Competitive analysis of aggregate max in windowed streaming

Elias Koutsoupias
University of Athens

Luca Becchetti
University of Rome

July 9, 2009

The streaming model

Streaming

- A stream is a sequence of items (numbers) $a_1, a_2, \ldots$
- Much longer than the algorithm's memory

$\ldots, 43, 4, 67, 2, 44, 10, 89, 34, 22, 67, 15, 67, 88, 91, 33, 7, 18, 14, 92 \ldots$

Windowed streaming

- A stream, but we care only about the values of a **sliding window** of length n.
- n is much longer than the algorithm's memory

$\ldots, 43, 41, 67, 23, 44, 10, 89, 34, 22, 67, 15, 67, 88, 91, 33, 7, 18, 14, \ldots$

The streaming model

Streaming

- A stream is a sequence of items (numbers) $a_1, a_2, \ldots$
- Much longer than the algorithm's memory

$\ldots, 43, 4, 67, 2, 44, 10, 89, 34, 22, 67, 15, 67, 88, 91, 33, 7, 18, 14, 92 \ldots$

Windowed streaming

- A stream, but we care only about the values of a **sliding window** of length n.
- n is much longer than the algorithm's memory

$\ldots, 43, 41, 67, 23, 44, 10, 89, 34, 22, 67, 15, 67, 88, 91, 33, 7, 18, 14, \ldots$

The streaming model

Streaming

- A stream is a sequence of items (numbers) $a_1, a_2, \ldots$
- Much longer than the algorithm's memory

$\ldots, 43, 4, 67, 2, 44, 10, 89, 34, 22, 67, 15, 67, 88, 91, 33, 7, 18, 14, 92 \ldots$

Windowed streaming

- A stream, but we care only about the values of a **sliding window** of length n.
- n is much longer than the algorithm's memory

$\ldots, 43, 41, \color{red}{67, 23, 44, 10, 89, 34, 22, 67, 15, 67, 88}, 91, 33, 7, 18, 14, \ldots$

The streaming model

Streaming

- A stream is a sequence of items (numbers) $a_1, a_2, \ldots$
- Much longer than the algorithm's memory

$\ldots, 43, 4, 67, 2, 44, 10, 89, 34, 22, 67, 15, 67, 88, 91, 33, 7, 18, 14, 92 \ldots$

Windowed streaming

- A stream, but we care only about the values of a **sliding window** of length n.
- n is much longer than the algorithm's memory

$\ldots, 43, 41, 67, 23, 44, 10, 89, 34, 22, 67, 15, 67, 88, 91, 33, 7, 18, 14, \ldots$

The streaming model

Streaming

- A stream is a sequence of items (numbers) $a_1, a_2, \ldots$
- Much longer than the algorithm's memory

$\ldots, 43, 4, 67, 2, 44, 10, 89, 34, 22, 67, 15, 67, 88, 91, 33, 7, 18, 14, 92 \ldots$

Windowed streaming

- A stream, but we care only about the values of a **sliding window** of length n.
- n is much longer than the algorithm's memory

$\ldots, 43, 41, 67, 23, \color{red}{44, 10, 89, 34, 22, 67, 15, 67, 88, 91, 33}, 7, 18, 14, \ldots$

Statistical questions

Questions about streams

- What is the maximum value so far; the average? the number of distinct values?
- Some of them are easy in the streaming model
- Impossible in the windowed streaming model

Statistical questions

Questions about streams

- What is the maximum value so far; the average? the number of distinct values?
- Some of them are easy in the streaming model
- Impossible in the windowed streaming model

Why windowed streaming?

Typical information on a temperature sensor

- The maximum temperature in the last hour

Our setting

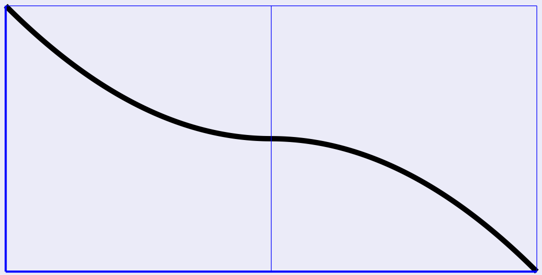
Assumptions

- We care about the maximum value in each window
- We use competitive analysis (worst-case input)
- The online algorithm has limited memory; of size k ($k \leq n$)

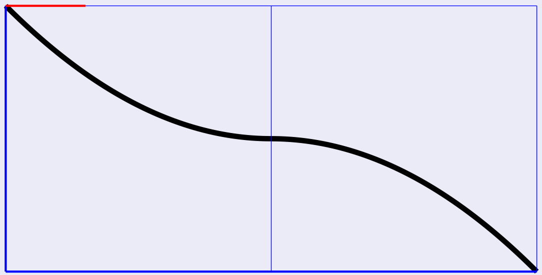
Objective

- g_t the maximum value in online algorithm's memory at time t
- m_t the value in the memory of an optimal offline algorithm
- $$\text{Minimize } \rho(k) = \frac{\sum_t m_t}{\sum_t g_t}$$

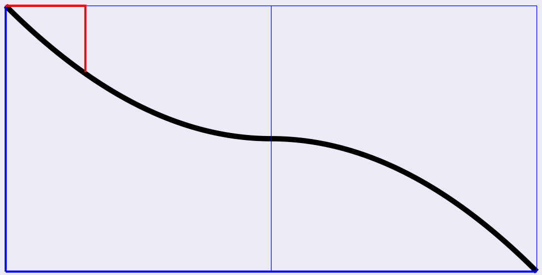
Typical online situation for $k = 1$



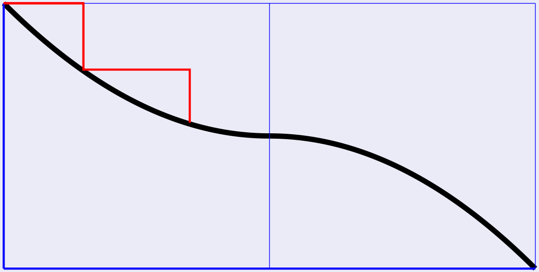
Typical online situation for $k = 1$



Typical online situation for $k = 1$

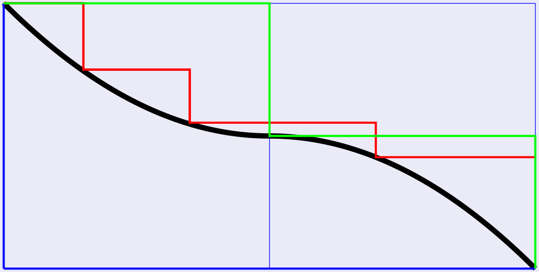


Typical online situation for $k = 1$

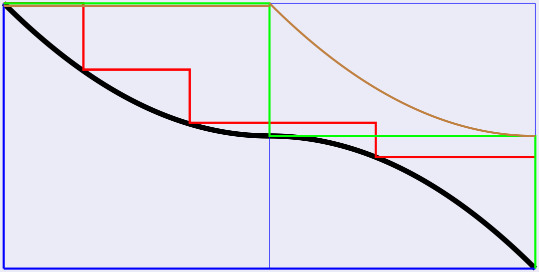


Typical online situation for $k = 1$

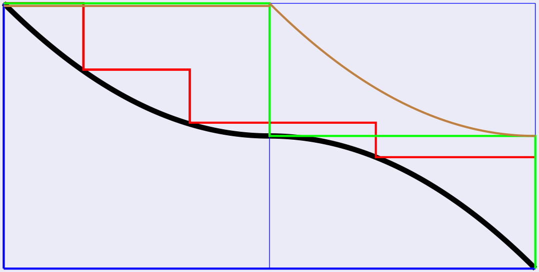
Typical online situation for $k = 1$



Typical online situation for $k = 1$



Typical online situation for $k = 1$



$$\text{Online gain} = \text{area below the red line}$$
$$\text{Offline gain} = \text{area below the green line}$$
$$\text{Optimal max} = \text{area below the brown line}$$
$$\text{Competitive ratio} = \frac{\text{Offline gain}}{\text{Online gain}}$$

Comparison to streaming

Streaming	**Competitive analysis**
Fix approx ratio and optimize the memory size	Fix memory size and optimize the approximation ratio
Values within the approx ratio always	Values within the approx ratio on average
worst-case input	worst-case input
compare to max in the window	compare to max achieved by an offline algorithm with the same memory constraints

Streaming results

- Datar, Gionis, Indyk, and Motwani introduced window streaming. They showed that no exact algorithm can do better than keeping all the items
- Feigenbaum, Kannan, Zhang considered the problem of estimating the diameter of 2-dim points
- Chan and Sadjad improved their results. They showed that for maintaining the diameter of points on a line can be done with memory

  $$O(\frac{1}{\epsilon} \log M)$$

  memory slots, where M is the diameter.

Main result

Theorem

For fixed memory k, the competitive ratio is

$$\rho(k) = 1 + \Theta\left(\frac{1}{k}\right)$$

The theorem holds in the strongest possible sense

- **Upper bound:** There exists a **deterministic** algorithm which achieves a competitive ratio against the optimal with memory n.
- **Lower bound:** **Randomized** against the optimal with memory k.

Upper bound

The Partition-Greedy algorithm

- We partition the sequence into parts of size n/k and we associate with the memory slot i the parts $i \pmod{k}$.
- For each part, the associated slot accepts the first item.
- In every other respect, the slots are updated greedily: the algorithm updates the slot value whenever a greater value appears.

Theorem

The Partition-Greedy *algorithm has competitive ratio* $k/(k-1)$, *against the absolute maximum.*

Sketch of the upper bound

- m_t: maximum of the last n values at time t
- g_t: the maximum value in online memory at time t

$$\begin{aligned} \text{either} \quad & g_t = m_t, \\ \text{or} \quad & g_{t-w/k} \geq m_t \text{ and } \cdots \text{ and } g_{t-w(k-1)/k} \geq m_t, \end{aligned}$$

- Why? If the value m_t appeared in the last $k-1$ parts of the window, it must be still in the online memory.

$$\begin{array}{rl} \text{either} & g_t = m_t, \\ \text{or} & g_{t-w/k} \geq m_t \text{ and } \cdots \text{ and } g_{t-w(k-1)/k} \geq m_t, \end{array}$$

implies

$$\sum_{i=t-r}^{t} g_{t-i*w/k} \geq \sum_{i=t-r+1}^{t} m_{t-i*w/k},$$

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

Summing up these inequalities for every t and for $r = k-1$ we (almost) get

$$k \sum_{t=0}^{T} g_t \geq (k-1) \sum_{t=0}^{T} m_t.$$

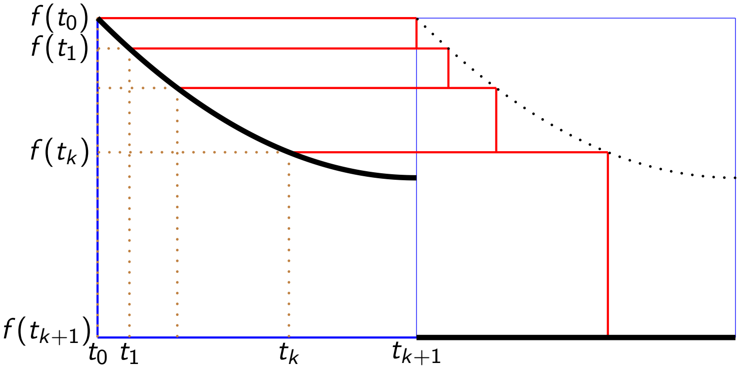
We need to take care of some minor problems near the end.

Lower bound

Main idea

- The proof is based on Yao's Lemma
- The input consists of two parts:
  - the first part of n items has values $f(t)$, $t = 0, \ldots, n-1$ of some function f
  - the second part of the input consists of x items of value 0, where x is random value uniformly distributed in $1, \ldots, n$.
  - the online algorithm knows everything, except of when the input sequence stops.

The lower bound construction



Lower bound

There is a function f such that when the sequence stops at a random point uniformly distributed in the second part

- the expected gain of every online algorithm is at most

$$\frac{11}{8} - \frac{1}{8(k+1)}.$$

- the offline gain is at least

$$\frac{11}{8} - \frac{1}{48}\frac{5k-6}{(k-1)^2}.$$

- Therefore the competitive ratio at least

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

Computing the online gain

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

The expected online gain is given by

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

Define $r_i = 1-(1-t_i)^2$.
The expected online gain then is

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

Which f?

- We need to select f so that
  - We can find the optimal deterministic algorithm
  - We can upper bound its expected gain
  - We can lower bound the expected optimal gain

A convenient choice

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

With this the expected online gain is

$$\begin{aligned} 1 + \sum_{i=0}^{k}(r_{i+1}-r_i)(1-\frac{r_{i+1}}{2}) &= 1 + \frac{1}{2}\left(\frac{3}{4} - \frac{1}{4}\sum_{i=0}^{k}(r_{i+1}-r_i)^2\right) \\ &= \frac{11}{8} - \frac{1}{8}\sum_{i=0}^{k}(r_{i+1}-r_i)^2. \end{aligned}$$

Selecting the optimal online algorithm

We select t_i (or r_i) which maximize

$$\frac{11}{8} - \frac{1}{8}\sum_{i=0}^{k}(r_{i+1} - r_i)^2.$$

$$r_{i+1} - r_i = \frac{1}{k+1} \implies r_i = \frac{i}{k+1}$$

The expected online gain is

$$\frac{11}{8} - \frac{1}{8}\sum_{i=0}^{k}\frac{1}{(k+1)^2} = \frac{11}{8} - \frac{1}{8(k+1)}$$

Computing the expected optimal cost

Offline algorithm

- The advantage of the offline algorithm: it knows x.
- It keeps items only from $[0, x]$.
- The online algorithm keeps values from $[0, 1]$; values after x are useless.
- We need only an upper bound; we select a convenient suboptimal algorithm.
- It 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

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

The lower bound

In summary

- Online gain at most

$$\frac{11}{8} - \frac{1}{8(k+1)}.$$

- Offline gain at least

$$\frac{11}{8} - \frac{1}{48}\frac{5k-6}{(k-1)^2}.$$

- Competitive ratio at least

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

Items with expiration times

Generalization of the problem

- Each item has its own expiration time. We want to have the maximum of the non-expired items.
- In the window streaming all items expire after n steps.

Theorem

The deterministic competitive ratio is unbounded.

The aggregate min problem

Theorem

The aggregate min problem has unbounded competitive ratio

The anytime max problem

Anytime max

In the anytime max problem we want the online algorithm to have an almost maximum item in memory **at all time steps**.

- Aggregate → minimize

$$\frac{\sum m_t}{\sum g_t}$$

- Anytime → minimize

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

Theorem

The aggregate min problem has unbounded competitive ratio.

Comments and open problems

- The case of $k = 1$ is of particular importance.
- Similar problem: When we care about the ranks (rank=position in the ordered list) and not the values.
- Interesting connection with the secretary problem: The input is adversarial random-order (as opposed to worst-order).