$ approximation by applying $\mu' \ge \frac{1}{5}\mathrm{OPT}^*(X)$.

Otherwise, $I_{\mu'}$ has fewer than k points, which means that every point in the stream has distance at most μ' to some point in $I_{\mu'}$. Let OPT be the optimum solution of stream X. We map every point in OPT to its closest point in $I_{\mu'}$. Note that this is a many-to-one mapping. Let OPT$'$ be the multi-set of projected points. By Property 1, $f(\mathrm{OPT}')$ is at least $f(\mathrm{OPT}(X))-2\mu'$. We want to upper bound $f(\mathrm{OPT}')$ in terms of $f(I_{\mu'})$.

For remote-edge, if there are duplicate points in OPT$'$, $f(\mathrm{OPT}')$ is zero, which makes the claim trivial. Otherwise, OPT$'$ should be the same as $I_{\mu'}$, as they both have k distinct points, and the former is a subset of the latter.

For the other objectives (remote-tree, remote-cycle, remote t-trees, remote t-cycles), we remove duplicate points of OPT$'$ to achieve the set OPT$''$. This operation does not decrease the diversity, so $f(\mathrm{OPT}'')$ is also at least $f(\mathrm{OPT}(X))-2\mu'$. On the other hand, OPT$''$ is a subset of $I_{\mu'}$. Therefore, $f(\mathrm{OPT}'')$ is at most $2f(I_{\mu'})$, since any tree or tour spanning a set of points Q can be transformed to a tree or tour spanning $Q' \subseteq Q$ with total distance of, at most, twice the original one. This can be done by doubling all the edges in the tree or tour and traversing over the Eulerian tour of the doubled set of edges. We conclude that: $f(I_{\mu'}) \ge \frac{f(\mathrm{OPT}'')}{2} \ge \frac{f(\mathrm{OPT}')}{2}$, which is, $\ge \frac{f(\mathrm{OPT}(X))}{2} - \mu' \ge (0.3-\epsilon)\mathrm{OPT}^*(X)$, which concludes the first claim of this lemma.

For mergeability, we show that, whenever $f(z(Z(A))) \leq (\frac{1}{5} - \epsilon)\,\mathrm{OPT}^*(A^t \cup B)$, we have $f(h(Z(A), Z(B), t)) \geq (\frac{1}{5} - \epsilon)\,\mathrm{OPT}^*(A^t \cup B)$. We know that the solution produced by composer function $h(Z(A), Z(B), t)$ is $\mathrm{ALG}(H_{\mu^*})$, where $\mu^* = \mathrm{argmax}_\mu\, f(\mathrm{ALG}(H_\mu))$. We note that in this particular case, composer function h returns a feasible solution, and therefore we can directly evaluate it in the term $f(h(Z(A), Z(B), t))$ without applying a decoder z.

The solution $z(Z(A))$ is the solution of summary on A which is the maximizer of $\max_\mu f(I_\mu(A))$. In particular, $f(z(Z(A)))$ is at least $f(I_{\mu''}(A))$, where μ'' is chosen such that $\frac{1}{5}\,\mathrm{OPT}^*(A^t \cup B) \leq \mu'' \leq \frac{1}{5}(1+\epsilon)\,\mathrm{OPT}^*(A^t \cup B)$, and $\mathrm{OPT}^*(A^t \cup B)$ is the optimum value of the active window $A^t \cup B$.

Since $f(z(Z(A))) \leq (\frac{1}{5} - \epsilon)\,\mathrm{OPT}^*(A^t \cup B)$, we also have $f(I_{\mu''}(A)) \leq (\frac{1}{5} - \epsilon)\,\mathrm{OPT}^*(A^t \cup B)$. Therefore, $I_{\mu''}(A)$ cannot have k points, otherwise, its value will be at least μ'', contradicting the upper bound on it by definition of μ''.

We now lower bound the value of the composed solution, $f(\mathrm{ALG}(H_{\mu^*}))$. By definition of μ^* and approximation factor of ALG, we have $f(\mathrm{ALG}(H_{\mu^*})) \geq f(\mathrm{ALG}(H_{\mu''}) \geq \zeta\,\mathrm{OPT}^*(H_{\mu''})$. So, we focus on $\mathrm{OPT}^*(H_{\mu''})$.

If $I_{\mu''}(B)$ has k points, $f(I_{\mu''}(B))$ will be at least $\mu'' \geq \frac{\mathrm{OPT}^*(A^t \cup B)}{5}$. The claim is proved by noting that $I_{\mu''}(B)$ is a subset of $H_{\mu''}$, and therefore $\mathrm{OPT}^*(H_{\mu''}) \geq f(I_{\mu''}(B))$.

Now, we can focus on the case that $I_{\mu''}(A)$ and $I_{\mu''}(B)$ have strictly fewer than k points. We show that every point x in the active window has distance of at most $2\mu''$ to some point in $H_{\mu''}$ (the union of $I_{\mu''}(B)$ and points in $I'_{\mu''}(A)$ that arrive at or after t). Since $I_{\mu''}(B)$ has fewer than k points, every point in B has distance of at most μ'' to some point in $I_{\mu''}(B)$. For a point $x \in A^t$, we have two options. Either x is in $I_{\mu''}(A)$, which means its representative is present in $H_{\mu''}$. Or, if x was not added to $I_{\mu''}(A)$, it was because x had distance of at most μ'' to some point in $y \in I_{\mu''}(A)$. The representative of y is present in $H_{\mu''}$ and has distance of at most $2\mu''$ to x by triangle inequality.

Therefore, we can move the points of optimum solution of $A^t \cup B$ each by at most $2\mu''$ such that they are all present in $H_{\mu''}$. This transformation cannot decrease the value of optimum by more than $4\mu''$ (similar to the first part of the proof). We conclude that $\mathrm{OPT}^*(H_{\mu''})$ is at least $\mathrm{OPT}^*(A^t \cup B) - 4\mu'' \geq \frac{1-4-4\epsilon}{5}\,\mathrm{OPT}^*(A^t \cup B)$. Combining this with the centralized approximation factor ζ yields an overall approximation of at least $\frac{\zeta}{5} - \epsilon$, completing the proof. □

Corollary 4.8. *There is a $\frac{\zeta}{5}$-approximation algorithm in sliding window for remote-edge, remote-tree, remote-cycle, remote t-trees, remote t-cycles. The space used is $O(k \log^2 \Delta)$.*

4.2.2 Solving remote-clique, remote-star, remote-bipartition, remote-pseudoforest, remote-matching. We design a summary that is very similar to the one in Subsection 4.2.1. The main difference is in the number of representative points we keep. Instead of keeping just one representative point for selected point $P_{\mu,i}$, we keep up to k of them.

Constructing approximate summary. Like before, we maintain a set I_μ of at most k points for each μ. Whenever a new point x arrives, if its distance to points I_μ is at least μ, we add it to I_μ. Otherwise, we find the first (based on arrival time) point $y \in I_\mu$ whose distance to x is at most μ. We add x to the set of representatives of y, and if there are already k representatives for y, we delete the most outdated (with the earliest arrival time) representative and replace it with x. Like before, for a fixed μ, the representative sets are disjoint. The summary Z consists of all sets I_μ and the representatives of their points. The solution finder function z returns the maximizer of $\max_\mu f(I_\mu)$.

Mergeable summary. We define H_μ to be the union of $I_\mu(B)$ and all representative points $I'_\mu(A)$ that arrive at or after time t. Function h runs some ζ-approximation centralized algorithm ALG on each H_μ and returns the highest diversity solution (the maximizer of $max_\mu f(\mathrm{ALG}(H_\mu))$). We note that with this definition of composer function h, it returns a feasible solution, and therefore we can directly evaluate it via function f without applying any decoder z.

Lemma 4.9. *The summary (Z, z) defined above is $(\frac{\zeta}{5} - \epsilon)$-approximate for diversity maximization problems: remote-clique, remote-star, remote-bipartition, remote-pseudoforest, and remote-matching. Moreover, the summary (Z, z) is $(\frac{\zeta}{5} - \epsilon, \frac{1}{5} - \epsilon)$-mergeable for these problems when accompanied with the composer function h and centralized algorithm* ALG *defined above. Here, ζ is the approximation factor of centralized algorithm* ALG.

Proof. The proof of the first claim is identical to the first part of the proof of Lemma 4.7. Recall that the second claim is essentially proving $f(h(Z(A), Z(B), t)) \geq (\frac{1}{5} - \epsilon)\,\mathrm{OPT}^*(A^t \cup B)$ whenever we have $f(z(Z(A)) \leq (\frac{1}{5} - \epsilon)\,\mathrm{OPT}^*(A^t \cup B)$.

Let μ be such that $\frac{1}{5}\,\mathrm{OPT}^*(A^t \cup B) \leq \mu \leq \frac{1}{5}(1+\epsilon)\,\mathrm{OPT}^*(A^t \cup B)$. Identical to the proof of Lemma 4.7, we can assume that $I_\mu(A)$ and $I_\mu(B)$ both have strictly fewer than k points.

We assign points in the optimum solution of $A^t \cup B$ to selected centers in $I_\mu(A)$ and $I_\mu(B)$. Each optimum point x that is in A as well is assigned to the first (earliest arrival time) point in $I_\mu(A)$ with distance of at most μ from x. We perform a similar assignment of optimum points in B.

If L optimum points are assigned to a point y in $I_\mu(A)$ (the claim for $I_\mu(B)$ is identical), all these L points were valid candidates of being a representative for y. Therefore, y has at least L representative points in the active window. So we can map each optimum point in x to a distinct point in the summary H_μ with a distance of at most 2μ. Similar to

the proof of Lemma 4.7, the rest of the claim follows by a) property 1, b) the fact that diversity of any size k subset of H_μ is a lower bound on its optimum value $\text{OPT}^*(H_\mu)$, and c) ALG is a ζ approximation algorithm. □

5 REFINED APPROACH FOR MAXIMIZATION

We now present a stronger, more complex technique for maximization problems that allows for tighter bounds. The main goal of this section is to show that it is possible to trade-off space at the cost of some additional complexity. In the refined method presented in this section, we guess the value of the optimum solution and then run a copy of the summary algorithms for each guess, λ. In this way, we can reduce the total space used by our algorithms by a $\log \Delta$ factor.

The main idea behind the refined approach is to allow the algorithm to select the optimal λ to optimize the approximation factor and to reduce the memory used by a logarithmic factor. In fact, toward the end of the section, we will show that the new approach can be used to improve the state-of-the-art algorithm for submodular maximization under cardinality constraints.

In this section, we use the same notation as before for the evaluation function f encoding the maximization problem and its constraints. We assume that the function is monotone, as defined above.

Definition 5.1 (Good summary for maximization problems). *Fix $\lambda \in \Lambda$. Let $Z_\lambda : \mathcal{X} \to \mathcal{X}$ and $z_\lambda : \mathcal{X} \to \mathcal{Y}$. A summary is $(\lambda, \alpha, \delta)$-good if:*

Whenever there exists a $X' \subseteq X$ and $Y \in \mathcal{Y}$, such that Y is a feasible solution of X', with value $\lambda \in [f(Y), (1+\delta)f(Y)]$; then we have:

$$f(z_\lambda(Z_\lambda(X))) \geq \alpha\lambda.$$

The key to our definition is that the quality of the solution only needs to be approximately optimal w.r.t. λ when there is a solution of value comparable to λ in any subset of X. Note that we do not require any guarantee on the quality of the solution returned by the summary when the value of λ is significantly different from that of any feasible solution of any subset of X.

Definition 5.2 (Good family of summaries for maximization problems). *Let $(Z, z)_\lambda$ be a family of summaries for a maximization problem f for every $\lambda \in \mathbb{R}^{\geq 0}$. We say that the family is an (α, δ) good family of summaries for the problem f if for every λ and for a stream X, (Z_λ, z_λ) is a $(\lambda, \alpha, \delta)$ good summary for X.*

We now give the adapted definition of *strong mergeable summaries*.

Definition 5.3 (Strong mergeable summary). *We say that a family of (α, δ) good summaries is $(\alpha', \delta', \beta)$-mergeable if there exists a composition function $h : \mathcal{X} \times \mathcal{X} \times \mathbb{Z} \to \mathcal{X}$ such that for any λ, A, B, t, and $H = h(A_\lambda, B_\lambda, t)$: if $f(z_\lambda(A_\lambda)) \leq \beta \cdot \lambda$, then H is a $(\lambda, \alpha', \delta')$ good summary of $A^t \cup B$.*

Also in this case, while we do not explicitly forbid the composition with $B_\lambda = \emptyset$, the definition does not require us to be able to extract arbitrary suffixes. In fact, if $\alpha > \beta$, it is easy to see that by applying the composition with an empty set, then the condition $f(z_\lambda(A_\lambda)) \leq \beta \cdot \lambda$ is not satisfied whenever λ is close to the value of a feasible solution of A^t.

We now show that, given a family of mergeable summaries, it is possible to obtain a sliding window algorithm that uses only logarithmically many summaries. Also in this algorithm, for every λ, we will keep only three summaries for intervals A_λ, A^+_λ, and B_λ.

Theorem 5.4 (Tighter sliding window algorithm for maximization problems). *Let $(Z, z)_\lambda$ be a (α, δ)-good family of summaries and let $(Z, z)_\lambda$ be $(\alpha', \delta', \beta)$-mergeable where $\beta < \alpha$. Let $\delta_0 = \min(\delta, \delta')$ and let $\alpha_0 = \min(\alpha', \beta)$. There exists a sliding window algorithm that given m, M and the stream X preserves a α_0-approximation of a maximization problem f in space $O\left(s_Z \cdot \log_{1+\delta_0} \frac{M}{m}\right)$.*

Proof. As before, let $\Lambda' := \{m, (1+\delta_0)m, (1+\delta_0)^2 m, \ldots, M(1+\delta_0)\}$. For each $\lambda \in \Lambda'$, we maintain at all time $(\lambda, \alpha, \delta)$-good summaries A_λ, A^+_λ and B_λ almost as in Algorithm 1. At any time we maintain the following invariants: A_λ and B_λ are intervals for which we keep summaries that have processed, respectively, the items from the stream appearing in two disjoint consecutive intervals of timesteps $A_\lambda = [t_{\lambda,0}, t_{\lambda,1}]$ and $B_\lambda = [t_{\lambda,1}+1, t]$ with the second interval ending with the last item (the first interval may be empty, in which case the associated summary $Z(A_\lambda)$ is empty). Moreover, we preserve at all times that $f(z(Z(A_\lambda))) \leq \beta\lambda$ (unless A_λ has size one and it contains a single element of value $> \beta\lambda$) and either $f(z(Z(B_\lambda))) \leq \beta\lambda$ or the second interval contains a single element of value $> \beta\lambda$. Finally, unless the first interval is empty, we ensure that $f(z(Z(A^+_\lambda))$ is larger than $\beta\lambda$. Notice that it is always possible to preserve the previous invariants using Algorithm 1 by substituting $\leq \lambda$ with $\leq \beta\lambda$ in the condition of the first if statement at line 4.

Now, whenever we are asked for a solution for the current active window $W = [\max(t-w, 1), t]$, we compute a series of pairs of candidate solutions Y_λ and candidate summaries X_λ, one for each $\lambda \in \Lambda'$ that respects some properties outlined below, and later we output the one with larger value among the ones computed. For each λ, we have four cases, depending on the position of the intervals associated to λ:

- Case 1: The active window is exactly coincident with the second interval. We set $X_\lambda = Z(B_\lambda)$ and set $Y_\lambda = z_\lambda(X_\lambda)$.
- Case 2: The two intervals are both non-empty subsets of the active window. Then we use $X_\lambda = A^+_\lambda$, and $z(Z(A^+_\lambda))$ as a solution.
- Case 3: The active window intersects both intervals, and the first interval contains elements not in the active window. We use h to obtain a good solution by analyzing the summary $X_\lambda = h(Z(A_\lambda), Z(B_\lambda), t)$ and its associated solution.
- Case 4: Finally, if the active window is strictly contained in the second interval, we ignore this λ (we will show later it is not needed), and Y_λ is not set.

Let λ^* be such that $\lambda^* \in [\mathrm{OPT}^*(\mathrm{W}), (1+\delta_0)\,\mathrm{OPT}^*(\mathrm{W})] \cap \Lambda'$; we know that such lambda exists by construction. The main idea of the proof is to lower-bound the value of $f(Y_{\lambda^*})$. We now show that the solution output is an α_0-approximation. We consider the four cases.

In case 1, we provide an $\alpha \geq \alpha_0$-approximation, as we are using a $(\lambda^*, \delta, \alpha)$-good summary on W, so there exists a solution of value close to λ^*, and the summary will output a solution of value at least $\alpha\lambda^* \geq \alpha_0\,\mathrm{OPT}^*(W)$.

Now consider case 2. Here, by construction we know that $f(Y_{\lambda^*}) \geq \beta\lambda^* \geq \alpha_0\,\mathrm{OPT}^*(W)$.

For case 3, it is enough to notice that we are composing a $(\lambda^*, \alpha, \delta)$-good summary with value $\leq \beta\lambda^*$ with another $(\lambda^*, \alpha, \delta)$-good summary and a start time inside the first summary interval corresponding to the beginning of the active window. Hence the output summary X^*_λ is a $(\lambda^*, \alpha', \delta')$-good summary of the active window $[\max(t-w, 1), t]$. So again we get an α_0-approximation.

For case 4, we show that this cannot happen for λ^* as $\beta < \alpha$. In fact, in this case we know that B_{λ^*} is a $(\lambda^*, \alpha, \delta)$-good summary with value $\leq \beta\lambda^*$. But B_{λ^*} needs to output a solution of value $\geq \alpha\lambda^* > \beta\lambda^*$ when computed over a superset of W, so this gives a contradiction.

Finally, note that our algorithm requires us to compute $O(\log_{1+\delta_0}(\frac{M}{m}))$ many summaries, each of size less than s_Z, so the total space is $O(s_Z \log_{1+\delta_0} \frac{M}{m})$. □

Note that the summaries constructed are independent of the size of the active window (in contrast, for example, to the smooth histogram technique). Only the post-processing uses the value w to obtain the solution. This means that the algorithm preserves an approximate solution for any window size at all time.

Now, we explore an applications of our more refined approach for *submodular maximization with cardinality constraints*. As in Section 4.1, we write $f(Y) = g(Y)$ for a submodular function g whenever Y is a valid solution (i.e., in the case of a cardinality constraint k, when $|Y| \leq k$). Recall that a function g is submodular if for each $X_1 \subseteq X_2$ and for each $x \in X \setminus X_2$, $g(X_1 \cup \{x\}) - g(X_1) \geq g(X_2 \cup \{x\}) - g(X_2)$. Since f is determined by g, as long as the solution returned is feasible, to simplify the notation, we are concerned with optimizing g.

In the following, we improve all previous results by defining a $(\frac{1}{3}-\delta, \delta, \frac{1}{3}-\delta)$-mergeable summary that uses space $O(k)$ for monotone submodular maximization under cardinality constraints. Consequently, we obtain a $(\frac{1}{3}-\delta)$-approximation in sliding window that uses space $O(k \log_{1+\delta} \frac{M}{n})$.

Our summary function Z_λ maps a stream of elements X into a feasible solution Y_λ (i.e., a subset of size $\leq k$). When we receive the next element x_t, we add it to the current solution Y_λ if Y_λ has $< k$ elements and $g(Y_\lambda \cup \{x_t\}) - g(Y_\lambda) > c\frac{\lambda}{k}$ (c will be chosen later). The function z_λ is simply the identity, and will therefore be omitted in the rest of the proof. We now show our main lemma (whose proof is similar in spirit to the proof in [24])

Lemma 5.5. *The summary Z_λ is $(\lambda, \min(c, 1-c(1+\delta)), \delta)$-good.*

Proof. Suppose there exist $X' \subseteq X$ and $Y \in \mathcal{Y}$ such that $\lambda \in [f(Y), (1+\delta)f(Y)]$ and Y is a valid solution for X', otherwise the proof is trivial. If our summary contains k elements, then $g(Y_\lambda) \geq c\lambda$. Otherwise, let $Y = \{y_1, \ldots, y_\ell\}$, where $\ell \leq k$: for each y_i, either $y_i \in Y_\lambda$ or, when we processed y_i, the old solution Y'_λ verified $g(Y'_\lambda \cup \{y_i\}) - g(Y'_\lambda) < c\frac{\lambda}{k}$. Since $Y'_\lambda \subseteq Y_\lambda$, by submodularity, this means that $g(Y_\lambda \cup \{y_1, \ldots, y_{i-1}\} \cup \{y_i\}) - g(Y_\lambda \cup \{y_1, \ldots, y_{i-1}\}) < c\frac{\lambda}{k}$.

We conclude that:

$$\begin{aligned} g(Y) &\leq g(Y_\lambda \cup Y) \\ &= g(Y_\lambda) + \sum_{i=1}^{\ell} g(Y_\lambda \cup \{y_1, \ldots, y_i\}) - g(Y_\lambda \cup \{y_1, \ldots, y_{i-1}\}) \\ &\leq g(Y_\lambda) + c\lambda \\ &\leq g(Y_\lambda) + c(1+\delta)g(Y) \end{aligned}$$

Hence, $g(Y_\lambda) \geq g(Y)(1-c(1+\delta)) \geq \lambda(1-c(1+\delta))$. □

Lemma 5.6. *The family of summaries $\{Z_\lambda\}$ is $(\min(c, 1-2c(1+\delta)), \delta, \frac{c}{1+\delta})$-mergeable.*

Proof. Let A_λ be the summary of A and B_λ be the summary of B, we define $h(A_\lambda, B_\lambda, t) = B_\lambda$. We need to prove that for any λ, A, B, t, if $g(A_\lambda) \leq \frac{c}{1+\delta}\lambda$, then B_λ is a $(\lambda, \alpha', \delta)$-good summary with $\alpha' = \min(c, 1-2c(1+\delta))$.

In other words, if there exist $X' \subseteq A^t \cup B$ and $Y \in \mathcal{Y}$ such that Y is a feasible solution of X' and $\lambda \in [f(Y), (1+\delta)f(Y)]$, we want to show that $g(B_\lambda) \geq \min(c, 1-2c(1+\delta))\lambda$. Now, suppose the condition is true, as the other case is trivial.

First, if B_λ has at least k elements, $g(B_\lambda) \geq c\lambda$, and we are done. Hence, we can assume that B_λ has $< k$ elements.

Furthermore, $g(A_\lambda) \leq \frac{c}{1+\delta}\lambda < c\lambda$ by assumption, and consequently A_λ has $< k$ elements.

Let $Y := \{a_1, \ldots, a_i, b_1, \ldots, b_{l-i}\}$, $l \leq k$, where $a_j \in A^t$ and $b_j \in B$. As in the previous proof, $g(\{a_1, \ldots, a_i\}) \leq g(\{a_1, \ldots, a_i\} \cup A_\lambda) \leq g(A_\lambda) + c\lambda\frac{i}{k} \leq c\lambda(\frac{1}{1+\delta} + \frac{i}{k}) \leq c\lambda(1 + \frac{i}{k}) \leq c(1 + \frac{i}{k})(1+\delta)\lambda$. Similarly, $g(\{b_1, \ldots, b_{l-i}\}) \leq g(B_\lambda) + c\frac{l-i}{k}(1+\delta)\lambda$.

By merging the two inequalities, we obtain $g(Y) = g(\{a_1, \ldots, a_i, b_1, \ldots, b_{l-i}\}) \leq g(\{a_1, \ldots, a_i\}) + g(\{b_1, \ldots, b_{l-i}\}) \leq c(1 + \frac{i}{k})(1+\delta)\lambda + g(B_\lambda) + c\frac{l-i}{k}(1+\delta)\lambda = g(B_\lambda) + 2c(1+\delta)\lambda$.

We conclude that $g(B_\lambda) \geq [1 - 2c(1+\delta)]\lambda$. □

Corollary 5.7. *There exists a sliding window $(\frac{1}{3} - \delta)$-approximation algorithm for submodular maximization that uses space $O(k \log_{1+\delta} \frac{M}{m})$.*

Proof. By Lemmas 5.5 and 5.6, there is a $(\lambda, \min(c, 1-c(1+\delta)), \delta)$-good, $(\min(c, 1-2c(1+\delta)), \delta, \frac{c}{1+\delta})$-mergeable family of summaries. Plugging this into Theorem 5.4, we obtain a $\min(\frac{c}{1+\delta}, 1-2c(1+\delta))$-approximation in the sliding window model. By choosing $c = \frac{1}{3}$, we conclude the proof. □

6 CONCLUSION

In this work, we have presented a new approach for solving maximization problems in sliding window streams, in essence showing that augmenting standard insertion-only algorithms with an additional property of suffix composability is enough to obtain state-of-the-art approaches. We provide examples of our approach for constrained subadditive and submodular optimization problems as well as for diverse set selection. In many of the instances, we provide state-of-the art results that beat previously developed, hand-tuned algorithms. An obvious open problem is how to apply this approach to more problems (including minimization problems) and to tighten some of the approximation constants we derived. A more ambitious program should take this work as a building block towards developing a better understanding of the structure of sliding window problems, with the goal of designing new techniques to give provable upper and *lower* bounds for this important class of problems.

REFERENCES

[1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Graph Sketches: Sparsification, Spanners, and Subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '12)*. ACM, New York, NY, USA, 5–14. https://doi.org/10.1145/2213556.2213560

[2] Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 20–29.

[3] Arvind Arasu and Gurmeet Singh Manku. 2004. Approximate counts and quantiles over sliding windows. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 286–296.

[4] Brian Babcock, Mayur Datar, and Rajeev Motwani. 2002. Sampling from a moving window over streaming data. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 633–634.

[5] Brain Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. 2003. Maintaining Variance and K-medians over Data Stream Windows. In *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '03)*. ACM, New York, NY, USA, 234–243. https://doi.org/10.1145/773153.773176

[6] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 2014. Streaming submodular maximization: Massive data summarization on the fly. In *ACM SIGKDD*. ACM, 671–680.

[7] Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. 2002. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*. Springer, 1–10.

[8] Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. 2016. Efficient summing over sliding windows. *arXiv preprint arXiv:1604.02450* (2016).

[9] Paul Beame, Raphael Clifford, and Widad Machmouchi. 2013. Element Distinctness, Frequency Moments, and Sliding Windows. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS '13)*. IEEE Computer Society, Washington, DC, USA, 290–299. https://doi.org/10.1109/FOCS.2013.39

[10] Vladimir Braverman, Petros Drineas, Jalaj Upadhyay, and Samson Zhou. 2018. Numerical Linear Algebra in the Sliding Window Model. *arXiv preprint arXiv:1805.03765* (2018).

[11] Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. 2014. How to catch l2-heavy-hitters on sliding windows. *Theoretical Computer Science* 554 (2014), 82–94.

[12] Vladimir Braverman, Elena Grigorescu, Harry Lang, David P Woodruff, and Samson Zhou. 2018. Nearly Optimal Distinct Elements and Heavy Hitters on Sliding Windows. *arXiv preprint arXiv:1805.00212* (2018).

[13] Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. 2016. Clustering Problems on Sliding Windows. In *SODA*. 1374–1390. https://doi.org/10.1137/1.9781611974331.ch95

[14] Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. 2016. Clustering problems on sliding windows. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 1374–1390.

[15] Vladimir Braverman and Rafail Ostrovsky. 2007. Smooth Histograms for Sliding Windows. In *FOCS*. 283–293. https://doi.org/10.1109/FOCS.2007.55

[16] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. 2009. Optimal sampling from sliding windows. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 147–156.

[17] Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. 2017. MapReduce and streaming algorithms for diversity maximization in metric spaces of bounded doubling dimension. *Proceedings of the VLDB Endowment* 10, 5 (2017), 469–480.

[18] Ho-Leung Chan, Tak Wah Lam, Lap-Kei Lee, and Hing-Fung Ting. 2012. Continuous Monitoring of Distributed Data Streams over a Time-Based Sliding Window. *Algorithmica* 62, 3-4 (2012), 1088–1111. https://doi.org/10.1007/s00453-011-9506-5

[19] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. 2015. Streaming algorithms for submodular function maximization. In *International Colloquium on Automata, Languages, and Programming*. Springer, 318–330.

[20] Jiecao Chen, Huy L. Nguyen, and Qin Zhang. 2016. Submodular Maximization over Sliding Windows. *CoRR* abs/1611.00129 (2016). arXiv:1611.00129 http://arxiv.org/abs/1611.00129

[21] Vincent Cohen-Addad, Chris Schwiegelshohn, and Christian Sohler. 2016. Diameter and k-Center in Sliding Windows. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016) (Leibniz International Proceedings in Informatics (LIPIcs))*, Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi (Eds.), Vol. 55. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 19:1–19:12. https://doi.org/10.4230/LIPIcs.ICALP.2016.19

[22] Michael S Crouch, Andrew McGregor, and Daniel Stubbs. 2013. Dynamic graphs in the sliding-window model. In *European Symposium on Algorithms*. Springer, 337–348.

[23] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining stream statistics over sliding windows. *SIAM journal on computing* 31, 6 (2002), 1794–1813.

[24] Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. 2017. Submodular Optimization Over Sliding Windows. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 421–430. https://doi.org/10.1145/3038912.3052699

[25] Philippe Flajolet and G Nigel Martin. 1985. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences* 31, 2 (1985), 182–209.

[26] Phillip B Gibbons and Srikanta Tirthapura. 2002. Distributed streams algorithms for sliding windows. In *SPAA*. ACM, 63–72.

[27] Nuno Homem and Joao Paulo Carvalho. 2011. Finding top-k elements in a time-sliding window. *Evolving Systems* 2, 1 (2011), 51–70.

[28] Regant YS Hung and Hing-Fung Ting. 2008. Finding heavy hitters over the sliding window of a weighted data stream. *Lecture Notes in Computer Science* 4957 (2008), 699–710.

[29] Piotr Indyk. 2007. Sketching, streaming and sublinear-space algorithms. *Graduate course notes, available at* (2007).

[30] Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. 2014. Composable Core-sets for Diversity and Coverage Maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '14)*. ACM, New York, NY, USA, 100–108. https://doi.org/10.1145/2594538.2594560

[31] Lap-Kei Lee and HF Ting. 2006. Maintaining significant stream statistics over sliding windows. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. Society for Industrial and Applied Mathematics, 724–732.

[32] Lap-Kei Lee and HF Ting. 2006. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 290–297.

[33] Andrew McGregor. 2005. Finding graph matchings in data streams. In *APPROX-RANDOM*, Vol. 3624. Springer, 170–181.

[34] Shanmugavelayutham Muthukrishnan et al. 2005. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science* 1, 2 (2005), 117–236.

[35] Michael Saks and Xiaodong Sun. 2002. Space Lower Bounds for Distance Approximation in the Data Stream Model. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing (STOC '02)*. ACM, New York, NY, USA, 360–369. https://doi.org/10.1145/509907.509963

[36] Hing-Fung Ting, Lap-Kei Lee, Ho-Leung Chan, and Tak Wah Lam. 2011. Approximating Frequent Items in Asynchronous Data Stream over a Sliding Window. *Algorithms* 4, 3 (2011), 200–222. https://doi.org/10.3390/a4030200

[37] Yanhao Wang, Qi Fan, Yuchen Li, and Kian-Lee Tan. 2017. Real-time influence maximization on dynamic social streams. *Proceedings of the VLDB Endowment* 10, 7 (2017), 805–816.

[38] Linfeng Zhang and Yong Guan. 2008. Frequency estimation over sliding windows. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, 1385–1387.