An O(log n) Running Median or Running Statistic Method, for Use with BBC Jam Resistance

Leemon C. Baird III
William L. Bahn

Academy Center for Cyberspace Research
United States Air Force Academy

USAFA-TR-2009-ACCR-03

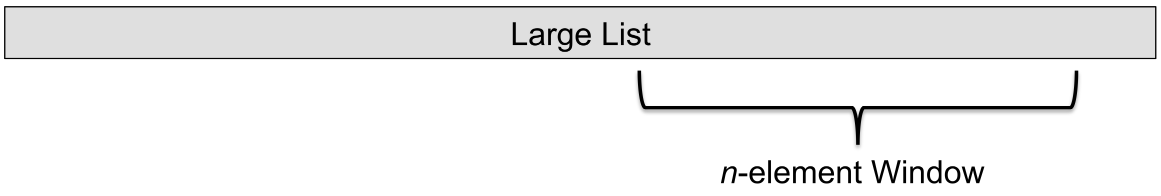
November, 2009

Abstract

Given a long list of real numbers, this paper describes an efficient algorithm for calculating the median (or other statistic) of a sliding window of size n. It requires O(n) memory, and O(log n) time after each slide. This is useful for an implementation of BBC jam resistance, where the list of numbers is the output of a correlator, and the running statistic algorithm is used to find a suitable threshold on each time step such that 2/3 of the numbers will map to 0, and 1/3 will map to 1. This allows efficient BBC decoding of the resulting packet, in the face of fluctuations in the power of the signal and the amount of noise in the environment.[1]

Background

Given a long list of real numbers, it can be useful to find the median of each sublist of *n* consecutive numbers. Or it can be useful to find a different statistic, such as the *k*th smallest number in each sublist, for some constant *k*. This *running statistic problem*, or *sliding window statistic problem* can arise in many contexts. This paper will give an algorithm for calculating this efficiently.



The problem without the sliding window has many known solutions. These find the median or *k*th smallest number of a single list of *n* numbers. A simple probabilistic algorithm is to perform a modified Quicksort. First a pivot is chosen, and then as in Quicksort, the entire list is partitioned into 3 sublists of those less than, equal to, and greater than the pivot. In Quicksort,

[1] This work was sponsored in part by the Air Force Information Operations Center (AFIOC), Lackland AFB, TX, and was performed at the Academy Center for Cyberspace Resaerch (ACCR) at the United States Air Force Academy.

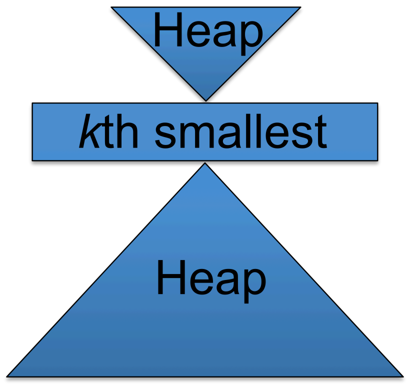
the algorithm would then recurse on both the "smaller" and "larger" partition. But for the *k*th statistic, the recursion need only be performed on one of those two lists. Simply look at how long each of the 3 partitions is, and recurse on the one that contains the *k*th smallest element. There is also a non-probabilistic algorithm that is more complex that involves partitioning the list into consecutive sublists of 5 elements, finding the median of each sublist, recursively finding the median of the medians, and then recursing on the set of numbers that is known to be greater or less than the median of medians.

These algorithms take O(*n*) time and space. A naïve application of these to the sliding window problem would take O(*n*) time for each possible position of the window. However, it would seem that there could be better algorithms for this, since two successive calculations would be dealing with two lists of numbers that are almost identical, differing in at most two numbers.

The equivalent problem in 2D is the median filter, which is typically found in most photo manipulation software. Many researchers have given algorithms for that case, and stated that they run in O(1) time for each window. However, they assume the numbers in question have a small bit depth, such as assuming all are 8-bit numbers. Here we will consider the 1D case of a simple list of numbers, but the numbers may be integers or fixed points real numbers with, say, 16 bits or 32 bits each. In this case, O(1) time per window will not be possible.

Running Statistic Algorithm

Suppose that we want the *k*th smallest element in each window of *n* consecutive elements in a large list. This can be accomplished for the first window position by putting all the elements into into a data structure that looks like this:



The bottom heap contains the *k*-1 smallest elements, and the upper heap contains the *n*-*k* largest elements. The element in the middle is the *k*th smallest element, which is the desired statistic. The bottom heap maintains its largest element at the root, and the upper heap maintains its smallest element at the root.

The double-heap data structure does not contain the actual elements. Instead, each position in the heaps contains a pointer into an n-element circular buffer that contains the actual elements to be sorted. A parallel array gives, for each element in the circular buffer, a pointer to the position in the double-heap that is pointing to that element. Thus the memory requirements are 4*n* numbers and 2*n* pointers, plus a handful of scalar variables.

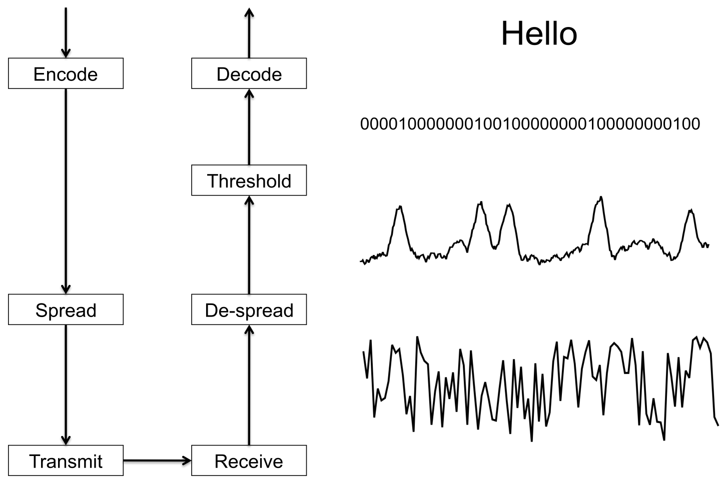
The algorithm starts with empty heaps and inserts all *n* elements from the initial window into the data structure. The first *k*-1 elements are inserted into the bottom heap. Then, if the kth element is greater than the root of that heap, it is placed in the middle. Otherwise, the maximum element in the bottom heap is removed from the heap and placed in the middle, and the kth element is inserted into that heap. Finally, the last *n-k* elements from the window are inserted, with each inserted into the top heap, bottom heap, or middle element as appropriate.

After the elements of the initial window are all inserted in the data structure, the middle element will be the desired statistic. It is then time to shift the window one element, and calculate the new statistic. The shift is done by adding a new element to the circular buffer, which overwrites the oldest element in the buffer. That oldest element is deleted from the double-buffer data structure, and the new element is inserted. This has an amortized time cost of O(*n*) operations, and the elements bubble up and down through the appropriate heaps. Then the answer again appears in the middle element.

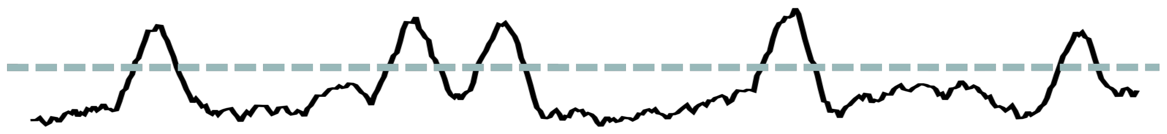
This algorithm is efficient. It takes O(*n*) space and O(*n*) time for each window position.

Application of the Algorithm

This algorithm is useful in decoding BBC-encoded (Baird, Bahn, Collins) jam resistant, wireless communication [1]. The process would look like this:



A message (”Hello”) is first encoded with BBC to yield a codeword that is mostly 0 bits with a few 1 bits. It is then spread to get a pseudo-random waveform that spreads each 1 in time and frequency. This waveform is transmitted through a wireless channel, and then received. The signal is de-spread by taking a correlation. The result is a signal like this:



That signal is then converted back to the encoded string by taking a threshold, and calling anything above the threshold a 1 and anything below a 0. This would be done using the running statistic algorithm, where *n* is the packet size, and *k*=*n*/3. The result is a packet with 1/3 of its bits set to 1, regardless of the amount of noise in the environment. BBC can efficiently decode such packets to find the messages contain within them. In this way, the described algorithm can form a useful component within a BBC communication system.

References

[1] Baird, Leemon C. III, Bahn, William L. & Collins, Michael D. (2007) Jam-Resistant Communication Without Shared Secrets Through the Use of Concurrent Codes, Technical Report, U. S. Air Force Academy, USAFA-TR-2007-01, Feb 14.