Research Proposal: *Decentralized indices for genomic data*

Luiz Irber[1]

[1]University of California, Davis

11 April 2019

Introduction

Background

Problem Description

Data sketches

A data sketch is a representative proxy for the original data focused on queries for specific properties. It can also be viewed as a probabilistic data structure (in contrast with deterministic data structures), since it uses hashing techniques to provide statistical guarantees on the precision of the answer for a query. This allows a memory/accuracy trade-off: using more memory leads to more accurate results, but in memory-constrained situations it still bound results to an expected error rate.

MinHash sketch: similarity and containment

The MinHash sketch [5] was developed at Altavista in the context of document clustering and deduplication. It provides an estimate of the Jaccard similarity $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$ (called **resemblance** in the original article) and the **containment** of two documents $C(A, B) = \frac{|A \cap B|}{|A|}$, estimating how much of document A is contained in document B. These estimates depend on two processes: Converting documents to sets ("Shingling"), and transforming large sets into short signatures, while preserving similarity ("Min-Hashing"). In the original use case the *w-shingling* Ω of a document D is defined as the set of all continuous subsequence of w words contained in D. *Min-hashing* is the process of creating $W = \{h(x) | \forall x \in \Omega\}$, where $h(x)$ is an uniform hash function, and then either

- keeping the n smallest hash values as a representative sketch of the original document ($\mathbf{MIN}_n(W)$)
- keeping elements that are $0 \mod m$ ($\mathbf{MOD}_m(W)$).

$\mathbf{MIN}_n(W)$ if fixed-sized (length n) and supports similarity estimation, but doesn't support containment. $\mathbf{MOD}_m(W)$ supports both similarity and containment, with the drawback of not being fixed-sized anymore, growing with the complexity of the document.

Mash [14] was the first implementation of MinHash in the genomic context, relating the w-shingling of a document to the k-mer composition of a genomic sequence, and using the $\mathbf{MIN}_n(W)$ fixed-sized formulation for the signature. Mash needs extra information (the genome size for the organism being queried) to account for genomic complexity in datasets. This extra information is required because using a fixed-size MinHash leads to different degrees of accuracy when comparing across highly-diverged organisms (bacteria to animals, for example), and it is even more extreme when taking more complex datasets into account (like metagenomes).

Bloom Filter: Set membership

The Bloom Filter [4] allows set membership queries. Inserting an element in a Bloom Filter is a two-step process: first use multiple hash functions on the element, and then for each bit set in the hashed value update the same position in the bit array. Querying an element involves calculating the multiple hash values for the element and then checking if the bits are set in the bit array. This guarantees that a false negative is impossible (if the element was inserted, a bit would be set), but can report false positives if there are collisions in the hash values.

khmer [7] implements a variation where longer (and multiple) bit arrays are used, and hash functions are derived from a composed hashing strategy $h_i(x) = h(x) \mod p_i$, where each bit array has a distinct length p_i (and p is a prime number), with $h(x)$ being a more CPU-intensive hash function.

Bloom Filters are used extensively in bioinformatics, including lossy representation of assembly graphs [15] and as a filtering step in processing pipelines [14].

HyperLogLog: Cardinality estimation

The HyperLogLog sketch [10] estimates the number of unique elements in a dataset. It is designed to lower the variability of a more basic estimator: given a run of ρ zeros in a binary sequence, it estimates the cardinality of the dataset to be 2^ρ. It achieves this by splitting the binary sequence in two: the lower bits define an index for m registers, and each register contain the longest run of zeros seen for that index. The HyperLogLog estimator $E(D)$ is an harmonic mean of the registers M, with a correction factor α_m for the number of registers:

$$E(D) = \alpha_m m^2 \left(\sum_{j=0}^{m-1} 2^{-M[j]} \right)^{-1}$$

More recent methods [12] improve the cardinality estimator by further refining the estimate based on empirical data.

In genomic contexts, the khmer [7] implementation of HyperLogLog [13] uses the k-mer composition of a genomic dataset and the *murmurhash3* hash function, together with the improved estimator from [12]. Dashing [2] is a recent method that supports both similarity and cardinality estimation using HyperLogLog, based on a better estimator for union and intersection of HyperLogLog sketches by [9].

Hierarchical index

Searching for matches in large collection of datasets is not viable when hundreds of thousands of them are available, especially if they are partitioned and not all present at the same place.

Bloofi [6] is a hierarchical index structure that extends the Bloom Filter basic query to collections of Bloom Filters. Instead of calculating the union of all Bloom Filters in the collection (which would allow answering if an element is present in any of them) it defines a tree structure where the original Bloom Filters are leaves, and internal nodes are the union of all the Bloom Filters in their subtrees. Searching is based on a breadth-first search, with pruning when no matches are found at an internal level. Bloofi can also be partitioned in a network, with network nodes containing a subtree of the original tree and only being accessed if the search requires it.

The Sequence Bloom Tree [16] adapts Bloofi for genomic contexts, rephrasing the problem as experiment discovery: given a query sequence Q and a threshold θ, which experiments contain at least θ of the original query Q? Experiments are encoded in Bloom Filters containing the k-mer composition of transcriptomes, and queries are transcripts.

Further developments focused on clustering similar datasets to prune search early [18] and developing more efficient representations for the internal nodes [17] [11] to use less storage space and memory.

Decentralized querying

Persistent Data Structures [8],

IPFS [3],

SRA closure [1],

A little centralization [20]

Aims

1. **Adapting genomic MinHash for containment and cardinality estimation.** The original MinHash article [5] defines two estimates for similarity resemblance and containment, with two variations of the MinHash sketch, one with fixed length, but only supporting resemblance, and another with variable length, supporting both resemblance and containment. Mash [14] uses the fixed length sketch, and defines a new distance metric called Mash distance to account for the size of the genomes being compared. The resemblance estimate works well for genomes of similar size, but when dealing with datasets of highly-diverged organisms or even more complex datasets (like metagenomes) the containment estimate is more appropriate and closer to the sort of problems that biologists need to solve.

   In sourmash [19] I implemented a variable length MinHash sketch (with fallback to the fixed length case for compatibility with Mash) called the scaled MinHash. The scaled MinHash supports containment estimation, allowing new applications not available on previous methods.

   Sketches are data structures planned for specific classes of queries, but it is possible to support additional use cases. For example, dashing [2] is a method that supports both similarity and cardinality estimation using HyperLogLog sketches [10], a data structure initially derived for supporting cardinality estimation and later extended to similarity cases [9], Even so, it doesn't support containment estimation.

   Scaled MinHash can be adapted to also support cardinality estimation. Due to how the scaled MinHash is constructed the same estimator from HyperLogLog can be used for it.

2. **Fast queries on many MinHash sketches using MinHash Bloom Trees** [16] introduces the Sequence Bloom Tree, an hierarchical index data structure for finding a query sequence in large dataset collections. It represents the k-mer composition of a dataset using a Bloom Filter [4], and the hierarchical aspect comes from organizing multiple dataset Bloom Filters into a tree structure, where the leaves are the dataset Bloom Filters and the internal nodes are Bloom Filters containing all k-mers below it. A query is evaluated by doing a breadth-first search of the tree, and truncating the search when the query is not present over a pre-determined threshold.

   I adapted the Sequence Bloom Tree to use MinHash sketches as dataset representations, referred as MinHash Bloom Tree from now on to highlight how they are distinct. Since a MinHash is a subset of the k-mer composition of a dataset, internal nodes are still Bloom Filters, but this time containing all the k-mers present in the MinHash.

   On top of supporting the search method defined by the Sequence Bloom Tree (a breadth-first search with early truncating on a similarity or containment threshold), the MinHash Bloom Tree index also supports another search method called `gather`, a variation of Best-First search using containment estimation. `gather` can be used for doing taxonomic classification of genomic datasets, finding all organisms present in a sample and how abundant they are. This is especially important when working with metagenomes, a dataset containing sequenced genomic data from an environmental sample (be it soil, ocean or human gut, for example) representing a community of organisms. `gather` does iterative Best-First searches, at each step removing matches from the original query, until there are no more hashes to search or a detection threshold is reached.

3. **Decentralized indices for genomic data.** The MinHash Bloom Tree can be viewed as a persistent data structure, since leaves never change once added and internal nodes only change if a new leaf is added to its subtree. This makes it a very good fit for content-addressable storage methods, and I'm exploring two different decentralized data storage systems (IPFS and dat) as ways of storing and interacting with MinHash Bloom Tree indices.

   On top the data storage aspects, another important point is how researchers can interact with these indices (both querying and updating it) in a way that centralized storage is not essential (but operations are optimized if it is). This is important in the context of long term sustainability of projects, something often overlooked in bioinformatics systems. The initial implementation is centralized for simplicity and prototyping the user interaction, supporting data submission and querying, with a data pipeline based on the experience downloading 800k+ microbial datasets from NCBI SRA and also the preparation of indices for the IMG database from JGI (60k+ datasets).

   Once this prototype is functional and we find what features are desirable, I plan to start decentralizing this process by defining updates in terms of CRDT operations, publishing a feed of these changes and moving to PubSub messaging between remote sites, allowing them to update their indices and keep them consistent with ours.

References

[1] Closure of the NCBI SRA and implications for the long-term future of genomics data storage. 12:402. ISSN 1474-760X. doi: 10.1186/gb-2011-12-3-402. URL http://dx.doi.org/10.1186/gb-2011-12-3-402.

[2] D. N. Baker and B. Langmead. Dashing: Fast and accurate genomic distances with HyperLogLog. page 501726. doi: 10.1101/501726. URL https://www.biorxiv.org/content/early/2018/12/20/501726.

[3] J. Benet. IPFS - content addressed, versioned, p2p file system. URL http://arxiv.org/abs/1407.3561.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. 13(7):422–426. ISSN 0001-0782. doi: 10.1145/362686.362692. URL http://doi.acm.org/10.1145/362686.362692.

[5] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE. URL http://ieeexplore.ieee.org/abstract/document/666900/.

[6] A. Crainiceanu and D. Lemire. Bloofi: Multidimensional bloom filters. 54:311–324. ISSN 03064379. doi: 10.1016/j.is.2015.01.002. URL http://arxiv.org/abs/1501.01941.

[7] M. R. Crusoe, H. F. Alameldin, S. Awad, E. Boucher, A. Caldwell, R. Cartwright, A. Charbonneau, B. Constantinides, G. Edvenson, S. Fay, and others. The khmer software package: enabling efficient nucleotide sequence analysis. 4. URL https://f1000research.com/articles/4-900/v1.

[8] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. 38 (1):86–124. ISSN 0022-0000. doi: 10.1016/0022-0000(89)90034-2. URL http://www.sciencedirect.com/science/article/pii/0022000089900342.

[9] O. Ertl. New cardinality estimation algorithms for HyperLogLog sketches. URL http://arxiv.org/abs/1702.01284.

[10] P. Flajolet, . Fusy, O. Gandouet, and F. Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. (1). URL http://www.dmtcs.org/dmtcs-ojs/index.php/proceedings/article/viewArticle/914.

[11] R. S. Harris and P. Medvedev. Improved representation of sequence bloom trees. page 501452. doi: 10.1101/501452. URL https://www.biorxiv.org/content/early/2018/12/19/501452.

[12] S. Heule, M. Nunkesser, and A. Hall. HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 683–692. ACM.

[13] L. C. Irber Junior and C. T. Brown. Efficient cardinality estimation for k-mers in large DNA sequencing data sets. page 056846. URL http://www.biorxiv.org/content/early/2016/06/07/056846.abstract.

[14] B. D. Ondov, T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy. Mash: fast genome and metagenome distance estimation using MinHash. 17:132. ISSN 1474-760X. doi: 10.1186/s13059-016-0997-x. URL http://dx.doi.org/10.1186/s13059-016-0997-x.

[15] J. Pell, A. Hintze, R. Canino-Koning, A. Howe, J. M. Tiedje, and C. T. Brown. Scaling metagenome sequence assembly with probabilistic de bruijn graphs. 109(33):13272–13277. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1121464109. URL http://www.pnas.org/content/109/33/13272.

[16] B. Solomon and C. Kingsford. Fast search of thousands of short-read sequencing experiments. 34(3): 300–302, . ISSN 1087-0156. doi: 10.1038/nbt.3442. URL https://www.nature.com/nbt/journal/v34/n3/full/nbt.3442.html.

[17] B. Solomon and C. Kingsford. Improved search of large transcriptomic sequencing databases using split sequence bloom trees. In *International Conference on Research in Computational Molecular Biology*, pages 257–271. Springer, .

[18] C. Sun, R. S. Harris, R. Chikhi, and P. Medvedev. AllSome sequence bloom trees. In *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 272–286. Springer, Cham. ISBN 978-3-319-56969-7 978-3-319-56970-3. doi: 10.1007/978-3-319-56970-3_17. URL https://link.springer.com/chapter/10.1007/978-3-319-56970-3_17.

[19] C. Titus Brown and L. Irber. sourmash: a library for MinHash sketching of DNA. 1(5). doi: 10.21105/joss.00027. URL http://joss.theoj.org/papers/10.21105/joss.00027.

[20] J. N. Tsitsiklis and K. Xu. On the power of (even a little) centralization in distributed processing. 39 (1):121–132.