Statistical Techniques for Data Science and Robotics - Assignment1

Hussein Younes
h.younes@innopolis.university

1 Introduction

In this report, I shall summarize the steps taken to implement the MRL98 algorithm, and report the obtained results. MRL98 is an algorithm for computing approximate quantiles for a large dataset in a single pass, what makes it suitable for working with data streams. The implementation of this algorithm adapts a tree representation for the logical data storage units (buffers) suggested in [1] that will be discussed in details later in this report. it is worth noting that MRL98 is a deterministic algorithm, meaning that it will always produce the same qualntile value given the same input data and query conditions. The algorithm also requires a prior knowledge of the lenght N of the datastream. The implementation was written in python, in an OOP fashion.

2 Methodology

MRL98 is mainly parameterized by two integers b and k, where the incoming data is chunked into b blocks each containing exactly k elements. Therefore, the memory footprint will be bk elements. Each buffer X is associated with a positive integer $w(X)$, which denotes its weight. Intuitively, the weight of a buffer, is the number of input elements represented by each element in the buffer. The buffer is marked either *full* of *empty*. Initially, all b buffers are marked empty. The algorithm is composed from an interleaved sequence of basic operations, namelt NEW, COLLAPSE, and OUTPUT, which are described in the following subsections.

2.1 NEW operation

NEW takes an empty buffer as input, populates the input buffer with the following k elements from the input stream, and assigns it a weight of 1. NEW operation is invoked only if there's an empty buffer, and at least one more element in the input stream. If there are no enough data to fill the buffer with k elements (no more input data), some form of padding is applied, using $-\infty$ and $+\infty$ elements. It is worth nothing that the data in the buffer should be sorted to maintain the order of the sequence.

2.2 COLLAPSE operation

COLLAPSE takes $c \geq 2$ full input buffers, $X_1, X_2, ..., X_c$, and return a full buffer, Y, of size k. The input buffers all collapse in one input buffer, Y. All buffers after the COLLAPSE operation are marked empty, except the output buffer Y. The weight of the output buffer $w(Y)$ is set to the sum of all the weights of the input buffers $\sum_{i=1}^{c} w(X_i)$. The values of Y are computed as follows:

- Make $w(X_i)$ copied of each element in X_i
- create a sorted sequence, denoted by S, from the buffers' elements all together after copying the elements.
- The elements of Y then are the k equally spaced elements of S. If $w(Y)$ is odd, these k elements are in positions $iw(Y) + \frac{w(Y)+1}{2}$, for $i = 0, 1, ..., k-1$. Otherwise, the positions of the k elements are $iw(Y) + \frac{w(Y)}{2}$, for $i = 0, 1, ..., k-1$.

We note here that making $w(X_i)$ copies of each elements of X_i is computationally expensive and need not to be materialized during the implementation, instead, sort the buffers individually and then start merging them. During the merging, a counter (initialized to zero) gets incremented $w(X_i)$ times, and whenever the value of the counter hits one of the desired positions described above, the corresponding element is added to the output buffer Y. An example of the COLLAPSE operation is illustrated in Fig. 1.

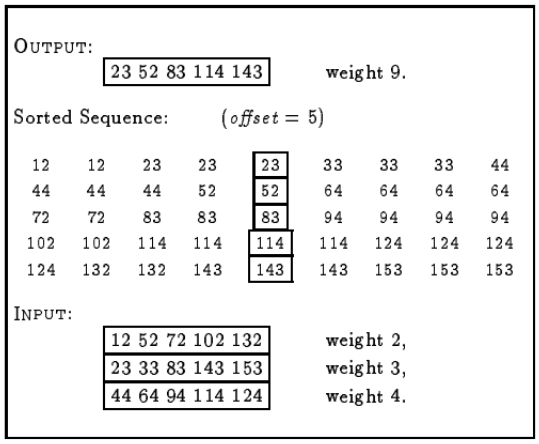

Figure 1. COLLAPSE illustrated [1]

2.3 OUTPUT

OUTPUT is performed exactly once, right before the termination of the algorithm. It takes $c \geq 2$ full input buffers as input, and returns a single element, the $\phi - quantile$ of the dataset. OUTPUT is invoked on the final set of full buffers, when there's no more data to process. Similar to COLLAPSE, the OUTPUT operation make $w(X_i)$ copies of each element

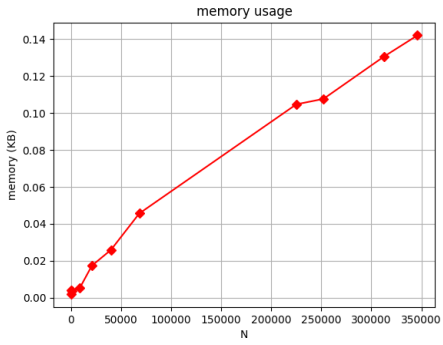

Figure 3. memory usage

in X_i and sorts all the input buffers together. The output is the element at position $\lceil \phi kW \rceil$ where ϕ is the queried quantile, k is the number of elements in each buffer, and W is the sum of weights of all input buffers $W = w(X_1) + w(X_2) + ... + w(X_c)$.

2.4 COLLAPSE policy

For each buffer X, we associate an integer $l(X)$ that denotes the level of the buffer. Let l be the minimum of all levels of the currently full buffer. If there is exactly one empty buffer, invoke NEW and assign it level l. If there are two or more empty buffers, invoke NEW on each buffer and assign it level 0. If there are no empty buffers, invoke COLLAPSE on the set of full buffers that are at level l and assign $l + 1$ to the output buffer.

3 Results

The above described algorithm was implemented and tested using python programming language, here's the link to the corresponding github repository. To simulate Data Streams, a Streamer was created to populated batched chunks of size k to feed the input buffers. Streams of different sizes where created and the following and the algorithm was tested against different sets of parameters. The following Fig. 2 illustrates the time consumption of the algorithm,

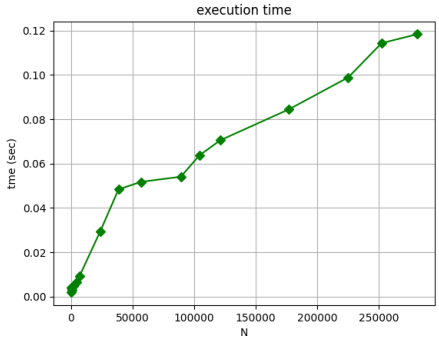

Figure 2. execution time

The time complexity is $O(\frac{1}{\epsilon} log^2(\epsilon N))$. Accordingly, the memory usage is of order $O(bk)$ as only $b*k$ need to be stored at run time. The following Fig. 3 illustrates the memory usage of the algorithm.

References

[1] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. 1998. Approximate medians and other quantiles in one pass and with limited memory. *ACM SIGMOD Record* 27, 2 (1998), 426–435.