A simple variation on the Greenwald-Khanna stream summary

November 30, 2018

1 Problem

Consider an infinite stream of (y_t, w_t) tuples, where $x_i \in U$, some totally ordered set, and $w_i \in \mathbb{R}^+$, the set of positive real numbers. The goal is scan the stream and maintain a data structure which at any point permits ϵ-approximate rank queries.

2 Naive solution

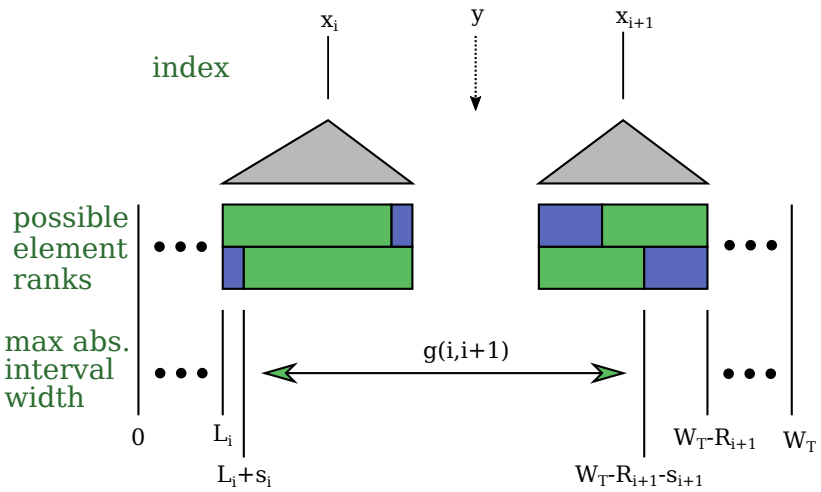
Let $t = 1, 2 \ldots$ be the indices of the input stream, and assume that the data-structure has been updated to account for the Tth element. It is straightforward to maintain the quantity $W_T = \sum_{t=1}^{T} w_t$. Beyond that, the algorithm maintains a collection of tuples of the form $(x_i, s_i, L_i, R_i)_{i=1}^{n_t}$.

Assume that $T-1$ pairs have been encountered, so that (y_T, w_T) is the current element. If $y_T = x_i$ for some tuple based on x_i, one adds w_T to s_i, and for all $j < i$, adds w_T to R_j, and for all $j > i$, adds w_T to L_j. If $y_T \neq x_i$ for all i, and i_- and i_+ at the indices of the immediate smaller and and larger elements, then one inserts the new element $\left(y_T, w_T, L_{i_-} + s_{i_-}, R_{i_+} + s_{i_+}\right)$, and updates all other x_i-tuples as before.

The inclusive range of ranks of the ith tuple is L_i to $W_T - R_i$; the bisected rank range is $L_i + s_i/2$ to $W_T - R_i - s_i/2$. (The doubly-exclusive range of ranks need not be nonempty.)

Our goal is to estimate the rank of a given element y with error at most ϵW_T. To achieve this goal, we maintain an invariant: that the maximum possible range of absolute ranks which could fall between two elements is $g\,(i, i+1) \coloneqq W_T - R_{i+1} - L_i - s_i - s_{i+1} \leq \epsilon W_T$. [1]

[1] This should not be confused with the maximum width between two elements, which is a local property.



Then, since y either matches some element's x value (in which case $W_T - R_i - L_i - s_i < \epsilon W_T$ is implied by the gap condition), or y falls inside the gap between x_i and x_{i+1}, and the range of possible values is $< \epsilon W_T$. Note that this invariant is preserved when new elements are inserted, as either the band is split, or $g\,(i, i+1)$ is preserved. To keep memory use and run-time low, we (eventually) delete all elements which can be deleted while preserving the invariant. (We delete node i when $W_T - R_{i+1} - L_{i-1} - s_{i-1} - s_{i+1} < \epsilon W_T$, so that afterwards the gap condition still holds.) The nontrivial part is proving that, for weights where $0 < \inf w_t < \sup w_t < \infty$, that this procedure obtains reasonable space bounds, and for that, see [2] Thm 1.

3 Partial-sum decomposition

This is equivalent to the naive representation; conversion between the two is straightforward; operations on the converted form have reduced cost.

Instead of storing tuples (x_i, s_i, L_i, R_i), store (x_i, s_i, l_i, r_i), with $i = 1 \ldots n_T$ in sorted order by x_i, where the properties that $\sum_{j=1}^{i-1} s_j + \sum_{j=1}^{i} l_j = L_i$ and $\sum_{j=i+1}^{n_t} s_j + \sum_{j=i}^{n_t} r_j = R_i$ are maintained.

Inserting a tuple equates to inserting $(y_T, w_T, 0, 0)$ in sorted position.

To delete (x_i, s_i, l_i, r_i), we remove it and increment its immediate neighbors to compensate, i.e, $l_{i+1} \mathrel{+}= l_i + s_i$, and $r_{i-1} \mathrel{+}= r_i + s_i$.

Useful interpretations of these coordinates:

- x_i is a sampled item
- s_i is the (minimum known) cumulative weight at item x_i
- l_i is the maximum possible value which can not, based on the current set of tuples, be ruled out from having been associated with a hypothetical value immediately less than x_i.
- r_i is like l_i, with *less* replaced by *greater*

3.1 Merging estimators

We can combine m different estimators by merging their partial-sum decomposed representations together, adding corresponding s_i, l_i, r_i values when multiple x_i match between different sources. This preserves relative uncertainty bounds.

Performing compression operations thereafter will reduce to an ϵ-approximate quantile summary.

4 Implementations

The specific implementations affect the run-time; however, de-amortization ensures $O(\log n_t)$ update time. Depending on the data structure, it may take $O(\log n_t)$ to query a single approximate rank, or $O(n_t)$ to uncompress the data structure, after which queries resolve to $O(\log n_t)$ binary search.

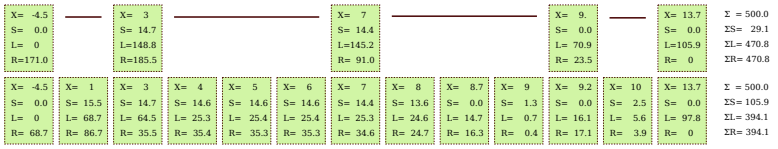
- A randomized skip list can be used to store the sorted list of all tuples at the base level; each higher level above summarizes the level below, so that the sum L_i can be computed in $O(\log n)$ time, by adding the highest level possible l_i and s_i terms on the path from the top left node to the ith base node.

  New entries (y, w) are inserted by locating the greatest i where $x_i \leq y$. If $x_i = y$, then w can be added to s_i. Otherwise, the L_{i-1} and R_{i+1} can be computed, in logarithmic time, so that one can check if a new added tuple would not immediately be deleted. (For instance, with uniform quantiles, one might immediately delete when $g(i-1, i+1) < \epsilon W$.) If the new added tuple is not deleted, an element $(x, s, 0, 0)$ should be inserted between i and $i+1$ at the base level and (depending on skip list rules) some number of levels above. In levels where a new tuple is not inserted, r_i and l_{i+1} should be incremented by w.

  Elements above or below the minimum or maximum element are always inserted; the old minimum or maximum should then be checked for deletion.

  Although a scheme for non-local deletions is likely possible (by tracking the minimum amounts added to the left and right which could induce a deletion), selecting a random base element and testing it for deletion at each phase likely suffices.

  This method has been implemented, and is somewhat slow (e.g. 1230 ns for 250 items, where for contrast an equivalent 1st-order interpolated histogram takes 80 ns) Contact mailbox `xslr` for source code. Example data structure diagram, produced from a mixed delta and smooth distribution, trending from left to right in time:



- A batch approach stores the tuples as a sorted array, and queues new elements in a priority heap. Periodically, one merges a sorted list produced from the heap with the sorted array. The now much larger list can be scanned linearly (with l_i and r_i updated incrementally) and tuples can be marked for deletion (based on the current state of the list). Finally, one removes every second deletion-marked tuple, symmetrically moving in from the left and the right.

  It's possible to amortize the batch operation by performing work in chunks of size $O(\log n)$. An additional heap can be used to queue items that arrived during the batch operation; although the existing heap can store the new items as well, i.e., by flipping the sign of the weights of incoming elements and adjusting the heap sort order to prefer the old items.

4.1 Misc

Sometimes the $(x, s, l, r)_i$ lists are produced in e.g. double precision format, but stored in single precision. If this conversion results in several adjacent entries with identical x, then their associated s, l, r values should be added to form a combined entry. (Note that there are cases in which the left r and right l values can be guaranteed to belong to x itself, e.g. when x is the least or greatest value.)

5 Analysis

Greenwald-Khanna is equivalent to maintaining an ϵ-approximate quantile summary.

This is a strict refinement to Greenwald-Khanna, in that the rank uncertainty provided by this method is bounded above by that estimate used by their algorithm. Operation run-time and space usage are thus the same.

See [1] for biased quantile estimation – instead of using the invariant that $g(i, i+1) < \epsilon W_T$, one could use some rank-dependent function on the right hand side, i.e. ϵL_i, to concentrate elements at low percentiles.

6 Generalizations

- For the circular case, one fix the first element in the stream, and define a total order over all y_t using the interval excluding it. Alternatively, without such a reference, one can record the total contribution in the band between any two elements; this uses n_T^2 memory usage, where n_T^2 is

the number of stored comparison elements at time T. Update time is still $\log n_T$. This may be more useful for more complicated range queries.

- Higher dimensional cases permit the same partial sum decomposition. In the general d-dimensional case, 3^d quantities per element would be required to count a new $(\boldsymbol{y}_t, w_t)$ pair.

7 Significance

The problem occurs, for instance, in Monte Carlo simulations estimating a temporal distribution from statistically/color/energy/otherwise weighted samples drawn from it. This model variation is slightly more intuitive (requires one less type of approximation) than the Greenwald-Khanna algorithm. This is important since Monte Carlo estimation already involves a complicated set of underlying assumptions.[2]

References

[1] Graham Cormode et al. "Effective computation of biased quantiles over data streams". In: *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*. IEEE. 2005, pp. 20–31.

[2] Michael Greenwald and Sanjeev Khanna. "Space-efficient Online Computation of Quantile Summaries". In: *SIGMOD Rec.* 30.2 (May 2001), pp. 58–66. ISSN: 0163-5808. DOI: 10.1145/376284.375670. URL: https://doi.acm.org/10.1145/376284.375670.

[2] The conceptually simpler histogram is not as memory efficient.