CS 5968/6968: Data Str & Alg Scalable Computing April 5th, 2023

Lecture 17: Graphs in Computational Biology (cont.)

Prof. Prashant Pandey *Scribe: Aakash Kulkarni*

1 Overview

In the last lecture we covered graphical assembly algorithms used in computational biology such as the de Bruijn graph and Overlap-Layout-Consensus.

In this lecture we will consider the sample discovery problem and introduce solutions to this problem. We will also discuss scalability issues with some of the solutions, and introduce the colored de Bruijn graph to address those issues.

2 Sample Discovery Problem

2.1 Definition

We want to know if a given transcript of a DNA sequence appears in samples acquired from raw sequencing datasets (which contain errors). There may be 1 to 100 million samples, each storing 2-100 GB worth of data. These samples are stored in the NIH SRA database and are anonymized.

2.2 Applications

Finding out if a strain of COVID is a novel variant is an instance of the sample discovery problem. It is critical to find an answer to this fast, so preventive measure can be implemented.

3 Sequence Bloom Tree (SBT) based approaches

Here's how a SBT-based approach solves the sample discovery problem:

1. Decompose the transcipt and each sample into k-mers.
2. If more than a Θ fraction of k-mers from a query appear in a sample, then there is a high chance that the query appears in that sample.

3.1 Construction of SBT

A sequence bloom tree (SBT) is a rooted binary tree consisting of bloom filters. The leaf nodes are bloom filters built directly on top of the samples' k-mers. All other nodes are bloom filters obtained by merging its children bloom filters. Figure 1 shows a visual representation of a SBT.

An important property of the SBT is that all bloom filters are of the same size, so smaller samples will have sparser filters, whereas larger samples will have dense filters.

A search operation over the SBT tree can be done as follows:

1. Build bloom filter on top of query transcript which is the same size as the bloom filters in the SBT.
2. Perform an intersection of the root bloom filter and the query bloom filter.
3. Prune search space based on % match, i.e., if a node's bloom filter doesn't give $> \Theta$ match, then prune the entire subtree starting at that node.

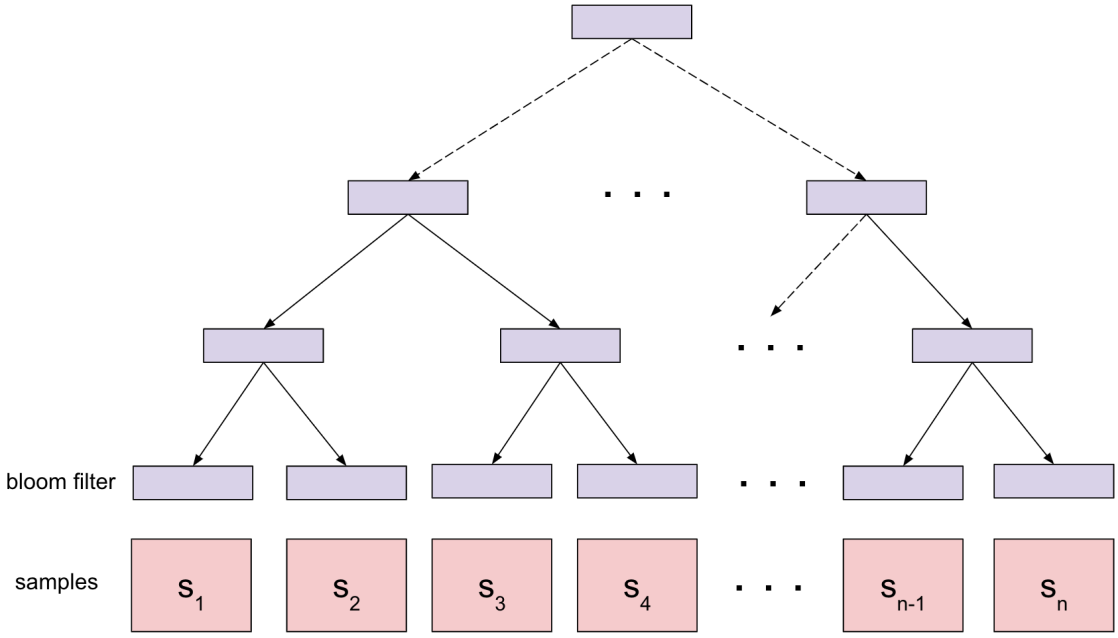


Figure 1: Sequence Bloom Tree

3.1.1 Issues with SBT

The SBT is not an optimal way of solving the sample discovery problem as there are numerous issues. Each node's bloom filter (other than leaf nodes) is derived by doing a merge operation on its children bloom filters. To do a bloom filter merge, the two filters *must* be of the same size. In this case, sample size variance is very high, so the bloom filters will be very large. Here are all the issues summarized:

1. Not all samples are of same size leading to bloom filters of highly-varying density
2. Random accesses required to do a search operation
3. Skewed paths in the tree caused by larger sample sizes, because there will naturally be a higher % match on the bloom filters

4. Highly approximate data structure: bloom filters are already an approximate data structure, which is made worse by the fact that samples may be error-prone

3.2 Variances of SBT

Solutions have been proposed to resolve some of the issues that the standard SBT introduces. One approach you could take is to cluster samples based on sample similarity together using locality-sensitive hashing or other light-weight clustering.

Another approach is to split large samples at the leaf nodes into smaller samples, which provides a decrease in space usage. Split Sequence Bloom Trees (SSBT) [1] and AllSome Sequence Bloom Trees [2] are two approaches that use splitting that provide both a performance speedup and optimize space.

Here are the space usage comparisons of different approaches based on the Blood-Brain Barrier database (b3db) [3]:

- Standard Sequence Bloom Tree: 97 Gb
- Split Sequence Bloom Tree: 40 Gb
- AllSome Sequence Bloom Tree: 40 Gb
- Mantis (Inverted Index): 32 Gb

4 Mantis (Inverted Index)

Mantis [4] is an exact index which is 20% smaller than the SSBT index. A Mantix index is constructed by decomposing the samples into k-mers and making a mapping of unique k-mers to the sample subsets they appear in. Figure 2 shows a toy example and the subsequence Mantis index construction.

To handle queries, you need to first decompose the query into k-mers, and then get all samples (color sets) in which those query k-mers appear and return an intersection of the samples. Notice that there is no approximation in this data structure, and it returns exact matches. To improve accuracy, you can also return a weighted intersection based on how many query k-mers appear in a specific sample subset.

4.1 Scalability Challenges

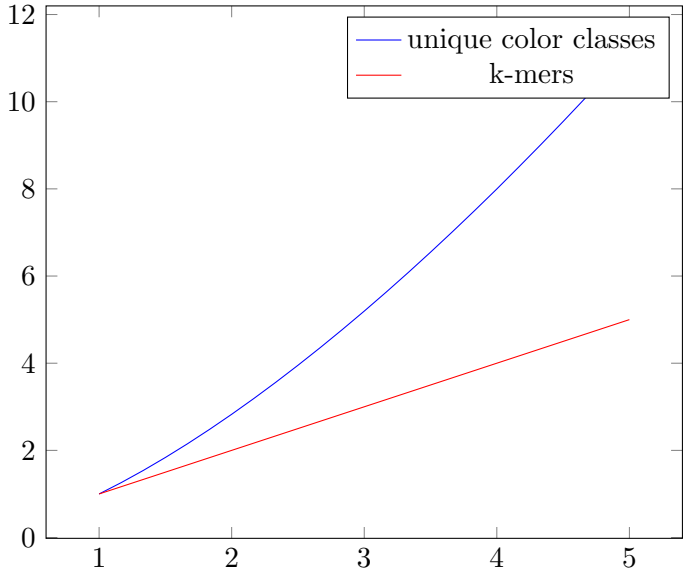
The Mantis index provides a fast query and scales well for up to index built on thousands of samples. However, we really need to build an index on $10^5 - 10^6$ number of samples.

4.2 Key Observations and Exploits

There are a couple of key observations of the Mantix index that help build intuition for the colored de Bruijn graph:

1. k-mers grow *at-worst* linearly
2. color classes (sample subsets) grow super-linearly

We can exploit the coherence between rows of the color class because if we know a k-mer's color subset, then we also know a lot about the k-mer's neighbors color subsets. We also know of a way to represent the relationship between all k-mers in a given transcript: the de Bruijn graph! Combining these ideas, we have built up intuition for the colored de Bruijn graph.



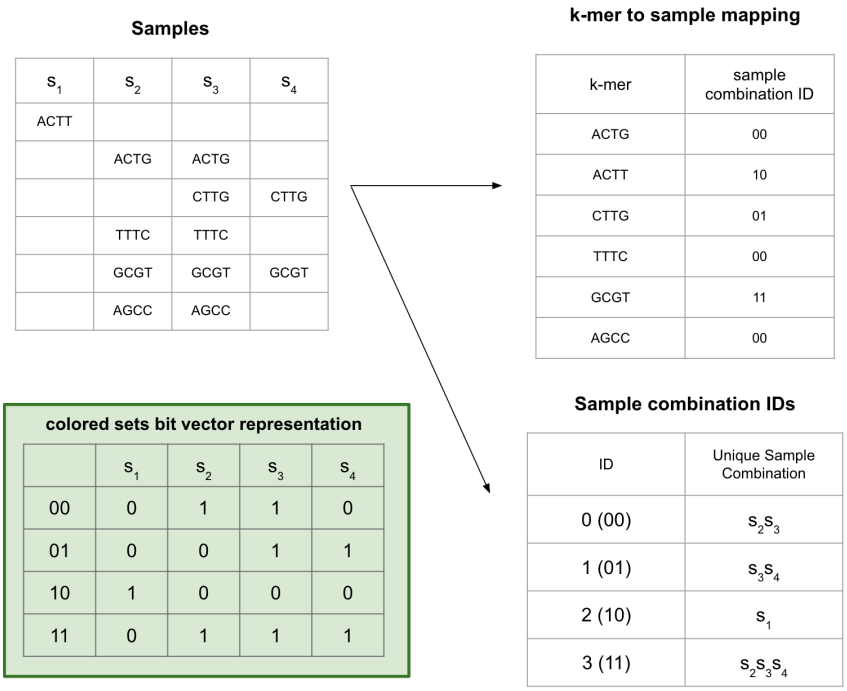
5 Colored de Bruijn graphs

We want to find distance between different bit vectors storing color set information for each sample. We notice that we only really need to store a few bit vectors and we can derive the other ones using delta encodings.

5.1 Color Class Graph

We can store the bit vector representation as a color class graph where each vertex is a color class and each pair of vertices is connected. To introduce notion of distance between color sets, we could use the Hamming distance as the edge weights.

There are some serious issues with this construction however, even if we derive the minimum spanning tree (MST) of this graph. First, there are, in the worst-case, $O(N^2)$ vertices and they are high dimensional. Another downside is that neighbor-search within this graph is expensive. Clustering methods such as locality sensitive hashing don't have enough information gain from each vertex to provide any probabilistic guarantees.



s_1	s_2	s_3	s_4
ACTT			
	ACTG	ACTG	
		CTTG	CTTG
	TTTC	TTTC	
	GCGT	GCGT	GCGT
	AGCC	AGCC	

k-mer	sample combination ID
ACTG	00
ACTT	10
CTTG	01
TTTC	00
GCGT	11
AGCC	00

	s_1	s_2	s_3	s_4
00	0	1	1	0
01	0	0	1	1
10	1	0	0	0
11	0	1	1	1

ID	Unique Sample Combination
0 (00)	s_2s_3
1 (01)	s_3s_4
2 (10)	s_1
3 (11)	$s_2s_3s_4$

Figure 2: Mantis Index Construction

5.2 Using de Bruijn Graphs to implicitly store colored sets

We know that we can use a de Bruijn graph consisting of all k-mers and also represent relationships between each k-mer. We also know that in this case, there will be, *in the worst case*, $O(N)$ k-mers. So, the de Bruijn graph will be asymptotically smaller than the color class graph.

We also noted previously that there is coherence between rows of the color class because if we know a k-mer's color class, we also know information about the k-mer's neighbors' color classes. The de Bruijn graph imposes biological relationships between color class bit vectors.

We build a **colored de Bruijn graph** by storing each colored set bit vector as a vertex and overlay the de Bruijn graph to obtain the edges which encodes relations between colored sets. This graph only has $O(N)$ edges which is much better than a standard color class graph. We add that the MST derived from this de Bruin graph based color class graph is similar to the MST derived from the complete color class graph. C We can also build delta encoding on top of this MST to obtain the colored de Bruijn graph [5].

There are variants of the colored de Bruijn graph optimized for space such as the succint colored de Briijn graph [6] and BiFrost [7].

References

[1] B. Solomon and C. Kingsford, "Improved search of large transcriptomic sequencing databases using split sequence bloom trees," *bioRxiv*, 2016. [Online]. Available: https://www.biorxiv.org/content/early/2016/12/02/086561

[2] C. Sun, R. S. Harris, R. Chikhi, and P. Medvedev, "Allsome sequence bloom trees," *bioRxiv*, 2017. [Online]. Available: https://www.biorxiv.org/content/early/2017/03/23/090464

[3] F. Meng, Y. Xi, J. Huang, and P. W. Ayers, "A curated diverse molecular database of blood-brain barrier permeability with chemical descriptors," *Scientific Data*, vol. 8, no. 1, p. 289, Oct 2021. [Online]. Available: https://doi.org/10.1038/s41597-021-01069-5

[4] P. Pandey, F. Almodaresi, M. A. Bender, M. Ferdman, R. Johnson, and R. Patro, "Mantis: A fast, small, and exact large-scale sequence-search index," *Cell Systems*, vol. 7, no. 2, pp. 201–207.e4, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405471218302394

[5] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean, "De novo assembly and genotyping of variants using colored de bruijn graphs," *Nature Genetics*, vol. 44, no. 2, pp. 226–232, Feb 2012. [Online]. Available: https://doi.org/10.1038/ng.1028

[6] M. D. Muggli, A. Bowe, N. R. Noyes, P. S. Morley, K. E. Belk, R. Raymond, T. Gagie, S. J. Puglisi, and C. Boucher, "Succinct colored de bruijn graphs," *Bioinformatics*, vol. 33, no. 20, pp. 3181–3187, Oct. 2017.

[7] G. Holley and P. Melsted, "Bifrost: highly parallel construction and indexing of colored and compacted de bruijn graphs," *Genome Biology*, vol. 21, no. 1, p. 249, Sep 2020. [Online]. Available: https://doi.org/10.1186/s13059-020-02135-8