The Expiration Streaming Model: Diameter, k-Center, Counting, Sampling, and Friends

Lotte Blank
University of Bonn, Germany

Sergio Cabello
University of Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

Mohammad Taghi Hajiaghayi
University of Maryland, College Park, MD, USA

Robert Krauthgamer
The Harry Weinrebe Professorial Chair of Computer Science,
Weizmann Institute of Science, Rehovot, Israel

Sepideh Mahabadi
Microsoft Research, Redmond, WA, USA

André Nusser
Université Côte d'Azur, CNRS, Inria, Sophia Antipolis, France

Jeff M. Phillips
University of Utah, Salt Lake City, UT, USA

Jonas Sauer
Karlsruhe Institute of Technology, Germany

Abstract

An important thread in the study of data-stream algorithms focuses on settings where stream items are active only for a limited time. We introduce a new *expiration model*, where each item arrives with its own arbitrary expiration time. The special case where items expire in the order that they arrive, which we call consistent expirations, contains the classical sliding-window model of Datar, Gionis, Indyk, and Motwani [SICOMP 2002] and its timestamp-based variant of Braverman and Ostrovsky [FOCS 2007].

Our first set of results explores the expiration streaming model and presents algorithms for several fundamental problems, including approximate counting, uniform sampling, and weighted sampling by efficiently tracking active items without explicitly storing them all. Naturally, these algorithms have many immediate applications, e.g., to range counting.

Our second and main set of results for the expiration model designs algorithms for the diameter and k-center problems, where items are points in a metric space. Our results significantly extend those known for the special case of sliding-window streams by Cohen-Addad, Schwiegelshohn, and Sohler [ICALP 2016], and obtain a strictly better approximation factor for the diameter in the important special case of high-dimensional Euclidean metrics. We develop new decomposition and coordination techniques along with a geometric dominance framework to filter out redundant points based on both temporal and spatial proximity.

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

Keywords and phrases clustering, diameter, streaming, sliding window, sampling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.37

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2509.07587

53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026).
Editors: Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis;
Article No. 37; pp. 37:1–37:24

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This research was initiated during the Workshop "Massive Data Models and Computational Geometry" held at the University of Bonn in September 2024 and funded by the DFG, German Research Foundation, through EXC 2047 Hausdorff Center for Mathematics and FOR 5361: KI-FOR Algorithmic Data Analytics for Geodesy (AlgoForGe).
Lotte Blank: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 459420781 (FOR AlgoForGe).
Sergio Cabello: Funded in part by the Slovenian Research and Innovation Agency (P1-0297, N1-0218, N1-0285, J1-70045). Funded in part by the European Union (ERC, KARST, project number 101071836). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
Mohammad Taghi Hajiaghayi: The work is partially supported by DARPA expMath, ONR MURI 2024 award on Algorithms, Learning, and Game Theory, Army-Research Laboratory (ARL) grant W911NF2410052, NSF AF:Small grants 2218678, 2114269, 2347322.
Robert Krauthgamer: Work partially supported by the Israel Science Foundation grant #1336/23, by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center, and by a research grant from the Estate of Harry Schutzman.
André Nusser: Supported by the French government through the France 2030 investment plan managed by the National Research Agency (ANR), as part of the Initiative of Excellence of Université Côte d'Azur under reference number ANR-15-IDEX-01.
Jeff M. Phillips: Work supported by funding from NSF 2115677 and 2421782, and Simons Foundation MPS-AI-00010515.
Jonas Sauer: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 459420781 (FOR AlgoForGe).

Acknowledgements We thank the organizers and participants of the Bonn workshop for the stimulating environment that inspired this research. We also thank the anonymous reviewers for useful references and comments that improve the exposition.

1 Introduction

The sliding-window streaming model is widely used to represent a time-sensitive stream, i.e., a sequence of data items that arrive over time and are active only for a limited time. In the classical formulation of this model [5, 19], the w most recent items, for a parameter w, form an *active window*, and all queries are applied to this active window. Hence, non-active items are also called *expired*. It is convenient to think of the items as arriving at successive time steps $t = 1, 2, \ldots$, and thus every item is active for exactly w time steps. Another variant of this model allows blank time steps where no item arrives, which is essentially like having discrete items in a continuous time horizon, and in this case the active window's size (number of items) might vary over time. Most theoretical research has focused on the first variant above, called sequence-based windows, but, as explained, most results easily extend to the second variant, called timestamp-based windows. For instance, see [10], which mentions both but explicitly analyzes only the first variant.

We introduce a significantly richer *expiration model*, where each item arrives with its own expiration time. We stress that the order in which items expire (i.e., become non-active) is arbitrary. This generality befits numerous scenarios where items are heterogeneous in terms of data type, reliability, origin, policy settings, and so forth. For example, think of credentials in a computer network or graphics objects on a screen. A notable special case is the *consistent expiration model*, where items expire in the order in which they arrive. We sometimes emphasize that we consider the general case, where expirations need not be consistent, by referring to it as the *general expiration model*. The consistent expiration model contains the classical sliding-window model discussed above, where all items are active for

the same duration w. Many sliding-window algorithms barely depend on w (e.g., their space bound has $\log w$ factors). In fact, some merely use the property that the items expire in the order in which they are inserted but not that exactly w items are active at any point in time, and therefore these algorithms carry over immediately to the consistent expiration model. However, extending them to the general expiration model seems considerably more challenging, as new ideas seem necessary to handle items with non-consistent expirations. It is an intriguing question which problems are indeed harder in this new model and how, i.e., whether they truly require more storage or merely a more sophisticated algorithm.

Another related model is *turnstile streaming*, where the input stream consists of item insertions and deletions. The special case where an item can only be deleted after it was inserted is called *strict turnstile* or, especially in geometric and graph settings [28, 4], *dynamic streams*. Although this model bears similarity to our expiration model, it is actually incomparable. The crucial difference is that in turnstile streaming, each deletion triggers the algorithm explicitly at the time of deletion, whereas in our expiration model the deletions occur implicitly, because the expiration information is provided only when the item arrives. Performing the deletions explicitly would require the algorithm to store the expirations of all active items, which is excessive. For an in-depth discussion of related models and further related work, we refer to Section 1.2.

Let us now formally define our expiration model. The input stream is a sequence of items $\langle x_1, \ldots, x_n \rangle$, where each item x consists of an actual data point $p(x)$ from some universe U (e.g., a metric space), an insertion time $S(x) > 0$, and an expiration time $E(x) > S(x)$, where the insertion times must be non-decreasing, i.e., $S(x_1) \leq \ldots \leq S(x_n)$. For convenience, we assume that the insertion and expiration times are integers, and we allow items that never expire by having $E(x) = \infty$. We sometimes identify each item x with its data point $p = p(x)$, which slightly abuses notation because data points do not need to be distinct, and denote its insertion and expiration times by $S(p)$ and $E(p)$, respectively. An item x is *active* at time t if $S(x) \leq t < E(x)$, i.e., from its insertion time up to (but not including) its expiration time. A query at time t is evaluated only on the set of items active at that time t. For simplicity, we design our algorithms to handle a single query that arrives at an arbitrary time $t > 0$ not known in advance to the algorithm. These algorithms often extend to handle multiple queries. For deterministic algorithms this is immediate., and for randomized algorithms there are standard arguments, such as probability amplification via independent repetitions.

When items never expire, i.e., $E(x) = \infty$ for all x, this is precisely the classical model of insertion-only streams. We say that the stream has *consistent expirations* if the expiration times are non-decreasing, i.e., $E(x_1) \leq \ldots \leq E(x_n)$. When $E(x) - S(x) = w$ for all x, particularly if insertion times are successive, i.e., every $S(x_i) = i$, then this is precisely the classical sliding-window model. In fact, it is convenient to focus on the special case where every $S(x_i) = i$, which, as mentioned earlier, holds without loss of generality if we allow blank time steps where no item arrives.

Our space-complexity bounds count machine words (unless mentioned otherwise), where a word can store a data point (e.g., from a metric space), a time instantiation (e.g., some $S(x)$), or a counter in the range $[\text{poly}(n)]$, where throughout we define $[k] = \{1, \ldots, k\}$. This convention avoids bit-representation issues, although in a simplified case where every $S(x_i) = i$ and data points lie in a universe $U = [\text{poly}(n)]$, every word has $\Theta(\log n)$ bits.

1.1 Results

Fundamental Problems. The first problems we consider in the general expiration model are some fundamental streaming problems that are standard building blocks for solving many other problems, thereby gaining a better understanding of the challenges when designing

algorithms in our new model. We start with *counting*, which formally asks for the number of active 1's in a stream of items from the universe $U = \{0, 1\}$. For exact counting, $\Omega(n)$ bits of space are required, even in the consistent expiration model; which follows from the known bound for the sliding-window model [19], in contrast with insertion-only streams, where $\mathcal{O}(\log n)$ bits clearly suffice.[1] Due to these lower bounds, we turn to *approximate* counting. We design two randomized streaming algorithms, one achieves ε-additive error using $\mathcal{O}(\varepsilon^{-1})$ space, and the other ε-relative error using $\tilde{\mathcal{O}}(\varepsilon^{-1} \log(\varepsilon n))$ space,[2] by employing powerful tools of insertion-only streams, like quantile sketches. Moreover, we show that these space bounds are tight. Approximate counting is a useful primitive when designing other algorithms. We demonstrate this by designing a Count-Min sketch [18], which solves frequency estimation (aka point queries) with ε-additive error, and thus also ℓ_1-heavy-hitters, using $\mathcal{O}(\varepsilon^{-2})$ space.

Another fundamental problem is to *sample k items* uniformly from the stream. In insertion-only and sliding-window streams, this can be done via reservoir sampling [34, 5]. We build on this technique to present an algorithm for the expiration model that uses $\mathcal{O}(k \log n)$ space, and further present extensions to two more challenging tasks: sampling *without replacement*, and *weighted sampling*, where each item is sampled with probability proportional to its weight. Sampling tasks are useful primitives, and we indeed use them to design other algorithms, e.g., for approximate quantiles with ε-additive error using $\mathcal{O}(\varepsilon^{-2} \log n)$ space. We also use them for geometric problems, such as range counting, logistic regression, and kernel density estimation (KDE); here, items are points in $\mathbb{R}^d$, and our algorithms use space $\mathcal{O}(\varepsilon^{-2} \log n)$, assuming for simplicity that parameters such as d and the VC-dimension are $\mathcal{O}(1)$. Moreover, these sampling methods apply to matrix-approximation problems, where each stream item is a row vector in $\mathbb{R}^d$, including rank-k approximation and ε-covariance error.

Technically, these results are less involved and build heavily on prior work. They also leave several questions for further investigation. For example, our algorithms often use more space than the analogous ones for insertion-only and turnstile streams, and thus may possibly be improved, e.g., better dependence on ε. Note that for frequency estimation in sliding-window streams, near-optimal bounds are known [8]. Furthermore, our techniques do not yield Count-Sketch-type bounds [15] for ℓ_2-point queries and ℓ_2-heavy-hitters, which are known for sliding-window streams [8, 24]. We are also not aware of any strict separation (in space complexity) between consistent and general expirations.

Clustering Problems. Our main results (which are technically more challenging) are for the diameter and k-center problems in a general metric space $\mathcal{M}$, where $d_{\mathcal{M}}$ denotes its distance function and the subscript may be omitted when clear from the context. In the streaming setting, the metric $\mathcal{M}$ is fixed in advance and each item x contains a data point $p(x) \in \mathcal{M}$, and recall that we may identify $p(x)$ with x and write $x \in \mathcal{M}$. To avoid precision issues, we assume that $d_{\mathcal{M}}(x, y) \in [1, \Delta]$ for all distinct $x, y \in \mathcal{M}$.[3]

[1] An alternative definition, which asks to count the total number of active items, exhibits strict separation between the sliding-window and expiration models. In the sliding-window model (without blank time steps), the answer is always w and thus $\mathcal{O}(1)$ space suffices, whereas with expirations, even consistent ones, the lower bound of $\Omega(n)$ bits still holds.

[2] Throughout, the notation $\tilde{\mathcal{O}}(f)$ hides logarithmic factors in f.

[3] Our results hold even if we allow $d_{\mathcal{M}}(x, y) = 0$, i.e., $\mathcal{M}$ is a pseudometric. In fact, the same point may arrive multiple times, possibly with different expirations.

In the *diameter problem*, the goal is to compute $\operatorname{diam}(X) := \max\{d(x,y) \mid x,y \in X\}$ for the set $X \subset \mathcal{M}$ of active items. We devise an algorithm for this problem in the expiration model; it significantly extends the previously known algorithm, that works in the more restricted sliding-window model [17], while achieving the same approximation factor and space complexity. We prove the following in Section 3.

▶ **Theorem 1.1.** *There exists a deterministic expiration-streaming algorithm maintaining a $(3+\varepsilon)$-approximation of the diameter in a general metric $\mathcal{M}$ storing $\mathcal{O}((1/\varepsilon)\log\Delta)$ words.*

Our approximation factor is almost tight, as even in the sliding-window model, achieving a $(3-\varepsilon)$-approximation for the diameter for any fixed $\varepsilon > 0$ requires $\Omega(\sqrt[3]{n})$ space [17].[4]

We further improve the approximation factor to $1+\sqrt{3}+\epsilon \approx 2.73$ when the metric space $\mathcal{M}$ is Euclidean, regardless of the dimension (we only require that each point can be stored in a machine word). Previously, an approximation factor below 3 was not known for the Euclidean case, even in sliding-window streams.

▶ **Theorem 1.2.** *There exists a deterministic expiration-streaming algorithm maintaining a $(1+\sqrt{3}+\varepsilon)$-approximation of the diameter in Euclidean space storing $\mathcal{O}((1/\varepsilon)\log\Delta)$ words.*

Finally, we turn to the *k-center problem*, where the goal is to compute

$$OPT_k := \min_{C\subset\mathcal{M},|C|=k} \max_{x\in X} d(x,C),$$

where $d(x,C) := \min_{c\in C} d(x,c)$ is the distance from a point x to its closest point in a set C.

We present an algorithm for the expiration model that achieves a $(6k+2)(1+\varepsilon)$-approximation using $\mathcal{O}(\varepsilon^{-1}k^2\log\Delta)$ words of space. In comparison, previous work achieved $(6+\varepsilon)$-approximation using $\mathcal{O}(\varepsilon^{-1}k\log\Delta)$ words in the significantly more restricted sliding-window model [17]. We prove the following in Section 4.

▶ **Theorem 1.3.** *There exists a deterministic expiration-streaming algorithm that stores $\mathcal{O}((K^2/\varepsilon)\log\Delta)$ words and maintains a $(6k+2)(1+\varepsilon)$-approximate solution for k-center, for every $k \le K$, in a general metric $\mathcal{M}$.*

For $k=1$, we use our diameter algorithm to achieve an improved approximation factor of $4+\varepsilon$ in a general metric (Theorem 3.2) and $1+\sqrt{3}+\varepsilon$ in Euclidean space (Theorem 3.6). All our algorithms report not only an objective value but also a feasible solution, i.e., a pair of points (that is approximately the farthest) or a set of k center points. Table 1 lists the known approximation factors for diameter and 1-center in high-dimensional Euclidean space under different streaming models, including our new expiration model, and two more restricted ones of sliding-window and insertion-only streams. (We restrict the table to $k=1$ and the Euclidean case to minimize clutter.)

Many results and proofs are omitted from this version due to space constraints. They can be found in the full version.

1.2 Related Work and Models

The expiration model that we propose is not only a natural generalization of the sliding-window and other streaming models, it also pertains to numerous other topics in computer science, as we outline next. One example where general (i.e., non-consistent) expirations arise

[4] It is easy to see that $\mathcal{O}(1)$-approximation requires space complexity $\Omega(\log\Delta)$ already in sliding-window streams, e.g., consider the one-dimensional input where the i-th point is either 2^i or 0 [23].

Table 1 Approximation factors of streaming algorithms for diameter and for 1-center (also called Minimum Enclosing Ball) in high-dimensional Euclidean space $\mathbb{R}^d$. We list only algorithms whose space complexity is $\text{poly}(d \log(n\Delta))$. Some algorithms work also in general metric spaces.

model	diameter	1-center (MEB)	references & comments
insertion only:			
	2	2	folklore; general metrics
		1.5	[38]
	$\sqrt{2}+\varepsilon$	$\frac{1+\sqrt{3}}{2}+\varepsilon \approx 1.37$	[3]
		$1.22+\varepsilon$	[13]
	$\sqrt{2}+\varepsilon$	$1.22+\varepsilon$	[26]
turnstile/dynamic:			
	$\mathcal{O}(\sqrt{\log n})$		[27]
sliding-window:			
	$3+\varepsilon$	$4+\varepsilon$	[17]; general metrics
		$9.66+\varepsilon$	[35]
general expiration:			
	$1+\sqrt{3}+\varepsilon \approx 2.73$	$(1+\sqrt{3}+\varepsilon) \approx 2.73$	Theorem 1.2, Theorem 3.6
	$3+\varepsilon$	$4+\varepsilon$	Theorem 1.1, Theorem 3.2; general metrics

is online monitoring settings. Another example is computational economics and networking, where certificates or contracts are issued with known lengths/expirations, and must be managed by computer systems. Our results are applicable when one is willing to sacrifice accuracy (bounded approximation) to attain dramatic space improvements.

Semi-Online Data Structures. In the study of dynamic data structures, the semi-online model allows for insertions and deletions where the time of deletion is provided at the time of insertion. Here, unlike the sketches we study, no approximation is generally allowed, and the focus is instead on reducing the running time of various operations. In particular, Dobkin and Suri [22] employed the method of Bentely and Saxe [6] under a data structure that can be constructed in time $P(n)$ and can handle queries in time $Q(n)$, in order to handle insertions and deletions in this semi-online model in $\mathcal{O}(\log n)$ time while increasing the query time to $\mathcal{O}((P(n)/n + Q(n)) \log n)$. Several linear and near-linear space algorithmic improvements followed, where the focus is primarily on exact methods and improved update times. Notably, Chan [12] provided improvements for several problems in computational geometry, including discrete 1-center in 2 dimensions with slightly sublinear updates.

Persistent Stream Queries. The database community studied persistent sketches (for streaming), where queries may be restricted to subsets of data in certain time windows [36]. Within this setting, Shi, Zhao, Peng, Li, and Phillips [33] considered at-the-time persistence (ATTP) and back-in-time persistence (BITP) models, where the time window of a query must include the first or last time, respectively (in other words, queries about any prefix or any suffix of the streams). Notably, the BITP model can be interpreted as a sliding-window query that specifies the window size w at query time (rather than in advance). This is closely related to consistent expirations, especially if all expirations occur after all relevant insertions. All methods we are aware of for the consistent expiration model should work for this BITP model.

In this context, Shi et al. [33] studied a variety of problems related to weighted counting and sampling, where each item x_i is associated with a weight w_i (it could be uniform), and the desired error bound is an additive εW where $W = \sum_i w_i$ is the total weight. This setting is very useful in standard sketching bounds for frequency estimation, quantiles, approximate range counting, kernel density estimates, and matrix-covariance sketching, which we study as well. In particular, they show that a random sample of size k can be maintained in $\mathcal{O}(k \log n)$ expected space with $\mathcal{O}(\log k)$ expected amortized update time. If the items are selected at random proportionally to their weight and the weights are in the range $[1, U]$, then the expected space is $\mathcal{O}(k \log(nU))$. Their methods for BITP use a sampling-in-reverse analysis, which was discovered earlier by Braverman et al. [7] in the context of sliding-window linear-algebra problems, and also used later for sampling Lewis weights in sliding-window streams [37]. Moreover, they show that for mergeable sketches [1] of size $s(\varepsilon)$, a BITP sketch of size $\mathcal{O}(\varepsilon^{-1} \log n \cdot s(\varepsilon))$ can be maintained.

Our results extend these ideas to the general expiration model, and formalize the proofs in the full version of this paper. Dealing with general expirations requires finding the relevant tools (in prior literature) and additional ideas, to manage samples that expire in a completely different order than their arrivals.

Sliding-Windows and Smooth Histograms. The sliding-window model has been studied extensively, including for frequency and counting problems, maintaining aggregate statistics, and for geometric and graph problems. An extremely popular technique for designing sliding-window streaming algorithms is the smooth-histogram framework of Braverman and Ostrovsky [10]. For monotone functions f that satisfy a certain smoothness property, they show how to convert an algorithm that estimates f in insertion-only streams into an algorithm that estimates f, using slightly more space, in sliding-window streams. This framework has been successfully employed for many different problems, from counting and frequency problems to graph problems, but not for geometric problems, which are often not smooth, e.g., k-median and k-means clustering [9]. Krauthgamer and Reitblat [31] defined a relaxation of this smoothness property, called almost-smoothness, which is still sufficient to convert algorithms from insertion-only to sliding-window streams, albeit with a bigger loss in the approximation factor. It is not difficult to see that the diameter problem, in a general metric space, is 2-almost-smooth, and since it admits a folklore 2-approximation in insertion-only streams, the conversion of [31, Theorem 1.7] also implies an $(8 + \varepsilon)$-approximation in sliding-window streams. Although immediate, this bound is worse than the known $(3 + \varepsilon)$-approximation for diameter [17]. Unfortunately, this entire framework seems inapplicable to general expirations.

Streaming Algorithms for Diameter and k-Center. Clustering problems have been studied extensively in the streaming model. The metric k-center problem was studied in insertion-only streams, culminating in a $(2 + \varepsilon)$-approximation [14, 32, 25], and further extensions to k-center with outliers. In Euclidean space of high dimension d, i.e., when the space bound is restricted to be polynomial in d, better approximation factors are known, particularly for $k = 1$ and for diameter [38, 3, 13, 29, 26], and for small k [30]; see Table 1 for the precise constants. These results were extended to sliding-window streams in [17, 35] as mentioned above. However, they usually do not extend to dynamic streams, which seem harder.

In low Euclidean dimension, i.e., when allowing a space bound that grows exponentially with d, several results achieve a $(1+\epsilon)$-approximation for k-center [2, 11]. These results often extend to dynamic and sliding-window streams, and to handle outliers [21, 20]. Some of the above references prove near-matching lower bounds, but usually for algorithms that must store input points [25, 3, 20], i.e., these are not bit-complexity bounds for general algorithms.

2 Technical Overview

A key challenge in the expiration model is that the algorithm has to track the active items without explicitly storing them all. In particular, it must be prepared for a scenario where no additional items arrive, and at some future time it will be asked for an estimate.

Counting. Perhaps the simplest challenge is to just maintain a count of the active items. In insertion-only streams, it suffices to maintain a single counter. In contrast, in the expiration model, the algorithm must provide an answer at all possible future times, which by a reduction to INDEXING requires $\Omega(n)$ bits of space.

We observe that approximate counting in the expiration model is equivalent to approximating a 1-dimensional distribution on the expiration times. In particular, guaranteeing additive error εn on the count corresponds to ε-error in the Kolmogorov-Smirnov distance. If the expirations are consistent, it can be handled by simply recording a check point every εn insertions. The general-expiration case may seem much more complicated, but fortunately, it maps directly to the classic problem of quantiles summary in insertion-only streams: each item arrival in the expiration stream corresponds to inserting the expiration time of that item into the quantile summary. Thus, we can employ both quantile sketch upper bounds and their structural lower bounds. For additive error εn with a constant probability of failure to answer one query, this uses $\Theta(1/\varepsilon)$ space. For ε-relative error, it uses $\tilde{\Theta}((1/\varepsilon)\log(\varepsilon n))$ space.

Sampling. Maintaining a random sample, poses a similar challenge in that a sample is in some sense a subset approximation of the count of the items. However, the maintained sample might expire, and other items must be stored in advance to replace the expired one. In the general-expiration case, the set of active items changes dynamically, and not in a controlled manner as in consistent expirations.

The main insight is that we can imagine running a reservoir sampler in the reverse order of the expirations – assuming we know these expiration times all in advance. For maintaining a single sample, we just keep track of when the reservoir gets updated; our sketch only needs to maintain items that ever get placed in the reservoir. The size of this set can be analyzed as a coupon collector problem, and is $\mathcal{O}(\log n)$ in expectation for a single sample.

Now we must maintain this sample without knowing the expiration times in advance. The trick is to assign each item a random value $u_i \in \mathsf{Unif}[0,1]$ and select the smallest active value as our sample. Because this randomness is assigned once and does not change later, we only need to maintain the smallest u_i value among the active items. Thus, an item with a larger u_i value than another item that expires later does not need to be maintained, and the remaining items are maintained, say in sorted order by expiration. The above description produces a single sample, and we can just run k independent copies to get a sample of size k.

We can then adapt this analysis to sampling proportional to weights, as long as the total weight W of the stream is bounded and each weight is at least 1. By changing to an exponential distribution, namely each $u_i \sim \mathsf{Exp}(w_i)$, we get that the item with minimal value is chosen proportional to its weight. This follows from the min-stability property of the exponential distribution, as previously exploited by Cohen [16]. This has the same complexity as an expiration stream of W items with uniform weights.

Implications of Counting and Sampling. Many statistically motivated summaries essentially require only access to a counter of the data items or to a random sample from it. Therefore, being able to maintain a counter or a sample over a stream has numerous applications, and indeed fairly direct implications follow for problems involving quantiles, range counting, classification, regression, kernel density, and even matrix sketching.

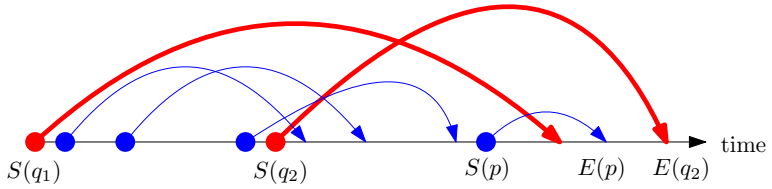

Figure 1 The long items of the stream are q_1 and q_2. Item p is not long as it is dominated by q_2.

2.1 Diameter

The next technical challenge is maintaining the diameter of a point set. Unlike the counting and sampling problems, here items differ not only in their expiration time, but also in their geometric information, and the algorithms must preserve the *geometry* of *all* active items at all times.

Simple $\mathcal{O}(1)$-Approximation in General Metrics. Let p and q be items of the stream. We say that p is *dominated* by q if $S(q) < S(p)$ and $E(p) \leq E(q)$, i.e., for the entire time that p is active, q is active as well. An item that is not dominated by any other item is called *long*. See Figure 1 for illustration. Note that it is easy to decide with $\mathcal{O}(1)$ words of space whether an arriving item is dominated or long, by keeping track of the latest expiration time among all the items that have arrived so far.

Consider first the substream consisting of all long items, and notice that its expirations are consistent. It is easy to show that the sliding-window algorithm of [17] generalizes to the consistent expiration case, and so we can use it as a subroutine for the long items. For an item q that is not long, suppose that it is geometrically close to some long item p that dominates it. Then we can simply ignore q and use p as a proxy for it: geometrically, they are close, and timewise, p is active whenever q is active. In the other case, i.e., if q is far from p, then the pair p, q may be a candidate for the diameter as long as q is active, and thus we need to keep track of such pairs.

More precisely, for a dominated item q, let us assign one long item that dominates q as its *parent*. Again, this is easy to implement, as soon as q arrives, using $\mathcal{O}(1)$ words of space. Now there can be two cases:

(a) Every dominated point is close to its parent (say up to a small constant fraction of the diameter). In this case, the diameter of the long items is a constant-factor approximation to the diameter of all items (by simple triangle-inequality arguments).

(b) At least one dominated point is far from its parent. To keep track of this scenario, we store an array A, where each entry $A[j]$ maintains a pair of points, namely, a dominated point p and its parent q, selected as follows: from all such pairs where also $d(p, q) \in [(1+\varepsilon)^j, (1+\varepsilon)^{j+1})$, select the pair for which the expiration time of p is the latest. The size of this array is only $\mathcal{O}((1/\varepsilon) \log \Delta)$.

The above reasoning suffices to obtain a constant-factor approximation of the diameter in the general expiration model, however it does not match the approximation factor $3+\varepsilon$ that is known for the sliding-window setting.

An Improved Algorithm. To improve the constant, we replace the black-box subroutine for consistent expirations that is used to handle the long items. Our redesigned subroutine is conceptually different from the sliding-window algorithm of [17] and allows for a tighter integration with the array A, which seems not possible using the algorithm of [17]. If space

complexity is not a concern, then long items can be handled by simply having every long point p track the maximum distance to any point inserted after it and not dominated by it, which we call the *radius* of p. From this information, we can retrieve the exact diameter of the long items. The main insight to achieve the claimed space bound is that if a long point p is "sandwiched" between two points with a similar radius, one inserted before p and the other after, then we can discard p while only losing a constant factor in the approximation guarantee.

In addition, we again store an array A that captures the distances of dominated items to long items. In contrast to the above, we do not assign a particular parent to each dominated point p. Rather, when p arrives, we consider its distance to all stored long items that dominate p. A more involved analysis shows that this improves the approximation factor to $3 + \varepsilon$, matching the best approximation factor known also for the sliding-window model, while using the same space bound.

Further Euclidean Improvement. An additional advantage of our new subroutine for the consistent expiration case is that it admits further improvements in the Euclidean space. In general metric space, the algorithm stores a radius r for every stored long item p to bound the distance between p and the set S of points that are inserted after p and are not dominated by p. In Euclidean space, the algorithm additionally stores for every p a carefully chosen point $\widetilde{p}$, whose distance to points in S is at most r as well. Hence, S lies in the intersection of two balls of radius r, instead of only one ball of radius r. Since the distance between p and $\widetilde{p}$ is bounded, we can further improve the approximation factor for both consistent and general expirations. In fact, this is strictly better than the best approximation known (and possible) for general metrics, even in the sliding-window model. See Section 3 for details.

2.2 Approximating k-Center in Expiration Streams

We focus on the decision version of k-center, where the goal is to decide whether the value of an optimal solution is roughly γ. We can then instantiate the decision algorithm for $\mathcal{O}((\log \Delta)/\varepsilon)$ different guesses of γ to obtain our approximation.

As with the diameter, one can verify that the sliding-window algorithm of [17] in fact works for the consistent expiration model, and thus it provides an $\mathcal{O}(1)$-approximation of k-center on the set of all long items. Once again, if a point q that is dominated by p is also geometrically *close* to p, then there is no need to keep the point q, since any center that is close to p is also not far away from q. Thus, our strategy is to divide the stream into k substreams $D_1, \ldots, D_k$ such that for each substream D_i, the expirations are consistent and therefore a modified instance of the algorithm of [17] can be run on it. The challenge lies in ensuring that k substreams are sufficient. To illustrate the argument, we first consider a simplified scenario in which each substream is allowed to use an unlimited amount of space. Then, we explain how the algorithm can be modified to achieve the desired space bound.

A Simplified Scenario. We define D_1 as the substream containing all long points p, as well as each point q that is geometrically close to a long point p in D_1 that dominates it. If a point q is not geometrically close to any long point in D_1 that dominates it, then q is fed to the second substream D_2. All points passed on to D_2 are then processed in the same manner as for D_1, and the residual points not handled by D_2 are again passed to the next layer, etc. If a point is rejected from the final substream D_k, it is discarded. If each substream is allowed to use unlimited space, then there is no need to discard any long items and we can show that k substreams are sufficient. If a point q is rejected from all k substreams $D_1, \ldots, D_k$, then there is a sequence of points $p_1, \ldots, p_k$ such that

(i) p_i belongs to the ith substream,
(ii) these k points (as well as q) are pairwise geometrically far away from each other,
(iii) the points are nested timewise, i.e., p_{i+1} is dominated by p_i and q is dominated by p_k.

As long as q is not expired, then all points $p_1, \ldots, p_k$ are also active, which provides a certificate that any solution for k-center must have a large value with respect to γ. Thus, there is no need to keep q, and more generally, to maintain more than k substreams.

To bound the space within each substream, a naive approach would be to apply the algorithm of [17] to each substream independently. However, because the pruning of long items is not coordinated between the substreams, it becomes impossible to maintain conditions (ii) and (iii) simultaneously. Consider three points p_i, p_j, p_ℓ with $i < j < \ell$ such that p_i is discarded between the insertions of p_j and p_ℓ. Then the algorithm cannot check whether p_ℓ is geometrically close to p_i, so condition (ii) may be violated. If p_i is replaced with another point p'_i from D_i, then it cannot be guaranteed that p'_i dominates p_j, so condition (iii) may be violated. Thus, a more coordinated approach is needed.

Our Approach. Our goal is to maintain a set of points $p_1, \ldots, p_k$ that satisfy conditions (i)–(iii), while bounding the space usage of the algorithm within each substream and the total number of substreams. If we define p_i as the longest-living point in D_i, then this set of points satisfies conditions (i) and (iii). To restore condition (ii), we discard points from D_i if they are geometrically close to points from substreams D_j with $j < i$ that are inserted later. This provides the coordination between the different substreams that was lacking in the naive approach, but it creates two new issues.

The first issue arises when our algorithm does not store a point p in substream D_i because it is close to another point p'_i in D_i (in which case we consider p to be *covered* by p'_i in D_i). Later on, we may have to discard p'_i to ensure condition (ii) because it is close to another point p'_j from a substream D_j with $j < i$. We then consider p to be covered by p'_j instead, but this increases the distance between p and the point that covers it by a constant factor. Because p'_j may itself be discarded later on to ensure condition (ii), this effect can cascade up to k times, which increases the approximation guarantee to $\mathcal{O}(k)$.

The second issue is that the longest-living point from a substream D_i may need to be discarded to restore condition (ii), and this can in turn cause condition (iii) to be violated because the points $p_1, \ldots, p_k$ are no longer nested. We restore the condition by moving the longest-living point among all substreams D_j with $j > i$ to D_i. Note that this may potentially violate condition (ii) again. However, we show that a single sweep over all substreams suffices to restore both conditions. For a more detailed overview of the algorithm, see Section 4.

3 Approximating the Diameter in Expiration Streams

This section presents algorithms that approximate the diameter of a point set in the expiration streaming model. We first present an algorithm that matches the approximation factor known for sliding-window streams. Afterwards, we modify this algorithm for Euclidean space to achieve a better approximation factor. For simplicity, we assume in this section that the insertion times of the items are distinct, i.e., $S(p) \neq S(p')$ for all $p \neq p'$. If multiple items arrive simultaneously, we fix an arbitrary order to add them to the stream.

3.1 $(3+\varepsilon)$-Approximation in General Metrics

Our algorithm maintains a subset $q_1, \ldots, q_k$ of the long items that have been inserted so far. Along with every stored item q_i, we store the radius r_i of the smallest ball centered at q_i that contains all elements that were inserted after q_i and are not dominated by q_i,

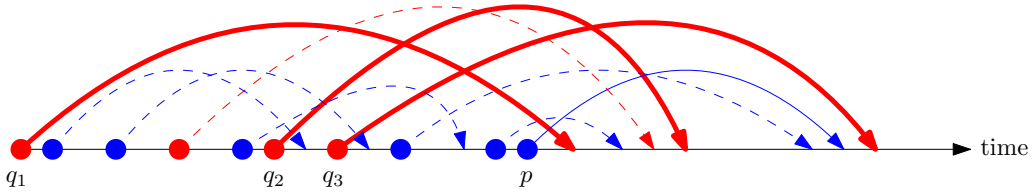

Figure 2 Long items drawn in red. Item p is dominated by q_3. At time $S(p)$, when p is inserted, items q_1, q_2, q_3 are all stored. The algorithm then updates the radii r_1 and r_2 corresponding to q_1 and q_2, respectively, and also the entry in A corresponding to the distance $d(p, q_3)$.

i.e., expire later than q_i (see Figure 2). In addition, the algorithm stores an array A. For each $j \in [\lceil \log_{1+\varepsilon/3} \Delta \rceil]$, the entry $A[j]$ stores the maximum expiration time $E(p)$ among all pairs of points q and p such that (1) q is long and stored, (2) p is dominated by q, and (3) $d(q,p) \in [(1+\varepsilon/3)^j, (1+\varepsilon/3)^{j+1})$. Before the stream starts, we insert a dummy item p that expires immediately after it is inserted. Then, initially we store $q_1 = p$, $r_1 = 0$, $k = 1$, and $A[j] = 0$ for all j.

Update Procedure. Each time an item (p, S, E) is inserted, we use Algorithm 1 to update the data structure. If the item is long, we store it. Line 5 ensures that for each stored (long) item q_i, the radius r_i is correct: if q_i does not dominate the new item p and the distance from q_i to p is larger than r_i, we update the radius to be this distance. Otherwise, if q_i dominates p, then for the value of j such that their distance is in $[(1+\varepsilon/3)^j, (1+\varepsilon/3)^{j+1})$, the algorithm updates $A[j]$ to $\max\{A[j], E(p)\}$ (lines 7–8).

To achieve our bound on the number k of the stored items, we further discard all stored items that are inserted between two stored items with a similar radius. In particular, if q_i and q_j with $i \le j$ are two stored points such that $(1+\varepsilon/3) \cdot r_j > r_i$, then we discard all items q_t with $t \in \{i+1, \ldots, j-1\}$. Finally, at the end of the update procedure of adding an item, we reindex the stored items starting from 1, keeping their order.

Handling Expirations. As long as at least two items are stored, the algorithm maintains the invariant that $E(q_1) \le T < E(q_2)$ for the current time step T. In the time step $T = E(q_2)$, the algorithm discards q_1 from the data structure along with its radius r_1. The remaining items are reindexed starting from 1, keeping their order. As all stored items are long, at each time step at most one stored item expires.

Answering Queries. At time T, we report $r = \max\{r_2\} \cup \{(1+\varepsilon/3)^j \mid A[j] > T\}$ as an approximation to the diameter if q_2 exists. Otherwise, there are no active points in the stream at time T. We show that this is a $(3+\varepsilon)$-approximation.

▶ **Lemma 3.1.** *For every consecutively stored items q_{i-1}, q_i, and for every item p satisfying $S(q_{i-1}) < S(p) < S(q_i)$ and $E(q_{i-1}) < E(p)$, it holds that $d(q_{i-1}, p) \le (1+\varepsilon/3) \cdot r_i$.*

Proof. Because q_{i-1} and q_i are consecutive, any long item that was inserted between them has been discarded. If no such long item exists, then the claim follows because there is no item p with $S(q_{i-1}) < S(p) < S(q_i)$ and $E(q_{i-1}) < E(p)$. Otherwise, let T be the time at which the last long item between q_{i-1} and q_i is discarded. Denote with r_{i-1}^T (resp. r_i^T) the radius stored at time T for the point which is now q_{i-1} (resp. q_i). Then, $r_{i-1}^T < (1+\varepsilon/3) \cdot r_i^T$ holds due to line 11. In particular, for every item p with $S(q_{i-1}) < S(p) < S(q_i)$ and $E(q_{i-1}) < E(p)$, it holds that $d(q_{i-1}, p) \le r_{i-1}^T \le (1+\varepsilon/3) \cdot r_i^T$. The radius corresponding to point q_i can only increase afterwards, so the claim follows. ◀

Algorithm 1 Diameter update procedure for inserted item (p, S, E).

```
1  if E > E(q_k) then
2      q_{k+1} ← p, r_{k+1} ← 0                                  // (p, S, E) is long
3  for i ∈ [k] do
4      if E > E(q_i) then
5          r_i ← max{r_i, d(q_i, p)}
6      else
7          let j be such that d(p, q_i) ∈ [(1 + ε/3)^j, (1 + ε/3)^{j+1})
8          A[j] ← max{A[j], E}
9  i ← 1
10 while i ≤ k − 1 do
11     find the largest j ≥ i such that (1 + ε/3) · r_j > r_i
12     discard q_t and r_t for all t ∈ {i + 1, . . . , j − 1}
13     i ← max{j, i + 1}
14 k ← number of remaining elements
15 reindex the values q_i and r_i (keeping their order) to q_1, . . . , q_k and r_1, . . . , r_k
```

To show that the algorithm achieves $(3+\varepsilon)$-approximation, we analyze different cases based on the insertion and expiration times of the items that realize the diameter.

▶ **Theorem 1.1.** *There exists a deterministic expiration-streaming algorithm maintaining a $(3+\varepsilon)$-approximation of the diameter in a general metric $\mathcal{M}$ storing $\mathcal{O}((1/\varepsilon)\log\Delta)$ words.*

Proof. By lines 11–13 of Algorithm 1, it holds that $(1+\varepsilon/3)\cdot r_{i+2} \le r_i$ for each $i \in [k-2]$. Therefore, $k \in \mathcal{O}(\log_{1+\varepsilon/3}\Delta)$. The size of the array A is in $\mathcal{O}(\log_{1+\varepsilon/3}\Delta)$. The space bound follows, since $\mathcal{O}(\log_{1+\varepsilon/3}\Delta) = \mathcal{O}((1/\varepsilon)\log\Delta)$.

Let T be the current time. If there does not exist a stored point q_2, then all long items are expired at time T. Therefore, all items are also expired and there are no active items at time T. Otherwise, let j be the maximum value such that $A[j] > T$. Then, the query algorithm at time T returns $r = \max\{r_2, (1+\varepsilon/3)^j\}$. By our invariant, q_2 has not yet expired at time T. As r_2 is the radius corresponding to q_2 and q_2 is still active, there exists an item p with $S(p) \ge S(q_2)$ and $E(p) > E(q_2)$ such that $d(p, q_2) = r_2$. Similarly, if j exists, there exist two items p and p' with $E(p') \ge E(p) = A[j] > T$ such that $d(p, p') \ge (1+\varepsilon/3)^j$. Hence, the diameter at time T is at least r. It remains to prove that the diameter is at most $(3+\varepsilon)r$. We first show that the following holds for every item p that is active at time T:

(1) If $S(p) \le S(q_2)$, then $d(p, q_1) \le (1+\varepsilon/3)r$ by Lemma 3.1.

(2) If $S(p) \ge S(q_2)$, then $d(p, q_2) \le (1+\varepsilon/3)r$: If $E(q_2) < E(p)$, then we have $d(q_2, p) \le r_2 \le r$, since r_2 is the radius of the smallest ball centered at q_2 that contains all items that were inserted after q_2 and expire after q_2. Otherwise, we have $T < E(p) \le E(q_2)$ and $d(q_2, p) < (1+\varepsilon/3)^{j+1} \le (1+\varepsilon/3)r$ by the definition of $A[j]$.

Now, let p and p' with $S(p) \le S(p')$ be the items that realize the diameter at time T. As all stored items are long and $E(q_1) \le T < E(p)$, it holds that $S(q_1) < S(p)$. We consider the following three cases to prove that $d(p, p') \le (3+\varepsilon)r$.

a) If $S(p') < S(q_2)$ then $d(p, p') \le d(p, q_1) + d(q_1, p') \le (2+\varepsilon)r$ by (1).

b) If $S(p) < S(q_2) \le S(p')$ then $d(p, p') \le d(p, q_1) + d(q_1, q_2) + d(q_2, p') \le (3+\varepsilon)r$ by (1), (2).

c) Otherwise $S(p) \ge S(q_2)$. Then $d(p, p') \le d(p, q_2) + d(q_2, p') \le (2+\varepsilon)r$ by (2). ◀

The same algorithm approximates also the minimum enclosing ball.

▶ **Theorem 3.2.** *There exists a deterministic expiration-streaming algorithm maintaining a $(4+\varepsilon)$-approximation of the minimum enclosing ball in a metric $\mathcal{M}$ storing $\mathcal{O}((1/\varepsilon)\log\Delta)$ words.*

3.2 $(1+\sqrt{3}+\varepsilon)$-Approximation in Euclidean Spaces

In the proof of Theorem 1.1, the case that prevents a better approximation factor is Case b), in which the diameter is realized by two items p and p' such that p is inserted before q_2 and p' after q_2. Here, our best available bound for the distance between q_1 to p' is via q_2. We show that in the Euclidean metric, this bound can be improved from $2+\varepsilon$ to $\sqrt{3}+\varepsilon$ by storing an additional point $\widetilde{q_2}$ that has a sufficiently large distance to q_2. This yields an approximation factor of $1+\sqrt{3}+\varepsilon$, which is strictly better than the current best approximation factor, even in the sliding-window model. The modified update procedure is depicted in Algorithm 2. The expiration handling and query answering remain unchanged.

In the beginning, for every newly stored item q_i we define $\widetilde{q_i} = q_i$. Consider the case that long items inserted between q_{i-1} and q_i are discarded. Further, let p be the item such that q_{i-1} and q_i become consecutive stored items at time $S(p)$. In this case, we update $\widetilde{q_i}$ to p. In contrast to Algorithm 1, the value r_i is defined by also considering distances to $\widetilde{q_i}$. We update r_i such that it holds that

(1) $d(p,q_i) \le r_i$ for all p with $S(q_i) < S(p) < E(q_i) < E(p)$,
(2) $d(p,\widetilde{q_i}) \le r_i$ for all p with $S(\widetilde{q_i}) < S(p) < E(q_i) < E(p)$, and
(3) there exists an item p with $S(\widetilde{q_i}) \le S(p) < E(q_i) < E(p)$ such that $r_i = \max\{d(p,q_i), d(p,\widetilde{q_i})\}$.

We store and reindex $\widetilde{q_i}$ and r_i along with q_i.

Algorithm 2 Euclidean diameter update procedure for inserted item (p,S,E).

if $E > E(q_k)$ **then**
  $q_{k+1} \leftarrow p$, $r_{k+1} \leftarrow 0$, $\widetilde{q}_{k+1} \leftarrow p$ `// (p,S,E) is long`
for $i \in [k]$ **do**
  if $E > E(q_i)$ **then**
    $r_i \leftarrow \max\{r_i, d(q_i,p), d(\widetilde{q_i},p)\}$
  else
    let j be such that $d(p,q_i) \in [(1+\varepsilon/3)^j, (1+\varepsilon/3)^{j+1})$
    $A[j] \leftarrow \max\{A[j], E\}$
$i \leftarrow 1$
while $i \le k-1$ **do**
  find the largest $j \ge i$ such that $(1+\varepsilon/3)\cdot r_j > r_i$
  discard q_t and r_t for all $t \in \{i+1,\dots,j-1\}$
  if $j > i+1$ **then** $\widetilde{q_j} \leftarrow p$
  $i \leftarrow \max\{j, i+1\}$
$k \leftarrow$ number of remaining elements
reindex q_i, $\widetilde{q_i}$, and r_i (keeping their order) to $q_1,\dots,q_k$, $\widetilde{q_1},\dots,\widetilde{q_k}$, and $r_1,\dots,r_k$

The additional stored points $\widetilde{q_i}$ give us a stronger version of Lemma 3.1.

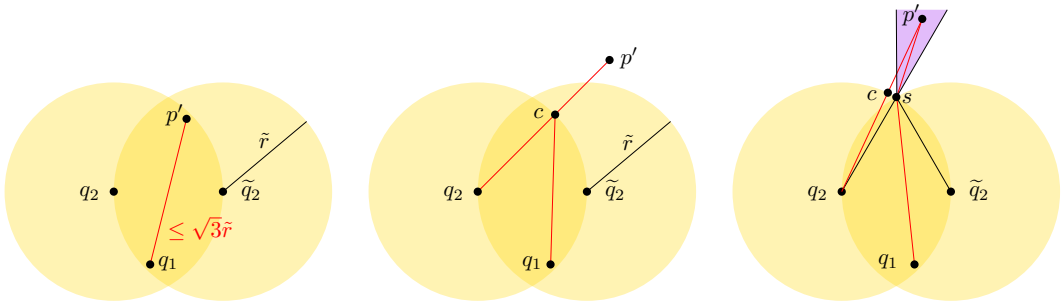

Figure 3 Visualization of Case i)-iii) (from left to right) of the proof of Theorem 1.2.

▶ **Lemma 3.3.** *For every consecutively stored items q_{i-1}, q_i, and for every item p satisfying $S(q_{i-1}) < S(p) < S(\widetilde{q}_i)$ and $E(q_{i-1}) < E(p)$, it holds that $d(q_{i-1}, p) \leq (1 + \varepsilon/3) \cdot d(\widetilde{q}_i, q_i)$.*

In the following, we use $B(p, r)$ to denote the ball of radius $r \in \mathbb{R}$ around the point $p \in \mathbb{R}^d$. We use the following two lemmas.

▶ **Lemma 3.4.** *Let $a, b, p, q \in \mathbb{R}^d$ be points such that $d(a, b) = r$ and $p, q \in B(a, r) \cap B(b, r)$. Then, it holds that $d(p, q) \leq \sqrt{3}r$.*

▶ **Lemma 3.5.** *Let q_2, s, p', and c be points and α the angle at s in the triangle spanned by the points q_2, p', and s such that $d(q_2, c) = d(q_2, s)$, $c \in \overline{q_2 p'}$, and $\alpha \geq (5/6)\pi$ (see Figure 4). Then, it holds that $d(s, p') \leq \sqrt{3}d(c, p')$.*

▶ **Theorem 1.2.** *There exists a deterministic expiration-streaming algorithm maintaining a $(1 + \sqrt{3} + \varepsilon)$-approximation of the diameter in Euclidean space storing $\mathcal{O}((1/\varepsilon) \log \Delta)$ words.*

Proof. Let r be the returned radius at time T, i.e., $r = \max\{r_2\} \cup \{(1 + \varepsilon/3)^j \mid A[j] > T\}$. By construction of r_2 and A, it holds that there are two items that are active and have distance at least r. Hence, the diameter is at least r at time T. Let p and p' with $S(p) < S(p')$ be the items that realize the diameter at time T. Similarly to the proof of Theorem 1.1, we consider different cases for the values of $S(p)$ and $S(p')$.

a) If $S(p) \geq S(q_2)$, then $d(p, p') \leq 2r$, by the same arguments as in the proof of Theorem 1.1.

b) If $S(p') < S(\widetilde{q}_2)$, then $d(p, p') \leq d(q_1, p) + d(q_1, p') \leq (2 + \varepsilon)d(\widetilde{q}_2, q_2) \leq (2 + \varepsilon)r$ by Lemma 3.3.

c) Otherwise, $S(p) < S(q_2) \leq S(\widetilde{q}_2) < S(p')$. Then, it holds that $d(p, p') \leq d(p, q_1) + d(q_1, p')$. For Case c), we show that $d(q_1, p') \leq (\sqrt{3} + \varepsilon)r$. Note that since $S(q_1) < S(p) < S(q_2)$ and $E(q_1) < E(p) \leq E(q_2)$, a long item was inserted between q_1 and q_2 but is not stored at time T. Hence, it holds that $q_2 \neq \widetilde{q}_2$. Let $\tilde{r} = d(q_2, \widetilde{q}_2)$. By Lemma 3.3, it holds that $d(q_1, q_2) \leq (1+\varepsilon/3)\tilde{r}$ and $d(q_1, \widetilde{q}_2) \leq (1+\varepsilon/3)\tilde{r}$, i.e., $q_1 \in B(q_2, (1+\varepsilon/3)\tilde{r}) \cap B(\widetilde{q}_2, (1+\varepsilon/3)\tilde{r})$. We assume that $d(\widetilde{q}_2, p') \leq d(q_2, p')$ as the other case follows analogously. To prove $d(q_1, p') \leq (\sqrt{3} + \varepsilon)r$, we distinguish between three cases (see Figure 3).

i) If $d(q_2, p') \leq \tilde{r}$, then it follows that $d(\widetilde{q}_2, p') \leq d(q_2, p') \leq \tilde{r}$. Hence, we have $p' \in B(q_2, \tilde{r}) \cap B(\widetilde{q}_2, \tilde{r})$. By Lemma 3.4 it holds that $d(q_1, p') \leq (\sqrt{3}+\varepsilon/\sqrt{3})\tilde{r} \leq (\sqrt{3}+\varepsilon/\sqrt{3})r$.

Otherwise, define c to be the point on $\overline{q_2 p'}$ with $d(c, q_2) = \tilde{r}$.

ii) If $d(c, \widetilde{q}_2) \leq \tilde{r}$, then $d(q_1, c) \leq (\sqrt{3} + \varepsilon/\sqrt{3})\tilde{r}$ by Lemma 3.4. Hence,

$$\begin{aligned} d(q_1, p') &\leq d(q_1, c) + d(c, p') = d(q_1, c) + d(q_2, p') - d(q_2, c) \\ &\leq (\sqrt{3} + \varepsilon/\sqrt{3})\tilde{r} + r - \tilde{r} \leq (\sqrt{3} + \varepsilon/\sqrt{3})r. \end{aligned}$$

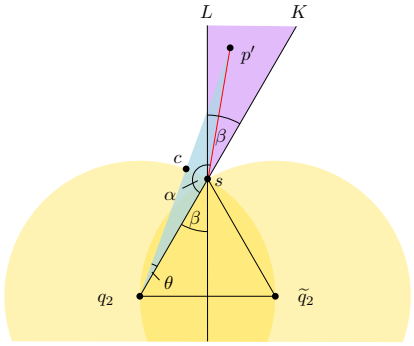

Figure 4 Visualization of Case iii) in Theorem 1.2. The angle β is $\pi/6$ because the triangle spanned by q_2, $\widetilde{q}_2$ and s is equilateral. Because p' lies in the sector spanned by L and K at s (shown in purple), it follows that $\alpha \geq \pi - \beta = (5/6)\pi$.

iii) Otherwise, it holds that $d(c, \tilde{q}_2) > \tilde{r}$. Then, consider the plane containing q_2, $\widetilde{q}_2$ and p'. Let s be the point in this plane with $d(q_2, s) = d(\widetilde{q}_2, s) = \tilde{r}$ that is closest to p' (see Figure 4). Then, $d(q_1, s) \leq (\sqrt{3} + \varepsilon/\sqrt{3})\tilde{r}$ by Lemma 3.4. We use Lemma 3.5 to obtain a bound on $d(s, p')$. Let α be the angle at s in the triangle spanned by the points q_2, p', and s. Further, let L denote the perpendicular bisector of $\overline{q_2\widetilde{q}_2}$ and let β be the smaller angle between the line L and the line containing q_2 and s denoted by K. We have $\beta = \pi/6$ because the triangle spanned by q_2, $\widetilde{q}_2$ and s is equilateral with angle 2β. Because $d(c, \widetilde{q}_2) > \tilde{r} = d(c, q_2)$, it follows that c and q_2 lie on the same side of L, which implies that p' lies on the same side of the line K as c. On the other hand, p' lies on the same side of L as $\widetilde{q}_2$ because $d(\widetilde{q}_2, p') \leq d(q_2, p')$. Hence, p' lies in the sector spanned by K and L at s away from q_2 (colored in purple in Figure 4). Therefore, it follows that $\alpha \geq \pi - \beta = (5/6)\pi$. Hence, by Lemma 3.5 we have $d(s, p') \leq \sqrt{3}d(c, p')$ and thus

$$d(q_1, p') \leq d(q_1, s) + d(s, p') \leq (\sqrt{3} + \varepsilon/\sqrt{3})\tilde{r} + \sqrt{3}d(c, p') \leq (\sqrt{3} + \varepsilon/\sqrt{3})r.$$

Therefore, in Case c) it holds that $d(p, p') \leq d(p, q_1) + d(q_1, p') \leq (1 + \sqrt{3} + \varepsilon)r$. ◀

Again, the same algorithm can be used to approximate the minimum enclosing ball in Euclidean space $\mathbb{R}^d$.

▶ **Theorem 3.6.** *There exists a deterministic expiration-streaming algorithm maintaining a $(1 + \sqrt{3} + \varepsilon)$-approximation of Euclidean Minimum Enclosing Ball storing $\mathcal{O}((1/\varepsilon) \log \Delta)$ words.*

4 Approximating k-Center in Expiration Streams

The work of [17] gives a $(6 + \varepsilon)$-approximation for metric k-center in the sliding-window streaming model storing $\mathcal{O}((k/\varepsilon) \log \Delta)$ words. It is not hard to verify that this algorithm also works under the consistent expiration streaming model with the same space bound and same approximation factor. For general expirations, the same algorithm still yields a $(6 + \varepsilon)$-approximation. However, the required space can be arbitrarily large. Below, we give an algorithm that computes a $(6k + 2 + \varepsilon)$-approximation in the general expiration model storing $\mathcal{O}((1/\varepsilon)k^2 \log \Delta)$ words. Because the sliding-window algorithm by [17] is used as a subroutine, we briefly summarize its main ideas.

4.1 Preliminaries on the Sliding-Window Algorithm

For a given estimate γ of the solution value, the k-center algorithm by [17] operates as follows. The algorithm maintains a set A of at most $k+1$ *attraction points*. Each attraction point a is associated with a ball of radius 2γ centered at a. The algorithm guarantees that no attraction point lies inside the ball of another attraction point. The *representative* $R(a)$ of a is the longest-living point inside this ball inserted while a is active. The set of all representatives is denoted by R. When a expires, its representative $R(a)$ is kept in memory until it itself expires. The representatives of expired attraction points are called *orphans* and are stored in a set O.

When a new point p_t is inserted and it lies in the ball of at least one attraction point, then p_t is considered *covered* and the representative $R(a)$ might be updated. If p_t is not covered, it is inserted as a new attraction point, with itself as the representative. If adding p_t increases the number of stored attraction points above $k+1$, then the algorithm identifies the shortest-living attraction point a_{old} and discards it (and the representative $R(a_{\text{old}})$ becomes an orphan). Finally, if the number of stored attraction points is greater than k, then all orphans that do not outlive a_{old} are discarded.

For every estimate $\gamma \in \{(1+\varepsilon)^i \mid i \in [\lceil \log_{1+\varepsilon} \Delta \rceil]\}$, an instance of the algorithm is run in parallel. To answer a query, the algorithm iterates over the estimates in the ascending order. For each estimate γ, the algorithm attempts to construct a solution C by picking an arbitrary point in $A \cup R \cup O$ and then greedily adding any point $p \in A \cup R \cup O$ with $d(p, C) > 2\gamma$. If $|C| > k$, then it is a certificate for $\text{OPT} > \gamma$. Otherwise, C is returned. It can be shown that C is a 6γ-coreset for the stream because $R \cup O$ is a 4γ-coreset.

The main insight regarding the space bound is that the orphans that are still in memory belong to a subset of the $k+1$ most recently discarded attraction points. Any orphan o that belongs to an even older attraction point must have been inserted before the most recently discard attraction point a, because the algorithm stores at most $k+1$ attraction points at any time. Due to the consistent expiration property of the stream, o expires before a, so o is discarded at the latest when a is discarded. In the general expiration model, consistent expirations are not guaranteed. This is the main reason why the size of O cannot be bounded, because then the orphans belonging to arbitrarily old attraction points may still be in memory.

4.2 Overview of Our Algorithm

We present an algorithm that, given a parameter $K \in \mathbb{N}$, solves k-center under the (general) expiration streaming model for every $k \leq K$. As with [17], our algorithm solves the decision version of the problem for a given estimate γ, and this decision algorithm is then run in parallel for different estimates. We observe that in the algorithm by [17], the bound on the number of stored orphans does not require that the entire stream has consistent expirations, only that the attraction points are long. This ensures that the attraction points always outlive the orphans of already discarded attraction points. In the general expiration model, we cannot guarantee that all attraction points are long. Therefore, we split the stream into K substreams. In each substream i, we run a modified version of the algorithm by [17], storing the sets A_i, R_i and O_i of attraction points, representatives and orphans. When a point is inserted, it is placed into the first substream in which it is long or covered by the 2γ-ball of an existing attraction point. This ensures that within each substream, all attraction points are long and the number of orphans is bounded.

The main challenge lies in ensuring that k substreams are sufficient to approximate k-center. If there is a point p that does not fit into any of the first k substreams, we maintain a certificate that OPT $> \gamma$ until p expires. This certificate consists of the longest-living point from each substream, plus p itself. Hence, we need to ensure that these points are pairwise more than 2γ apart. This is done by discarding points in different substreams if they are too close to each other: a point p from substream i is now also considered covered (and is therefore discarded) if it expires not later than a stored point p' from a lower substream $j < i$ with $d(p, p') \leq 2\gamma$. To ensure that it is sufficient to consider lower substreams, we maintain the invariant that the substreams are ordered in descending order of their longest-living stored point. If the longest-living point in substream i becomes covered, the invariant is violated. It is restored by moving the longest-lived point among all substreams $j > i$ down to substream i (which may lead to cascading effects).

A side effect of the expanded covering rule is that it causes the approximation factor to be dependent on k. When a representative or orphan r is discarded from substream i because it is covered by a point p from a lower substream, then the 4γ-coreset for substream i may be destroyed. If we replace r with p in the coreset, its approximation guarantee increases to 6γ. Moreover, the effect may be cascading because r may later be discarded when it is covered by another point that is inserted into an even lower substream.

4.3 Detailed Algorithm Description

In this section, we give a detailed description of the k-center algorithm under the expiration model. In addition to the sets A_i, R_i and O_i, each substream i also maintains two values e_i and t_i. The value e_i is the expiration time of the longest-living point stored in substream i, which is used to maintain the ordering of the substream. The value t_i is the earliest time such that every item p that is still active was originally inserted into a substream $j \leq i$ (note that it may have since been moved to a lower substream). In other words, until time t_i there is at least one active item that was not originally inserted into any of the substreams 1 to i, and t_i is the last expiration time among all such items. We show that as a consequence of this definition, until time t_i every solution to i-center has radius greater than γ.

Finally, the algorithm maintains a time $\hat{t}$, which indicates that until time $\hat{t}$, there is a set of at least $K + 1$ active points that were at some time all included in the set A_i for some substream i. Note that if $|A_i| > K$ holds, then A_i is a certificate that all k-center solutions have radius greater than γ until the first point from the set expires. However, points from A_i may be discarded before they expire if they are covered by a newly inserted point from a lower substream, which destroys the certificate. Hence, we store $\hat{t}$ to indicate that the certificate still exists, even if some the points are no longer stored.

Invariants. The algorithm maintains the following invariants at all times.

(1) For all $i \in [K]$, it holds that $e_i = 0$ if $A_i \cup R_i \cup O_i = \emptyset$, and $e_i = \max\{E(p) \mid p \in A_i \cup R_i \cup O_i\}$ otherwise.
(2) It holds that $e_1 \geq e_2 \geq \cdots \geq e_K$.
(3) For all $a \in \bigcup_{i \in [K]} A_i$, it holds that $d(a, R(a)) \leq 2\gamma$.
(4) For all distinct $a, a' \in \bigcup_{i \in [K]} A_i$, it holds that $d(a, a') > 2\gamma$.
(5) For all distinct $q_i \in R_i \cup O_i$ and $q_j \in R_j \cup O_j$ with $i \leq j$ and $E(q_i) \geq E(q_j)$, it holds that $d(q_i, q_j) > 2\gamma$.

Initialization. The data structures are initialized as follows.

- $A_i, R_i, O_i \leftarrow \emptyset$ for all $i \in [K]$,
- $e_i, t_i \leftarrow 0$ for all $i \in [K]$, and
- $\hat{t} \leftarrow 0$.

Item Insertion. When a point p is added to the stream, Algorithm 3 is called. We iterate over the different substreams with index $i = 1$ to K until we find a substream to which p can be added. If a point $q = R(a)$ in $R_i \cup O_i$ exists that expires later than p or at the same time as p and has distance at most 2γ to p, then p is added to substream with index i and p is covered by q. Note that as p expires not later than q, we do not have to update $R(a)$ and we do not store p. Further, if an attraction point $a \in A_i$ exists with $d(a, p) \leq 2\gamma$, we add p to substream i and p gets covered by a and check whether we have to update the representative point $R(a)$. To maintain Invariant 5 when the representative point $R(a)$ is updated, we run `DiscardCoveredPoints`.

If p lives longer than all points currently in substream i, then p would be a long item in substream i with the definition of Section 3. In the case that p has distance greater than 2γ to all points in A_i, it becomes an attraction point in A_i and we set $R(p) = p$. Then, `AddAttractionPoint` is called to maintain Invariant 4 and to bound the storage size. If p was not added to substream i at the end of iteration i, we update the earliest time t_i at which all active points are originally inserted into substreams with index at most i to $\max\{t_i, E\}$.

The rest of Algorithm 3 (lines 13–27) ensures that Invariant 2 remains true. So, the longest surviving point in substream i lives at least as long as the longest surviving point in substream $i + 1$ for all $i \in [K - 1]$, i.e., $e_1 \geq e_2 \geq \cdots \geq e_K$. We iterate over all substreams $i = 1$ to K and find in iteration i the longest surviving point q in the substreams with index $j = i$ to K. If q is not already contained in substream i, we remove it from its substream and add it to substream i as a new attraction point. In this case, its corresponding attraction point is discarded if it was still stored.

Item Expiration. If a point in O_i for any $i \in [K]$ expires, it is removed from O_i. If a point $a \in A_i$ for any $i \in [K]$ expires, it is removed from A_i and $R(a)$ is moved from R_i to O_i. Note that an item $R(a)$ is contained in R_i only if $a \in A_i$. As $E(R(a)) \geq E(a)$, an item $R(a)$ in R_i can only expire if $E(a) = E(R(a))$. Then, a is removed from A_i, $R(a)$ is added to O_i, and as $R(a) \in O_i$ expires it is removed from O_i.

Query Algorithm. As in [17], we run the algorithm in parallel for every estimate $\gamma \in \{(1+\varepsilon)^i \mid i \in [\lceil \log_{1+\varepsilon} \Delta \rceil]\}$. A query for $k \leq K$ and time T is answered by iterating through the estimates in ascending order. If $\max\{t_k, \hat{t}\} \geq T$ for the current estimate γ, then it is skipped. Otherwise, we first choose an arbitrary point $q \in \bigcup_{i \in [k]} A_i \cup R_i \cup O_i$ and add it to the center set C. Then, we greedily add any point $q \in \bigcup_{i \in [k]} A_i \cup R_i \cup O_i$ to C with distance greater than 2γ to all points in C. If $|C| > k$ at termination, we have a certificate that no solution with radius γ can exist. For the smallest value of γ with $|C| \leq k$, we return C and show that this gives a $(6k + 2 + \varepsilon)$-approximation to metric k-center.

Notation. We denote with A_i^T, R_i^T, and O_i^T the sets A_i, R_i, and O_i, with $R^T(a)$ the point $R(a)$, and with e_i^T and t_i^T the values e_i and t_i at time T. If an attraction point a is not stored anymore at time T, we denote with $R^T(a)$ the representative of a at the time a is removed from $\bigcup_{i \in [K]} A_i$. In addition, we define $R^\infty(a)$ to be the last representative point of a.

4.4 Storage Size

In this section, we prove a space bound for the stored sets A_i, R_i, and O_i.

▶ **Observation 4.1.** *At the time when a is added to A_i, it holds that $e_i = E(a)$.*

Algorithm 3 k-center update procedure for inserted item p.

```
for i = 1, ..., K do
    if ∃q ∈ R_i ∪ O_i with E(q) ≥ E(p) and d(p, q) ≤ 2γ then
        return                                   // p is added to substream i
    if ∃a ∈ A_i with d(p, a) ≤ 2γ then
        if E(p) > E(R(a)) then
            R_i ← {p} ∪ R_i \ {R(a)}, R(a) ← p
            DiscardCoveredPoints(p, i)
        return                                   // p is added to substream i
    if E(p) > e_i then
        AddAttractionPoint(p, i)
        return                                   // p is added to substream i
    t_i ← max{t_i, E}                            // p is not added to substream i
// reorder substreams
for i = 1, ..., K do
    if ⋃_{j=i}^{K} R_j ∪ O_j = ∅ then
        e_i ← 0                                  // substream i is empty
        continue
    e_i ← max{E(q) | q ∈ ⋃_{j=i}^{K} R_j ∪ O_j}
    Let j ≥ i be minimal such that q ∈ R_j ∪ O_j with E(q) = e_i
    Let q ∈ R_j ∪ O_j such that E(q) = e_i
    if i ≠ j then
        if q ∈ R_j then
            Let a be such that R(a) = q
            R_j ← R_j \ q, A_j ← A_j \ a          // discard a and q from substream j
            AddAttractionPoint(q, i)
        else
            O_j ← O_j \ q
            AddAttractionPoint(q, i)
```

▶ **Lemma 4.2.** *The size of each set A_i, R_i, and O_i is at most $K + 1$ for any $i \in [K]$.*

4.5 Correctness

To show correctness, we begin with proving that Invariants (1)-(5) hold. The next observation shows that items are moved only from substreams with higher index to substreams with lower index. This is a crucial property to get a bound on the approximation factor.

▶ **Observation 4.3.** *Let $q \in A_i^t \cup R_i^t \cup O_i^t$ for some time t. For all $T > t$ and $i < j \leq K$, it holds that $q \notin A_j^T \cup R_j^T \cup O_j^T$.*

▶ **Observation 4.4.** *Invariants (1)-(3) hold during the algorithm.*

▶ **Observation 4.5.** *Let $q \in R_i \cup O_i$. Then, for all $a \in \bigcup_{j \in [i-1]} A_j$, it holds that $d(q, a) > 2\gamma$.*

▶ **Lemma 4.6.** *Invariants (4) and (5) hold during the algorithm.*

Algorithm 4 Subroutines of the k-center update procedure for inserted item p.

Procedure `DiscardCoveredPoints`(p, i)
  for $j = i+1, \dots, K$ **do**
    `// F is moved to substream i`
    $F \leftarrow \{a \in A_j \mid E(R(a)) < E(p)$ and $(d(a,p) \leq 2\gamma$ or $d(R(a),p) \leq 2\gamma)\}$
    $A_j \leftarrow A_j \setminus F$
    $R_j \leftarrow R_j \setminus \{R(a) \mid a \in F\}$
    `// G is moved to substream i`
    $G \leftarrow \{o \in O_j \mid E(o) < E$ and $d(o,p) \leq 2\gamma\}$
    $O_j \leftarrow O_j \setminus G$
Procedure `AddAttractionPoint`(a, i)
  $A_i \leftarrow A_i \cup \{a\}$, $R(a) \leftarrow a$, $R_i \leftarrow R_i \cup \{R(a)\}$
  `DiscardCoveredPoints`(a, i)
  $e \leftarrow \min_{a \in A_i} E(a)$
  Let $a_{\text{old}} \in A_i$ be such that $E(a) = e$
  if $|A_i| > K+1$ **then**
    $A_i \leftarrow A_i \setminus \{a_{\text{old}}\}$, $R_i \leftarrow R_i \setminus \{R(a_{\text{old}})\}$, $O_i \leftarrow O_i \cup \{R(a_{\text{old}})\}$
  if $|A_i| > K$ **then**
    $O_i \leftarrow \{o \in O_i \mid E(o) \geq E(a_{\text{old}})\}$
    $\hat{t} \leftarrow \max\{\hat{t}, E(a_{\text{old}})\}$

Next we show two properties that ensure that the solution to k-center has radius greater γ.

▶ **Lemma 4.7.** *Let T be the current time. If $|\bigcup_{i\in[K]} A_i| > k$ or $\hat{t} > T$, then the solution for the metric k-center has radius greater than γ.*

▶ **Lemma 4.8.** *Let T be the current time and $k \in [K]$. If $t_k > T$, then the solution to the metric k-center has radius greater than γ.*

To handle the case that a solution of radius γ exists, we prove some distance bounds between different points of the stream.

▶ **Lemma 4.9.** *Let $R^\infty(a)$ be the last representative point of an attraction point a. Further, let t be the time and i the index such that $R^\infty(a) \in R_i^{t-1} \cup O_i^{t-1}$ and $R^\infty(a) \notin \bigcup_{j\in[K]} R_j^t \cup O_j^t$. Then one of the following holds:*

(i) $t = E(R^\infty(a))$,
(ii) $\hat{t}^T \geq E(R^\infty(a))$ *for all* $t \leq T$, *or*
(iii) *there is an* $\ell < i$ *and a point* $p \in R_\ell^t \cup O_\ell^t$ *such that* $E(R^\infty(a)) \leq E(p)$ *and* $d(a,p) \leq 4\gamma$.

▶ **Lemma 4.10.** *Let a be an attraction point and let $R^\infty(a)$ be its last representative point. Further, let t be the time and i the index such that $R^\infty(a) \in R_i^{t-1} \cup O_i^{t-1}$ and $R^\infty(a) \notin \bigcup_{j\in[K]} R_j^t \cup O_j^t$. Then, for any time $t \leq T < E(R^\infty(a))$ such that $\hat{t}^T \leq T$, there exists an $\ell \leq i$ and a point $r \in R_\ell^T \cup O_\ell^T$ such that $d(a,r) \leq (i-\ell)6\gamma + 2\gamma$.*

Next we obtain a distance bound for any active point p that expires later than $\max\{t_k, \hat{t}\}$.

▶ **Lemma 4.11.** *Let T be the current time and $k \in [K]$. If $\max\{t_k^T, \hat{t}^T\} < T$, then for any item p active at time T, i.e., $S(p) \leq T < E(p)$, there exists a point $r \in \bigcup_{i\in[k]} R_i^T \cup O_i^T$ with $d(p,r) \leq 6k\gamma$.*

Running the algorithm described in Section 4.3 for all values of

$$\gamma \in \{(1+\varepsilon)^i \mid i = 1, \ldots, \lceil \log_{1+\varepsilon} \Delta \rceil\}$$

in parallel results in the following theorem.

▶ **Theorem 1.3.** *There exists a deterministic expiration-streaming algorithm that stores $\mathcal{O}((K^2/\varepsilon)\log\Delta)$ words and maintains a $(6k+2)(1+\varepsilon)$-approximate solution for k-center, for every $k \leq K$, in a general metric $\mathcal{M}$.*

▶ **Corollary 4.12.** *In every metric space, there exists an algorithm in the expiration streaming model that can return at any time and for every value $k \leq K$*

(1) *a $(6k+2+\varepsilon)$-approximate solution for k-center storing $\mathcal{O}((K^3/\varepsilon)\log\Delta)$ words,*

(2) *a $(6k+3+\varepsilon)$-approximate solution for k-center storing $\mathcal{O}((K^2/\varepsilon + K^3)\log\Delta)$ words, and*

(3) *a $(7k)$-approximate solution for k-center with $k \geq 3$ storing $\mathcal{O}(K^2\log\Delta)$ words.*

References

1 Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Trans. Database Syst.*, 38(4):26, 2013. doi:10.1145/2500128.

2 Pankaj K. Agarwal and Cecilia Magdalena Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002. doi:10.1007/s00453-001-0110-y.

3 Pankaj K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. *Algorithmica*, 72(1):83–98, 2015. doi:10.1007/s00453-013-9846-4.

4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: Sparsification, spanners, and subgraphs. In *31st Symposium on Principles of Database Systems (PODS)*, pages 5–14. ACM, 2012. doi:10.1145/2213556.2213560.

5 Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, SODA '02, pages 633–634. ACM/SIAM, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545465.

6 Jon Louis Bentley and James B. Saxe. Decomposable searching problems I. Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980. doi:10.1016/0196-6774(80)90015-2.

7 Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P. Woodruff, and Samson Zhou. Near optimal linear algebra in the online and sliding window models. In *61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 517–528. IEEE, 2020. doi:10.1109/FOCS46700.2020.00055.

8 Vladimir Braverman, Elena Grigorescu, Harry Lang, David P. Woodruff, and Samson Zhou. Nearly optimal distinct elements and heavy hitters on sliding windows. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, volume 116 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.7.

9 Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering problems on sliding windows. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1374–1390. SIAM, 2016. doi:10.1137/1.9781611974331.ch95.

10 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*, pages 283–293. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.55.

11 Matteo Ceccarello, Andrea Pietracaprina, and Geppino Pucci. Solving k-center clustering (with outliers) in MapReduce and streaming, almost as accurately as sequentially. *Proc. VLDB Endow.*, 12(7):766–778, 2019. doi:10.14778/3317315.3317319.

12 Timothy M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM Journal on Computing*, 32(3):700–716, 2003. doi:10.1137/S0097539702404389.

13 Timothy M. Chan and Vinayak Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. *Comput. Geom.*, 47(2):240–247, 2014. doi:10.1016/J.COMGEO.2013.05.007.

14 Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. doi:10.1137/S0097539702418498.

15 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004. doi:10.1016/S0304-3975(03)00400-6.

16 Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997. doi:10.1006/jcss.1997.1534.

17 Vincent Cohen-Addad, Chris Schwiegelshohn, and Christian Sohler. Diameter and k-center in sliding windows. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55 of *LIPIcs*, pages 19:1–19:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.19.

18 Graham Cormode and Shan Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005. doi:10.1016/j.jalgor.2003.12.001.

19 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. doi:10.1137/S0097539701398363.

20 Mark de Berg, Leyla Biabani, and Morteza Monemizadeh. k-center clustering with outliers in the MPC and streaming model. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2023*, pages 853–863. IEEE, 2023. doi:10.1109/IPDPS54959.2023.00090.

21 Mark de Berg, Morteza Monemizadeh, and Yu Zhong. k-center clustering with outliers in the sliding-window model. In *29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.13.

22 David Dobkin and Subhash Suri. Maintenance of geometric extrema. *Journal of the ACM*, 38(2):275–298, 1991. doi:10.1145/103516.103518.

23 Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2005. doi:10.1007/S00453-004-1105-2.

24 Shiyuan Feng, William Swartworth, and David Woodruff. Tight bounds for heavy-hitters and moment estimation in the sliding window model. In *52nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 334 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 75:1–75:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ICALP.2025.75.

25 Sudipto Guha. Tight results for clustering and summarizing data streams. In *12th International Conference on Database Theory (ICDT)*, volume 361, pages 268–275. ACM, 2009. doi:10.1145/1514894.1514926.

26 Magnús M. Halldórsson, Nicolaos Matsakis, and Pavel Veselý. Streaming Diameter of High-Dimensional Points. In *33rd Annual European Symposium on Algorithms (ESA 2025)*, volume 351 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 58:1–58:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.58.

27 Piotr Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 539–545, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644200.

28 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *36th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 373–380, 2004. doi:10.1145/1007352.1007413.

29 Shaofeng H.-C. Jiang, Robert Krauthgamer, and Shay Sapir. Moderate dimension reduction for k-center clustering. In *40th International Symposium on Computational Geometry (SoCG 2024)*, volume 293 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 64:1–64:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SoCG.2024.64.

30 Sang-Sub Kim and Hee-Kap Ahn. An improved data stream algorithm for clustering. *Comput. Geom.*, 48(9):635–645, 2015. doi:10.1016/J.COMGEO.2015.06.003.

31 Robert Krauthgamer and David Reitblat. Almost-smooth histograms and sliding-window graph algorithms. *Algorithmica*, 84(10):2926–2953, 2022. doi:10.1007/s00453-022-00988-y.

32 Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, volume 5171 of *Lecture Notes in Computer Science*, pages 165–178. Springer, 2008. doi:10.1007/978-3-540-85363-3_14.

33 Benwei Shi, Zhuoyue Zhao, Yanqing Peng, Feifei Li, and Jeff M. Phillips. At-the-time and back-in-time persistent sketches. In *International Conference on Management of Data (SIGMOD)*, pages 1623–1636. ACM, 2021. doi:10.1145/3448016.3452802.

34 Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985. doi:10.1145/3147.3165.

35 Yanhao Wang, Yuchen Li, and Kian-Lee Tan. Coresets for minimum enclosing balls over sliding windows. In *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 314–323. ACM, 2019. doi:10.1145/3292500.3330826.

36 Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. Persistent data sketching. In Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 795–810. ACM, 2015. doi:10.1145/2723372.2749443.

37 David P. Woodruff and Taisuke Yasuda. Online Lewis weight sampling. *ACM Transactions on Algorithms*, 21(4):1–50, 2025. doi:10.1145/3715127.

38 Hamid Zarrabi-Zadeh and Timothy M. Chan. A simple streaming algorithm for minimum enclosing balls. In *Proceedings of the 18th Annual Canadian Conference on Computational Geometry, CCCG 2006*, 2006. URL: http://www.cs.queensu.ca/cccg/papers/cccg36.pdf.