Toward More Efficient Mergeable Quantile Summaries

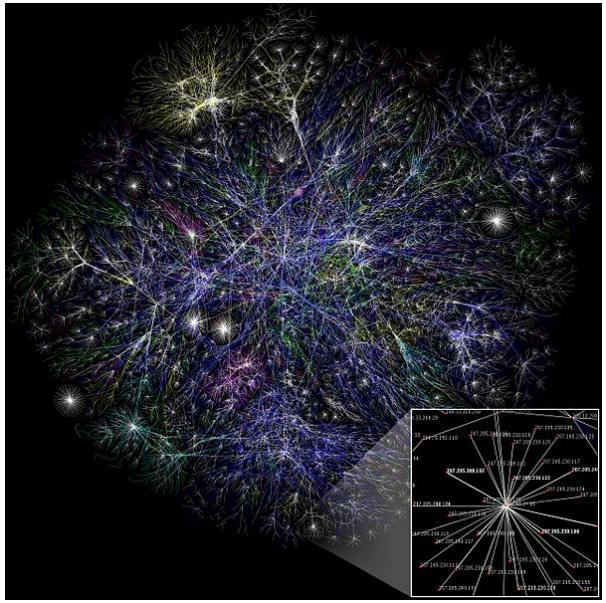
Dan Stubbs

Mentor: Graham Cormode

Outline

Review: Big Data

- In the modern world, with the size of availabe data sets increasing so quickly, RAM-model algorithms become increasingly unsuitable to answer many important questions, mostly related to the internet.



- Generally the problem of oversized data is solved by either viewing the data one element at a time (streaming), or by splitting the data into multiple segments (MUD).

Review: Summary Data Stuctures

- A standard procedure in any situation with big data is to maintain some summary as the bulk of the data gets processed and then discarded.
- In short, a summary data structure is any data structure that maintains information sufficient to answer some questions about an underlying data set in much less space than it would take to maintain the entire data set.

Review: ε-Approximate Quantile Summary

- An εAQS is a summary that can be given arbitrary quantile queries, and gives an answer whose rank is within ε times the size of the data set of the desired quantile.
- This can be accomplished by maintaining only $O(\varepsilon^{-1})$ different values, with each serving as the answer for an $O(\varepsilon)$-wide band of quantile queries.

Review: Mergeability of Summaries

- We say a summary is *mergeable* if two summaries S_1 and S_2 for disjoint data sets D_1 and D_2 with approximation bounds ε_1 and ε_2 can be used to construct a new summary S that $\max(\varepsilon_1, \varepsilon_2)$ approximates $D_1 \cup D_2$.
- Many summaries are in effect multiplication of the input's frequency vector by some random matrix. By associativity of matrix multiplication, summing two summary vectors constructed this way provides a summary of the sum of their frequency vectors, i.e. the union of their underlying data sets.

Outline

The GK Summary

- One type of εAQS that works in the streaming model
- It stores the minimum and maximum observations, as well as $O(\varepsilon^{-1}\log(\varepsilon n))$ other elements in between
- For each stored element, it tracks the minimum and maximum possible ranks that element could have
- When a quantile query is made, it returns some element x such that $(\phi-\varepsilon)n \leq \mathit{rmin}(x)$ and $(\phi+\varepsilon)n \geq \mathit{rmax}(x)$
- The space bound is maintained using the COMPRESS operation which makes up the heart of the algorithm's workings.

The COMPRESS Operation

- The goal of this operation is to keep as few elements as possible while maintaining enough elements to answer any quantile query, and to keep elements whose ranks are known more exactly.
- In order to maintain ε coverage, whenever an element is deleted, some adjacent element must take responsibility for the segment of the observations that element covered. WLOG, we make the next higher element responsible.
- Each stored element has an associated Band, roughly proportional to the log of the number of COMPRESS operations that have been made since it was observed. Elements of higher band have less error, so we disallow an element to consume an element of higher band – that is, an element of band higher than the stored element of immediately greater value cannot be deleted
- The paper shows that only $O(\varepsilon^{-1})$ elements per band can exist without consuming each other and that there will only be $O(\varepsilon^{-1})$ elements per band unable to consume the element below them due to its having higher band.
  - All told, this will use $O(\varepsilon^{-1})$ elements in each of the $O(\log(\varepsilon n))$ bands, for $O(\varepsilon^{-1} \log(\varepsilon n))$

Algorithm Jenga

- First, we put together a simulator to test variants on the basic algorithm
  - Mainly removing pieces to see if the algorithm stopped working
- Unsurprisingly, many pieces of the algorithm turned out to be mandatory
  - For instance, forgoing bands and using raw Δ (Compress-count) values leads to potential space overuse
- On the other hand, we did find some pieces that were extraneous, like a further restriction that a Compress could only be made if every contiguous element of lower band to the left of the element to be deleted could go with it

Making GK Merge

- Given the amount of thought given to mergeability in the pre-algorithm portion of the paper, creating a mergeable version of the GK summary is a question that emerges naturally
  - Some properties of the algorithm that seemed simple actually depend on the serial nature of the updates
  - A few portions of the proof make explicit mention of "arrival time," a difficult concept in the mergeable model
- We tried a few options, but our current solution is to alter the Δ values when a merge occurs to simulate serialness
- This seems to multiply space usage by $\log n$

Conclusion

- If it turns out to not increase space too much, the "pseudo-serial" model of mergeability may serve as a useful tool for converting streaming algorithms into mergeable algorithms
- It's quite possible that the log factor listed above is an overstatement of the true cost; proving this is a likely direction for future work
- *Thank you!* Any questions?