

Chapter 8

THE SLIDING-WINDOW COMPUTATION MODEL AND RESULTS*

Mayur Datar

Google, Inc.

datar@cs.stanford.edu

Rajeev Motwani

Department of Computer Science

Stanford University

rajeev@cs.stanford.edu

Abstract The sliding-window model of computation is motivated by the assumption that, in certain data-stream processing applications, recent data is more useful and pertinent than older data. In such cases, we would like to answer questions about the data only over the last N most recent data elements (N is a parameter). We formalize this model of computation and answer questions about how much space and computation time is required to solve certain problems under the sliding-window model.

Keywords: sliding-window, exponential histograms, space lower bounds

Sliding-Window Model: Motivation

In this chapter we present some results related to small space computation over sliding windows in the data-stream model. Most research in the data-stream model (e.g., see [1, 10, 15, 11, 13, 14, 19]), including results presented in some of the other chapters, assume that all data elements seen so far in the stream are equally important and synopses, statistics or models that are built should reflect the entire data set. However, for many applications this

*Material in this chapter also appears in **Data Stream Management: Processing High-Speed Data Streams**, edited by *Minos Garofalakis, Johannes Gehrke and Rajeev Rastogi*, published by *Springer-Verlag*.

assumption is not true, particularly those that ascribe more importance to recent data items. One way to discount old data items and only consider recent ones for analysis is the *sliding-window model*: Data elements arrive at every instant; each data element expires after exactly N time steps; and, the portion of data that is relevant to gathering statistics or answering queries is the set of last N elements to arrive. The sliding window refers to the window of active data elements at a given time instant and window size refers to N .

0.1 Motivation and Road Map

Our aim is to develop algorithms for maintaining statistics and models that use space sublinear in the window size N . The following example motivates why we may not be ready to tolerate memory usage that is linear in the size of the window. Consider the following network-traffic engineering scenario: a high speed router working at 40 gigabits per second line speed. For every packet that flows through this router we do a prefix match to check if it originates from the `stanford.edu` domain. At every instant, we would like to know how many packets, of the last 10^{10} packets, belonged to the `stanford.edu` domain. The above question can be rephrased as the following simple problem:

PROBLEM 0.1 (BASICCOUNTING) *Given a stream of data elements, consisting of 0's and 1's, maintain at every time instant the count of the number of 1's in the last N elements.*

A data element equals one if it corresponds to a packet from the `stanford.edu` domain and is zero otherwise. A trivial solution¹ exists for this problem that requires N bits of space. However, in such a scenario as the high-speed router, where on-chip memory is expensive and limited, and particularly when we would like to ask multiple (thousands) such continuous queries, it is prohibitive to use even $N = 10^{10}$ (window size) bits of memory for each query. Unfortunately, it is easy to see that the trivial solution is the best we can do in terms of memory usage, unless we are ready to settle for approximate answers, i.e. an exact solution to BASICCOUNTING requires $\Theta(N)$ bits of memory. We will present a solution to the problem that uses no more than $O(\frac{1}{\epsilon} \log^2 N)$ bits of memory (i.e., $O(\frac{1}{\epsilon} \log N)$ words of memory) and provides an answer at each instant that is accurate within a factor of $1 \pm \epsilon$. Thus, for $\epsilon = 0.1$ (10% accuracy) our solution will use about 300 words of memory for a window size of 10^{10} .

Given our concern that derives from working with limited space, it is natural to ask “Is this the best we can do with respect with memory utilization?” We answer this question by demonstrating a matching space lower bound, i.e. we show that any approximation algorithm (deterministic or randomized) for BA-

¹Maintain a FIFO queue and update counter.