Competitive analysis of aggregate max in windowed streaming*

Luca Becchetti[1] and Elias Koutsoupias[2]

[1] "Sapienza" University of Romebecchett@dis.uniroma1.it
[2] University of Athenselias@di.uoa.gr

Abstract. We consider the problem of maintaining a fixed number k of items observed over a data stream, so as to optimize the maximum value over a fixed number n of recent observations. Unlike previous approaches, we use the competitive analysis framework and compare the performance of the online streaming algorithm against an optimal adversary that knows the entire sequence in advance. We consider the problem of maximizing the *aggregate max*, i.e., the sum of the values of the largest items in the algorithm's memory over the entire sequence. For this problem, we prove an asymptotically tight competitive ratio, achieved by a simple heuristic, called partition-greedy, that performs stream updates efficiently and has almost optimal performance. In contrast, we prove that the problem of maximizing, for every time t, the value maintained by the online algorithm in memory, is considerably harder: in particular, we show a tight competitive ratio that depends on the maximum value of the stream. We further prove negative results for the closely related problem of maintaining the aggregate minimum and for the generalized version of the aggregate max problem in which every item comes with an individual window.

1 Introduction

In streaming applications, a sequence or *stream* of items arrives online to be processed by an algorithm with memory much smaller than the length of the sequence or the cardinality of the universe of possible items. The objective of the algorithm is to maintain some statistical information about the stream. For many statistical properties, only some recent part—called window—of the stream is important. A typical assumption is that the window of interest has fixed size n. In many cases of practical interest the window size is much larger than the available memory size. This is the sliding window streaming model [12, 23].

A typical application in the sliding window streaming model is the following: A sensor measures continuously the temperature and maintains the maximum temperature of the last hour. It should be clear that with limited memory, it is not possible for the sensor to know at every time step the maximum temperature of the past hour. The question is "how well can this task be done with limited memory?" This is the main question that we address in this work. We naturally treat this as an online problem and we use competitive analysis [6].

Our main focus is on a simple online problem defined by two parameters: n, the window size, and k, the memory size of the algorithm. An online algorithm with memory of size k processes a sequence of positive items with values $a_1, a_2, \ldots$. At every time t, the algorithm maintains in each of its k memory slots an item from the recent window $\{a_{t-n+1}, \ldots, a_t\}$ of length n (for simplicity, we assume that the stream extends to negative times with $a_t = 0$ for $t \leq 0$). The objective at every time t is to maximize the maximum value of the items in memory. More precisely, if g_t denotes the maximum value of the items in memory at time t, the objective of the algorithm is to maximize $\sum_t g_t$. We call this the *online aggregate max problem* and study it using competitive analysis.

This kind of sliding window streaming problems have been addressed before in the literature on streaming algorithms. In fact, much more general questions have been answered with spectacular success. For example, the paper by Datar, Gionis, Indyk, and Motwani [12] showed how to extend the pioneering paper of Alon, Matias, and Szegedy [2] to estimate many statistics on sliding windows; these results were improved and extended in many directions (for example, in [7]). The difference between this body of work and our work lies mainly in the approach: we use competitive analysis to study the performance of an algorithm. To do this, we consider online algorithms with memory of fixed size k (in which each memory position can hold one item) and then ask how good its competitive ratio can be, whereas previous approaches impose a bound on the accuracy, described by a parameter ϵ, and then try to minimize the amount of memory necessary to guarantee (worst case or with high probability) the desired accuracy.

We summarize the main differences of our approach with the approaches in the existing literature: We consider mainly the aggregate value of the maximum value in memory (although we also give tight bounds

* Partially supported by EU Projects AEOLUS and FRONTS and by MIUR FIRB project N. RBIN047MH9: "Tecnologia e Scienza per le reti di prossima generazione"

for the worst case). This is a weaker objective than requiring the maximum value in memory to be *always* within a fraction ϵ from the optimal. The aggregate objective is more appropriate for economic applications in which we care about the total value, rather than the worst case. We measure memory in slots not bits; each slot can keep exactly one item. This assumption is more realistic in practical settings where the items carry large satellite information. We fix the memory size to some constant k and we prefer the competitive ratio to depend mainly on k, not the window length n. As it is usually the case with competitive analysis our focus is on the information-theoretic constraints rather than the computational complexity constraints; in particular, we don't pay much attention to the execution time of the online algorithms, although all our algorithms are very efficient. As a result, the techniques of competitive analysis seem to be more appropriate to address the kind of questions we are interested in than those usually adopted in streaming algorithms, which often rely on embeddings (e.g., [2]).

Our results are not directly comparable to the existing results on streaming algorithms. To make this point clear, consider the results in [10] where they study streaming algorithms that estimate the diameter of a set of points. Their result for the 1-dimensional case (which resembles a lot our question about max) is roughly as follows: There is an efficient ϵ-approximation algorithm with memory that stores of $O(\frac{1}{\epsilon}\log M)$ points, where M is the maximum diameter. Our result is that for every k, there is a competitive algorithm of memory size of k slots with approximation ratio $1 + O(1/k)$. There is no direct translation of one result to the other, mainly because one is a worst-case result while the other is an aggregate result. The competitive ratio implied by the above result in [10] is $O(\frac{\log M}{k})$, which is very high compared to our result (and depends on the values of the stream).

Our contributions. We study the online aggregate max problem and give tight results on its competitive ratio: We show that the competitive ratio of online algorithms with memory size k is surprisingly low: $1+\Theta(1/k)$. The constants inside our lower and upper bounds inside the Θ expression are quite different. In particular, for upper bound, we give an intuitive deterministic online algorithm (which we call partition-greedy), and show that it has competitive ratio $k/(k-1)$. The lower bound is technically more interesting: We show that every randomized online algorithm has competitive ratio $1 + \Omega(\frac{1}{66k})$. These bounds hold in the strongest possible sense: the upper bound holds even against offline algorithms with unlimited memory, whereas the lower bound holds for offline algorithms with memory k.

We also study from a competitive point of view the *anytime max problem* in the sense that the competitive ratio is the worst-case ratio over all times t of the optimal value at time t over the maximum value of the online algorithms memory at time t. Naturally, this tighter objective results in much higher competitive ratio which is not independent of the values of the stream. We show a tight bound on the competitive ratio which is $\sqrt[k+1]{M}$, where M is the maximum value in the stream.

We also explore natural extensions of the online aggregate max problem. An interesting extension comes from viewing the items of the stream as expiring after exactly n steps. We ask the question what happens when each item has its own expiration time (selected by an adversary and revealed to the online algorithm together with its value). We show that in this case the competitive ratio is unbounded. We also explore the competitive ratio for the aggregate min problem and show that it is completely different than the aggregate max objective because its competitive ratio is unbounded.

Related work. We summarize here some relevant publications.

In [12], the authors introduce the sliding window model for streaming. They also prove that maintaining the maximum/minimum over a size n sliding window requires at least $n\log(M/n)$, where M is the size of the universe, i.e., the set of all possible values for items in the stream.

In [16] the authors consider the problem of estimating the diameter of a set of points in the windowed streaming model. The authors address mainly the 2-dimensional case. This is a generalization of the problem of maintaining the maximum. The authors propose streaming algorithms that use polylogarithmic space with respect to the window size and the diameter.

The result above was improved in [10]. In particular, for the 1-dimensional case, the authors provide an algorithm that maintains the diameter of a set of points with ϵ-approximation storing in its memory $O(\frac{1}{\epsilon}\log M)$ points, M being the ratio between the diameter and the minimum distance between any non-coincident pair of points. Notice that the memory in the above expression is measured in number of points not number of bits, the same with this work.

In [7], the authors recast these results in a more general framework, proving among others that for a large class of "smooth" functions, including sum, distance, maximum, and minimum, it is possible to maintain polylogarithmic space sketches that allow to estimate their value within a sliding window with ϵ-approximation.

Table 1. Notation

Symbol	Meaning
a_j	The j-th item observed in the stream
n	Window size
k	Max. no. items kept by algorithm
M	Maximum item value
g_t	Max. value in algorithm's memory at t
m_t	Max. value in algorithm's memory at t
$\mathbf{r}(k,n)$	Competitive ratio

2 Model and notation

Windowed streaming. We consider a streaming model in which the algorithm observes a sequence $\{a_1, a_2, \ldots\}$ of items over time, a_t being the item observed during the t-th *time step*. In practical settings, every item is a record consisting of several possible fields, including a distinguished *value* field. In the sequel we assume without loss of generality that the value field of a_j belongs to the (integer) interval $[1, M]$ for some $M > 1$ and we denote by a_j *both the j-th item and its value*. We assume that, at every time t, we are only interested in maintaining statistics over the (values of the) last n observations, i.e., over the window of items observed in the interval $\{t-n+1, \ldots, t\}$. For simplicity when $t < n$, we assume that the sequence extends to negative times with value 0. In the sequel, g_t denotes the item of maximum value maintained by the algorithm at time t and m_t denotes the maximum value in the window at time t.

Since we study the problems using competitive analysis, we are not only interested in the maximum value m_t of the window at time t, but also in the maximum value $\hat{m}_t$ which is stored in the memory of an offline algorithm; the offline algorithm has the same memory restrictions with the online algorithm, but it knows the future. In the definition of the competitive ratio, we should use $\hat{m}_t$. However, we study both ratios (against m_t and $\hat{m}_t$) and we show that the competitive ratio is essentially the same. We use m_t to denote both values.

In the literature of streaming algorithms, a streaming algorithm is always compared against the absolute optimum (such as m_t). It may be useful for other problems to adopt the competitive point of view and judge an algorithm against the value (such as $\hat{m}_t$) of an offline algorithm with the same limitations on its resources.

Memory space. We are interested in maintaining the items themselves, which can in general be complex data structures, not just their values. As a result, the required memory space is measured in units, each unit being the amount of memory necessary to exactly store an item. We assume that the algorithm maintains, at any time t, at most k items among those observed in $\{t-n+1, \ldots, t\}$ (where, typically, $k << n$). Table 1 summarizes the notation we use.

Objective functions. While approximating the maximum value over a sliding window can be done using polylogarithmic space [7], the basic task of maintaining the maximum value exactly is not feasible, unless one stores a number of elements in the order of n and this result also holds for randomized algorithms [12]. We consider the following objective functions, that measure how far we are from achieving this goal, both at every point in time and in the average over the entire sequence.

Aggregate max: Maximize $\sum_t g_t$, i.e., the average value of the largest item maintained by the algorithm.

Anytime max: For every t, maximize g_t, i.e., maximize the value of the largest item in the algorithm's memory at time t. As shown further, this function is harder to maintain for every t.

Competitive analysis. We compare the performance of the algorithm against the optimal algorithm that knows the entire sequence in advance [6]. Hence, in this case we define the competitive ratio $\mathbf{r}(k,n)$ as:

$$\mathbf{r}(k,n) = \max_{a \in \mathcal{S}} \frac{\sum_t g_t(a)}{\sum_t \hat{m}_t(a)},$$

where $\mathcal{S}$ is the set of possible input sequences and $g_t(a)$ (respectively, $\hat{m}_t(a)$) is the online algorithm's (respectively the offline algorithm's) maximum value at time t when input sequence a is observed. We also study the maximum value $m_t(a) = \max\{a_{t-n+1}, \ldots, a_t\}$ of the last n values instead of $\hat{m}_t$. In the sequel we drop the name of the sequence a to simplify the notation.

We note that our definition of the competitive ratio does not have an additive constant [6]. It is a simple observation that such an additive constant cannot play any role in this type of problems: the reason is that we can repeat a sequence many times (probably by appropriate scaling) to make the effects of an additive constant insignificant.

3 Competitive analysis of the online aggregate max problem

In this section we study the online aggregate max problem. We first prove a $1+\Omega(\frac{1}{k})$ lower bound on the competitive ratio of any randomized on line algorithm and then we prove an asymptotically tight deterministic upper bound for the partition-greedy algorithm.

3.1 Randomized lower bound

Theorem 1. *Every randomized online algorithm for the aggregate max problem with memory k has competitive ratio $1+\Omega(1/k)$.*

Proof. The proof is based on Yao's Lemma [6]: We fix a probability distribution on inputs and compute the ratio of the gain of the optimal online algorithm (which knows the distribution but not the outcome of the random experiment) against the optimal algorithm (which knows the exact input).

We select a simple probability distribution of inputs: The input consists of two parts: the first part of n items has values $f(t)$, $t=0,\dots,n-1$, where f is a decreasing function (to be determined later); the second part of the input consists of x items of value 0, where x is random value uniformly distributed in $1,\dots,n$. Thus the online algorithm knows everything, except of when the input sequence stops. For the above sequence, we compute the gain of the optimal online algorithm–which is the hard part—and we lower bound the gain of the optimal offline algorithm to get the desired ratio.

We can assume that the above sequence repeats arbitrarily many times (with independent values x each time) so that we can ignore additive constants in the definition of the competitive ratio.

Essentially, the online algorithm has only to decide which items to keep in memory from the first part of the sequence. To simplify the situation, we assume that the online gain during the first n steps is exactly $n\cdot f(0)$; this is an overestimate when the online algorithm drops the value $f(0)$ at some point, but we are fine since we care to bound its cost from above. With this assumption, an online algorithm is determined completely by the values $f(t_1)$,. . . ,$f(t_k)$ which has in its memory at the end of the first part. To compute the expected online gain we define $t_0=0$, $t_{k+1}=n$, and

$$h(t)=\begin{cases} f(t_1) & t_0\le t\le t_1\\ \vdots & \\ f(t_{k+1}) & t_k\le t\le t_{k+1}.\end{cases}$$

To simplify the presentation, we treat f and h as real functions. Also, by scaling appropriately the time (by a factor $1/n$) and the values (by a factor $1/f(0)$), we assume that $n=1$, $f(0)=1$ and that the values t_i are real numbers in $[0,1]$ with $t_0=0$ and $t_{k+1}=1$ (see Figure 1). The effects of these assumptions can be safely ignored for large n and k. The function h approximates from below the function f in $k+2$ points; the situation resembles the Gaussian-Legendre quadrature for approximating integrals, but with a different objective.

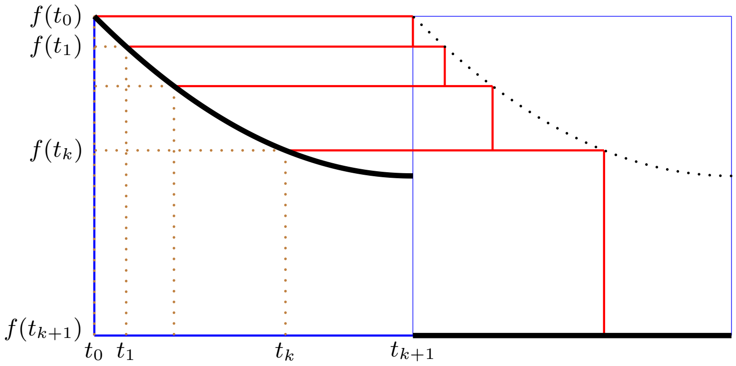

Fig. 1. The lower bound construction. The stream values follow the bold descending line. The descending staircase line shows the maximum value in the memory of the online algorithm. The area below the staircase is the online gain until some random point of the right part.

When the input sequence ends at time $1+x$, the online gain is $1+\int_0^x h(t)\,dt$. Therefore the expected online gain is given by

$$E[g] = 1 + \int_0^1 h(t)(1-t)\,dt = 1 + \int_0^1 h(t)d(1-(1-t)^2) = 1 + \int_0^1 h(1-\sqrt{1-r})dr.$$

The change in the variable suggests to consider $r_i = 1-(1-t_i)^2$ (and consequently $t_i = 1-\sqrt{1-r_i}$). Notice that $r_0 = 0$ and $r_{k+1} = 1$. The expected online gain then is

$$E[g] = 1 + \sum_{i=0}^{k} \int_{r_i}^{r_{i+1}} f(t_{t+1})dr = 1 + \sum_{i=0}^{k} (r_{i+1}-r_i) f(t_{i+1}).$$

We now want to select an appropriate f which simplifies the computations. In fact, it is very possible that many choices of the function f result in a similar lower bound—or even with a better coefficient of $1/k$ in the competitive ratio—and in particular linear functions such as $f(t) = 1-t/2$, but computing the optimal online gain appears to be very complicated. The difficulty lies in finding the optimum values for t_i. We however choose

$$f(t) = \frac{1}{2} + \frac{1}{2}(1-t)^2.$$

With this choice, we get that $f_i = 1 - r_i/2$, and the expected online cost is

$$E[g] = 1 + \sum_{i=0}^{k}(r_{i+1}-r_i)(1-\frac{r_{i+1}}{2}) = 1 + \frac{1}{2}\left(\frac{3}{4} - \frac{1}{4}\sum_{i=0}^{k}(r_{i+1}-r_i)^2\right) = \frac{11}{8} - \frac{1}{8}\sum_{i=0}^{k}(r_{i+1}-r_i)^2.$$

It is now very easy to select $t_i's$ (or equivalently r_i's) to maximize the above expression: Since $\sum_{i=0}^{k}(r_{i+1} - r_i) = r_{k+1} - r_0 = 1$, the optimal online algorithm for this particular f should select $r_{i+1} - r_i = 1/(k+1)$, or equivalently $r_i = i/(k+1)$. With this choice, the expected online cost is

$$E[g] = \frac{11}{8} - \frac{1}{8}\sum_{i=0}^{k}\frac{1}{(k+1)^2} = \frac{11}{8} - \frac{1}{8(k+1)}.$$

We now turn our attention to the offline algorithm. The advantage of this algorithm over the online algorithm is that it knows x; therefore it keeps in memory only items in $[0, x]$ (as compared to the online algorithm which keeps in memory items in $[0, 1]$ but the items after x are useless). We do not need to compute the optimal offline algorithm, it suffices to compute some lower bound[3]. We consider the offline (suboptimal) algorithm which keeps in memory the equidistant values $t'_i = \frac{i}{k-1}\cdot x$. For a given x, the gain of this algorithm is

$$1 + \sum_{i=1}^{k-1}(t'_i - t'_{i-1})f(t'_i) = 1 + \sum_{i=1}^{k-1}\frac{x}{k-1}\left(\frac{1}{2} + \frac{1}{2}\left(1 - \frac{i}{k-1}\cdot x\right)^2\right).$$

For uniformly distributed x in $[0, 1]$, we get that the expected offline gain is

$$1 + \sum_{i=1}^{k-1}\int_0^1 \frac{x}{k-1}\left(\frac{1}{2} + \frac{1}{2}\left(1 - \frac{i}{k-1}\cdot x\right)^2\right)dx = \frac{11}{8} - \frac{1}{48}\frac{5k-6}{(k-1)^2}.$$

Dividing it by the expected online cost $\frac{11}{8} - \frac{1}{8(k+1)}$, we get the ratio

$$1 + \frac{1}{66k} + O(\frac{1}{k^2}),$$

which proves the theorem.

[3] It is not hard to compute that even with arbitrarily large memory the expected optimal gain is $1 + \int_0^1 f(t)(1-t)dt = 11/8$ which also gives the same lower bound, although with an improved coefficient of $1/k$ in the competitive ratio.

```
PartitionGreedy(n, k)
t = 1
while Stream not finished {a is current item} do
  for j: 0...k - 1 do
    B[j].time = B[j].time + 1 {Window shifts 1 slot to the right}
  end for
  i = ⌈tk/n⌉ {Compute part to which t belongs}
  if B[i].time > n OR B[i].val ≤ a.val then
    B[i] = a
  end if
   t = t + 1
end while
```

Fig. 2. Partition greedy algorithm. For an item `a`, `a.val` and `a.time` repsectively denote the value of `a` and the time units elapsed since `a` was observed.

3.2 Deterministic upper bound

In this subsection we give (almost) tight bounds on the competitive ratio for large k. We will prove the result in the strongest possible sense: We will show that the deterministic competitive ratio is $\mathbf{r}(k,n) = 1+O(1/k)$.

We analyze a very natural algorithm which we call partition-greedy: We partition the sequence into parts of size n/k. Part s starts at time $(s-1)n/k+1$ and ends at time sn/k. For every part s of the sequence, a particular slot of memory is active, the memory slot $i = 1 + (s \pmod k)$. For each part of the sequence, the active slot of the memory accepts the first item. In every other respect, the active slot in each part is updated greedily: the algorithm updates the slot value whenever an item of larger (or the same) value appears. Clearly, the partition-greedy algorithm can be implemented very efficiently (using two counters to keep track of the current active slot of memory and the number of items seen in each part).

Theorem 2. *The partition-greedy algorithm has competitive ratio $k/(k-1)$.*

Proof. Let m_t denote the maximum value in $\{a_{t-n+1},\dots,a_t$ and let g_t denote the maximum value in the online algorithm memory at time t. Let also P_t denote the values in the part of size n/k in which t belongs. If the value m_t appeared in the last $k-1$ parts (i.e., in parts $P_{t-(k-1)n/k},\dots,P_t$), it must be still in the online memory. Therefore, when $g_t < m_t$, it must be the case that the value m_t appeared in part P_{t-n} and it was dropped in the current part P_t. Roughly speaking, in a decreasing sequence the online algorithm maintains the maximum value in memory for $k-1$ parts, which gives the competitive ratio. However, we need to be more careful as the sequence of values may not be decreasing. To do this, we "charge" $k-1$ online values $g_t, g_{t-n/k},\dots,g_{t-n(k-1)/k}$ to the optimal value m_t, as follows: We first observe that the above implies

$$\begin{array}{l}\text{either } g_t = m_t,\\ \quad\text{or } g_{t-n/k} \geq m_t \text{ and } \cdots \text{ and } g_{t-n(k-1)/k} \geq m_t\end{array}.$$

To simplify the presentation, we may use negative indices and we assume that both m_t and g_t are 0 for negative t. Since the above condition compares only values that differ by multiples of n/k (which we can assume to be an integer), we will use the notation $g'_t = g_{\lfloor tk/n \rfloor + t \pmod{n/k}}$, so that the above condition becomes

$$\begin{array}{l}\text{either } g'_t = m'_t,\\ \quad\text{or } g'_{t-1} \geq m'_t \text{ and } \cdots \text{ and } g'_{t-(k-1)} \geq m'_t\end{array}.$$

To prove the theorem, it suffices to show that $k\sum_t g'_t \geq (k-1)\sum_t m'_t$, and this is what we will do.

We first show the following claim: For every t and every $r = 0,\dots,k-1$, there is an index $\ell(t,r) \in \{t-r,\dots,t\}$ such that

$$-g'_{\ell(t,r)} + \sum_{i=t-r}^{t} g'_{t-i} \geq \sum_{i=t-r+1}^{t} m'_{t-i}. \tag{1}$$

In words, for r consecutive optimal values there are equal values in $r+1$ consecutive online values (and $\ell(t,r)$ denotes the unmatched online value). The proof of the claim is by induction on r: For $r=0$ the claim is trivial by taking $\ell(t,r)=t$. For the induction step,

- either $g'_t \geq m'_t$, in which case we take $\ell(t,r) = \ell(t-1,r-1)$; i.e., we match g'_t to m'_t and leave the same element $\ell(t-1,m-1)$ unmatched.

– or $g'_t < m'_t$, in which case we take $\ell(t,r) = t$; i.e., we match g'_t to the unmatched element $\ell(t-1,r-1)$ and leave t unmatched. For this, notice that $\ell(t-1,r-1) \geq t-(k-1)$ and therefore $g'_{\ell(t-1,r-1)} = m'_t$.

From (1), by dropping the term involving the missing element, we get

$$\sum_{i=t-r}^{t} g'_{t-i} \geq \sum_{i=t-r+1}^{t} m'_{t-i}, \tag{2}$$

for every $r = 0, \ldots, k-1$.

By summing up the above inequalities for every t and for $r = k-1$, we get that the left side is approximately $k\sum_{t=0}^{T} g'_t$ and the right hand side is approximately $(k-1)\sum_{t=1}^{T} m_t$, which would immediately prove the upper bound. There is a slight complication, in that the last values in the above sums appear fewer than k and $k-1$ times respectively; for example, g'_T and m'_T appears only once. To take care of this, we also add the inequalities (2) for $t = T$ and for every $r = 0, \ldots, k-2$. In detail, we sum up the following inequalities (2)

$$\sum_{t=0}^{T} \sum_{i=t-(k-1)}^{t} g'_{t-i} + \sum_{r=0}^{k-2} \sum_{i=T-r}^{t} g'_{T-i} \leq \sum_{t=0}^{T} \sum_{i=t-(k-2)}^{t} m'_{t-i} + \sum_{r=0}^{k-2} \sum_{i=T-r+1}^{t} m'_{T-i},$$

which simplifies to the desired inequality

$$k\sum_{t=0}^{T} g'_t \geq (k-1)\sum_{t=0}^{T} m'_t.$$

It is easy to give an example where the partition-greedy has competitive ratio $k/(k-1)$: The sequence has length $n(k+1)/k$ (equal to $k+1$ parts) and it consists entirely of 0's, except of the value at end of the first part which has value 1, i.e. $a_{n/k-1} = 1$. The online algorithm retains this value in memory for $k-1$ parts, while he optimal algorithm retains it for k parts.

4 Generalizations and other variants of the problem

In this section we investigate generalizations and variants of the problem.

4.1 Items with different expiration times

A natural generalization of the aggregate max problem is when each item has its own expiration time. In this generalization the input is a sequence $(a_1, n_1), \ldots, (a_n, n_n)$, where n_i is the time after which a_i expires. The online algorithm must decide for each item whether to keep it in memory or not. Again the restriction is that only k items can be kept in memory at any time and that an expired item has no value (or equivalently, that item i cannot be kept in memory for more than n_i steps). In the previous sections we have considered the special case $n_i = n$. We will show that no online algorithm can have bounded competitive ratio for this problem.

Theorem 3. *The deterministic aggregate max problem with variable expiration times has unbounded competitive ratio.*

Proof. The adversary gives $k+1$ different types of items: items of type i, $i = 1, \ldots, k+1$, have value v_i and duration T_i; the two sequences $v_1, v_2, \ldots, v_{k+1}$ and $v_{k+1}T_{k+1}, v_kT_k, \ldots, v_1T_1$ are steeply increasing. The items of every type are repeated periodically so that most of the time there is exactly one alive (not expired) item of each type. However, when the online algorithm rejects some item of type $j \in \{1, \ldots, k\}$ to accept an item of type $k+1$, the adversary stops giving more items of types $j+1, \ldots, k+1$ until the item of type j expires; the adversary resumes giving all types of tasks once the item of type j expires.

The online algorithm faces the following dilemma: To optimize the long-term gain, it should keep the items of the first k types in memory and reject the items of type $k+1$; this is so because the values of the items are such that v_iT_i is a steeply decreasing sequence. On the other hand, to optimize the short-term gain, the items of type $k+1$ must be kept because their value is much higher than the value of the rest. We show that there is no good way to resolve this dilemma. To do this, we show by induction on i that an online algorithm with bounded competitive ratio cannot reject an item of type $i \leq k$. But then, the competitive ratio against an offline algorithm which keeps the items of type $k+1$ is at least $v_{k+1}/(v_1+\cdots+v_k)$; this ratio can be also unbounded.

We first show only the basis of the induction for $i=1$, i.e., that an online algorithm with bounded competitive ratio has to keep the items of type $i=1$ in memory. Suppose that at some time t the online algorithm accepts the item of type $k+1$ by rejecting an item of type 1. Then no new items arrive until the expiration of the item of type 1. The online gain is at most $t(v_1+\ldots+v_k)+(v_2T_2+\cdots+v_{k+1}T_{k+1})$, while the optimal gain is at least $\max(v_1T_1, tv_{k+1}) \geq (v_1T_1+tv_{k+1})/2$. If we select values with $v_1T_1 \geq \lambda(v_2T_2+\cdots+v_{k+1}T_{k+1})$ and $v_{k+1} \geq \lambda(v_1+\ldots+v_k)$ for some positive parameter λ, then the competitive ratio is at least λ, which can be arbitrarily high.

For the induction step, we focus only on online and offline algorithms that always keep all items of type $l<i$ in memory. Since the values $v_1,\ldots,v_{i-1}$ are much smaller than the values of higher-type items, they have small effect on the competitive ratio and we can essentially repeat the above argument of the basis case.

4.2 The aggregate min problem

We show that when we change our objective from keeping the maximum value to keeping the minimum value, the problem is transformed completely. In particular, we show that the competitive ratio is unbounded for the aggregate min case.

Proposition 1. *The aggregate min problem has unbounded competitive ratio.*

Proof. Fix some online algorithm. The adversary gives a steeply increasing sequence $a_1, a_2, \ldots$. Let a_t (for some $t \leq k+1$) be the first item that is not kept by the online algorithm. The adversary ends the sequence at time $t+n-1$. The main point is that at time $t+n-1$ the online algorithm has in its memory only items that appeared after time $t+1$, whereas the offline algorithm can have the item a_t which has much smaller value. More precisely, the total online cost is at least $na_1+a_2+\ldots+a_{t-1}+a_{t+1}$, while the cost of an offline algorithm who keeps a_t (by replacing a_{t-1}, if it is needed) is $na_1+a_2+\ldots+a_{t-2}+2a_t$[4]. By selecting a_{t+1} to be much higher than $na_1+a_2+\ldots+a_{t-2}+2a_t$, we get an unbounded competitive ratio.

4.3 The anytime max ratio

In this section we address the question of approximating at any time t the maximum value of the current window. In the aggregate max problem, the objective is to optimize the *sum* of the maximum value whereas here it is to optimize the *maximum*. That is, we are interested in minimizing

$$\max_a \max_t \frac{g_t}{m_t}.$$

We show that the competitive ratio in this case cannot be independent of the values. More precisely, we show that the competitive ratio is $O(\sqrt[k+1]{M})$, where M is the maximum value in the stream. A similar result has been shown also in [10] where they give an $1+\epsilon$-approximation algorithm with memory $\frac{1}{\epsilon}\log M$. Although the results are essentially the same, this seems to be a cleaner (and simpler) problem when we cast it in the framework of competitive analysis.

Proposition 2. *The competitive ratio* $\max_a \max_t \frac{g_t}{m_t}$ *is*

$$O(\sqrt[k+1]{M}),$$

where M is the maximum value in the stream.

Proof. To show the lower bound, consider a geometrically decreasing sequence with $a_j = M\alpha^{j-1}$, where $\alpha = M^{1/n}$. Let $t+1$ be the first time that the online algorithm does not accept an item. In all previous times, the online algorithm accepts the current item by possibly replacing previously kept items. From then on, the adversary gives items with value 1. This means in particular that at time $t+n$ the online algorithm has only items with value 1 in its memory, while the offline algorithm could have kept $a_{t+1} = M/\alpha^t$. This gives a lower bound M/α^t for the online algorithm.

To obtain a different lower bound, observe that at time $t+1$ there are at most k items in the online memory. This means that there is a sequence of t/k consecutive items $a_i,\ldots,a_{i+t/k-1}$ none of which is in

[4] This expressions hold for $t>1$. The case $t=1$ is similar.

the online memory. Then at time $i+n-1$, the maximum item in the online memory is the item $a_{i+t/k}$, while an offline algorithm could have the item a_i. This gives a lower bound on the competitive ratio $\alpha^{t/k}$.

In summary, the competitive ratio is at least

$$\max\{M/\alpha^t, \alpha^{t/k}\},$$

which is at least $M^{1/(k+1)}$ when $\alpha = M^{1/n}$.

This result is tight: we show that a simple, deterministic, bucket-based algorithm can achieve the same approximation ratio. The algorithm arranges the interval $[1,\ldots,M]$ of possible values into k *buckets* as follows: for $i=1,\ldots,k$, bucket i contains all values in the interval $[(\sqrt[k]{M})^{i-1}, (\sqrt[k]{M})^{i})$. The algorithm maintains a list of *representatives*, one for every bucket. Clearly, the competitive ratio cannot be more than $M^{1/k}$. When the maximum value M is not known in advance, the online algorithm starts by assuming $M=k$; whenever a value higher than the current M appears, the online algorithm accepts it (in its top bucket) and updates M. The competitive ratio is not affected.

5 Conclusions and open problems

In this paper we considered windowed streaming problems from a competitive analysis perspective. We proved (positive and negative) results for several problems, revolving around or extending the central problem of maintaining the maximum value observed over the last n observations, while obeying stringent memory constraints, consistent with realistic streaming applications.

Many interesting open questions remain. First, there is the problem of tightening the results of Theorems 1 and 2, especially for the case of constant k and in particular for $k=1$. Also, an interesting direction is to study other statistical questions using the competitive framework we considered in this work. One such question, which is directly relevant to this work, is to maintain some aggregate of the r largest values observed in the window, for some $r \leq k$. Another direction is to consider the order statistics of the items, rather than their actual value. This is useful in settings where we are interested in maintaining a subset of elements that are some of the largest ones in the current window. In this case, the quality of the solution depends on how far the ranks of the top items maintained by the algorithm are from the actual top items in each window of interest. Performance indices for this kind of questions are known in information retrieval and data mining. Analyzing them in a competitive scenario is a challenging task.

Finally an interesting framework to study this type of problems is to assume that the items appearing in the stream are chosen by an adversary, but then presented to the algorithm in a random permutation. This assumption is common in a related, important problem in decision theory, namely the *secretary problem* [17]. Variants of the secretary problem are to maximize the probability to select the top r items [20] or to minimize the expected sum of the ranks of the selected items [1]. The question is how to address these problems in a sliding window setting.

References

1. Miklos Ajtai, Nimrod Megiddo, and Orli Waarts. Improved algorithms and analysis for secretary problems and generalizations. *SIAM Journal on Discrete Mathematics*, 14(1):1–27, 2000.
2. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proc. of the ACM Symposium on the Theory of Computing*, pages 20–29, 1996.
3. Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '04)*, pages 286–296, New York, NY, USA, 2004. ACM.
4. Brain Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proc. of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '03)*, pages 234–243, New York, NY, USA, 2003. ACM.
5. Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
6. Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.
7. Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 283–293, 2007.
8. Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Succinct sampling on streams. *Computing Research Repository (CoRR)*, abs/cs/0702151, 2007.
9. Andrei Z. Broder, Adam Kirsch, Ravi Kumar, Michael Mitzenmacher, Eli Upfal, and Sergei Vassilvitskii. The hiring problem and lake wobegon strategies. In *Proceedings of the nineteenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA '08)*, pages 1184–1193, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

10. Timothy M. Chan and Bashir S. Sadjad. Geometric optimization problems over sliding windows. In *Proc. of the 15th International Symposium on Algorithms and Computation (ISAAC)*, Lecture Notes in Computer Science, pages 246–258. Springer, 2004.
11. Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005.
12. Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal of Computing*, 31(6):1794–1813, 2002.
13. Mayur Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Proc. of the 10th Annual European Symposium on Algorithms (ESA '02)*, pages 323–334. Springer-Verlag, 2002.
14. R. El-yaniv, A. Fiat, R. M. Karp, and G. Turpin. Optimal search and one-way trading online algorithms. *Algorithmica*, 30:101–139, 2001.
15. Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '03)*, pages 28–36, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.
16. Joan Feigenbaum, Sampath Kannan, and Jian Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2004.
17. T. S. Ferguson. Who solved the secretary problem? *Statistical Science*, 3(4):282–296, 1988.
18. Lukasz Golab, David DeHaan, Erik D. Demaine, Alejandro Lopez-Ortiz, and J. Ian Munro. Identifying frequent items in sliding windows over on-line packet streams. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet Measurement (IMC '03)*, pages 173–178, New York, NY, USA, 2003. ACM.
19. Sudipto Guha and Andrew McGregor. Approximate quantiles and the order of the stream. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '06)*, pages 273–279, New York, NY, USA, 2006. ACM.
20. Robert Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '05)*, pages 630–631, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
21. L. K. Lee and H. F. Ting. 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 (PODS '06)*, pages 290–297, New York, NY, USA, 2006. ACM.
22. Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston. Finding (recently) frequent items in distributed data streams. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*, pages 767–778. IEEE Computer Society, 2005.
23. S. Muthukrishnan. *Data streams: algorithms and applications*. Now Publishers Inc., 2005.