CS 7280/4973: Data Str & Alg Scalable Comp Spring 2025

Lecture 21 — March 31, 2025

Prof. Prashant Pandey *Scribe: Ansh Mehta*

1 Overview

In the last lecture, we revisited static graph representations (adjacency matrices, edge lists, adjacency lists, CSR) and then explored how to handle streaming graphs, focusing on systems like STINGER, LLAMA, Aspen, and Terrace. We discussed the impact of cache locality on performance, emphasizing how data structures (e.g., purely functional trees and PMAs) can optimize updates and queries in dynamic environments.

In this lecture, we look at **practical applications of graph** and how they are used in the **real world**.

Specifically, we deep dive into the world of computational biology, specifically the **genomic assembly problem**.

2 Computational Biology

What is assembly problem? It refers to reconstructing a DNA sequence (or **genome**) from a collection of sampled DNA fragments (**reads**). It is one of the most computationally intensive problems in CS.

Its applications are in the fields of genomics (determining the complete sequence of an organism's genome), meta-genomics (reconstructing the genomes of microbial communities from environmental samples), etc.

2.1 Some Background

A typical genomic sequencing (say at a Lab) follows the process as illustrated in Figure 1.

A biological sample containing DNA (represented by bases A, C, T, G) is collected from a patient and processed by a sequencing instrument. The instrument outputs numerous **short DNA fragments** called **reads**.

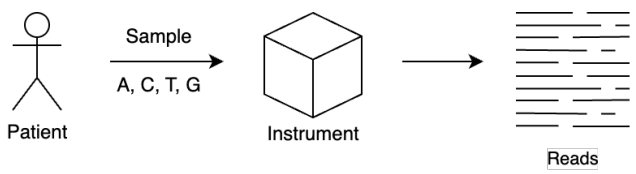

Figure 1: Basic workflow of genomic sequencing:

This instrument performs nucleic acid sequencing to identify the the base pair(A, C, G, T/U) in DNA or RNA. Using light emissions, one can identify specific nucleotide bases and determine the sequence. PCR methods (Polymerase Chain Reaction) are used for target amplification prior to the sequencing workflow.

2.2 Reads

Reads represents chunks of the actual full genome. For reference: a human genome consists of 3B base pairs. The instrument can only read chunks at a time, and cannot read the whole 3B base pair genomic sequence is a single shot. Hence we get **multiple chunks (called reads).**

2.2.1 Sequencing Errors in Reads

Sometimes instruments output some of the read chunks incorrectly. It might be attributed to some arbitrary or random chemical reaction.

Depending on the configuration of the instrument, we can get two types of reads, and both depicting different amount of errors.

1. **Short Reads**: Consists of 100-200 base pairs (BP), $0.5\% - 1\%$ errors.
2. **Long Reads**: Consists of 5K-10K BPs, $2\% - 12\%$ errors.

For example, for a Human Reference Genome ($3B$ base pairs), the instrument if configured for short reads would typically output:

1. **In Short Reads**: $\frac{3B}{200} = 15M$ reads.
2. **In Long Reads**: $\frac{3B}{10000} = 300K$ reads.

2.2.2 Types of Errors

There are 3 types of errors, illustrated in Figure 2.

1. **Insertion**: An extra base is incorrectly added into the read sequence compared to the original genome.
2. **Deletion**: A base from the original genome sequence is missing in the read sequence.
3. **Substitution**: A base in the read sequence is replaced by a different base compared to the original genome.

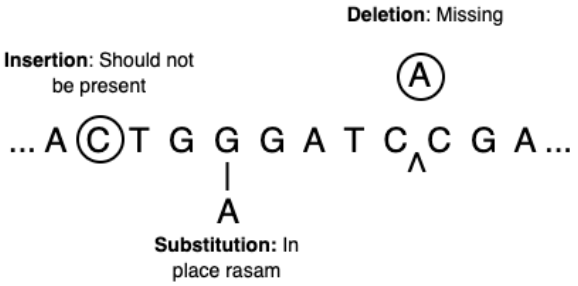

Figure 2: Types of Sequencing Errors

2.2.3 How to compensate for errors

For compensating, we can retry a particular region of the genome multiple times. This is called **Sequencing Depth**. Often a particular region is sequenced $30X - 50X$ times. Each time we sequence it might be possible that the prefix or the suffix is different and of different lengths. This technique helps as most of the times we are reading the correct, and errors are random, so the erroneous cases will have less frequency.

1. **Total Reads (Short Reads):**
   - Lower bound: $15\text{M} \times 30 = 450\text{M}$ reads
   - Upper bound: $15\text{M} \times 50 = 750\text{M}$ reads

   Total $\approx$ 450M - 750M short reads.

2. **Total Reads (Long Reads):**
   - Lower bound: $300K \times 30 = 9\text{M}$ reads
   - Upper bound: $300K \times 50 = 15\text{M}$ reads

   Total $\approx$ 9M - 15M long reads.

2.2.4 Some notes about reads

1. There is no order among the reads which the instrument outputs.
2. The reads are of different lengths.
3. There might be overlap in pairs of reads.

3 Assembly Problem

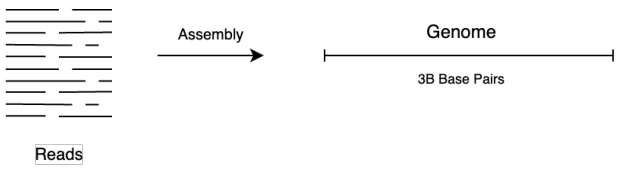

Figure 3: Assembly Process

3.1 Types of Assembly

1. **Reference Assembly**: Create a mapping for the read using the reference genome. This is like a puzzle where we arrange pieces knowing the actual final build. [2]
2. **De Novo Assembly**: We only have the reads and do not have the reference.

   For example: Rare species or organisms where we have the reads, but since its the first time we encounter such species, there is no reference sequence.

3.2 Assembly Algorithms

1. **OLC - Overlap Layout Consensus**
2. **DBG - de Bruijn Graph** [1]

We first discuss the DBG algorithm, and later discuss the OLC approach.

3.3 DBG - de Bruijn Graph [N.G de bruijn 1918-2012]

Before forming the de-bruijn graph, we must identify and remove the errors which occurs in the read from the instrument.

3.3.1 Error Correction Pipeline

Definition - k-mer: Similar to bi-gram, tri-grams in language modelling, we have k-mer which is a sequence of length k within a larger sequence.

Steps to remove errors.

1. We form something called k-mers. This is done using sliding window on every read. We set k between 20 - 30, and form $|R| - |k| + 1$ k-length strings. Figure 4, illustrates this process.
2. For every k-mer in every read, we insert it into a frequency map, and remove k-mer which only appear once.

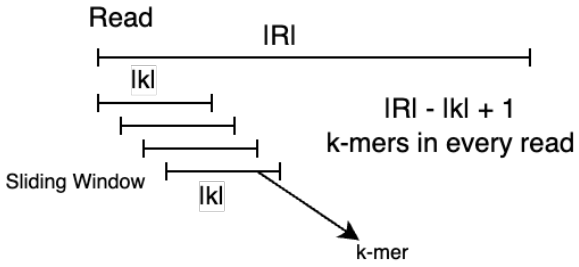

Figure 4: Forming K-mers for every read using sliding window.

Why above process works?

1. **Analogy**: Say we have to spell check a language that we have never seen before. If we see millions of words, and one specific "word" (k-mer) appears only once, while slightly different versions appear many times, the single-occurrence word is likely a typo (error).
2. Since, sequencing errors are random and infrequent, they usually creat unique k-mers not found elsewhere in the true sequence or other reads.
3. The high sequencing depth ensures correct k-mers appear many times. Removing k-mers seen only once targets these likely errors while retaining the well-supported, correct k-mers.

Graph: Frequency of the k-mer vs number of k-mer

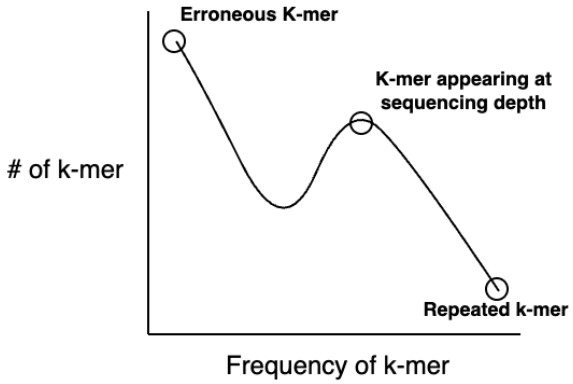

Figure 5: Analysing how often different k-mer occur

We can analyse from Figure 5:

1. **Errorneus k-mer** are a lot (y-axis), but each distinct erroneous k-mer appear only a few times (x-axis).
2. **Repeated k-mer** k-mers which are correct appear a lot of times (x-axis), but the correct k-mers are less in number.

Which data-structure to use to remove erroneous k-mer

Hash Table: It is a good and simple choice but consumes a lot of space.

If we do the calculations – $10B - 20B$ distinct k-mers, each taking 8 byte of space. Hence total space requirement blows upto $8 \times 20B = 8 \times 2^{35}$ for the keys and separately for values as well. This approximately amounts to TB scale space requirement.

Application of Filters: By using Bloom or Quotient filters, we can effectively identify and discard the vast majority of low-frequency, likely erroneous k-mers before needing to store exact counts for all of them. This reduces the memory footprint required for error correction.

For more details on representation of de bruijn graph refer to the paper by Chikhi, Limasset, Jackman [3].

3.3.2 How to assemble

Next we assemble k-mers that are not erroneous into a sequence to get the genomic sequence.

3.3.2.1 Building de bruijn graph .

Consider a genomic sequence

$$AAABBBBA$$

We get 3-mers (reads - sliding window):

$$AAA,\ AAB,\ ABB,\ BBB,\ BBB,\ BBA$$

Left/Right (prefix/suffixes) 2-mers from the 3-mers:

$$AA,\ AA,\ AA,\ AB,\ AB,\ BB,\ BB,\ ...$$

Next, we form a graph, with nodes being distinct 2-mers, and edges being the 3-mers they form by connecting two 2-mers.

3.3.2.2 Constructing the original genomic sequence from graph .

Definition: de Bruijn graph is a directed multigraph, i.e it consists of set of vertices V and multiset of directed edges E.

Idea: A walk of the graph, crossing each edge exactly once, gives us a reconstruction of the genome. This is an **eulerian walk**.

Eulerian Walk Definitions:

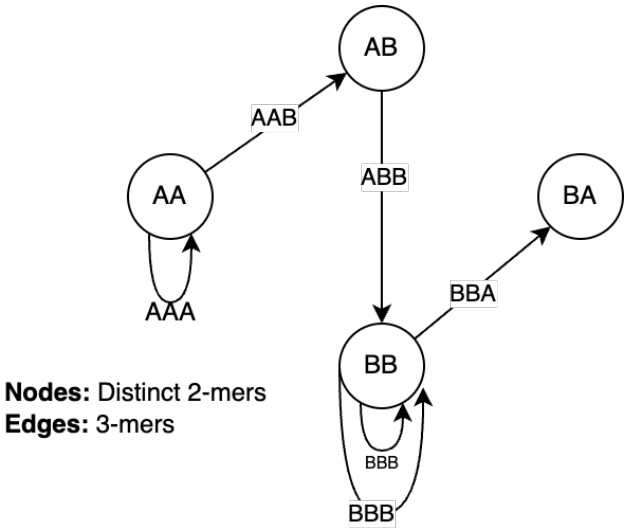

Figure 6: de Bruijn graph of the genomic sequence: AAABBBBA, with k=3

- **Node is balanced**: if $indegree = outdegree$
- **Node is semi-balanced**: if $indegree$ differs from $outdegree$ by 1 or vice-versa.
- **Graph is connected** if each node can be reached by some other node.
- **Eulerian Walk** visits each edge exacty once.
- Not all graphs have eulerian walk, graphs that do are eulerian.
- **A directed connected graph is eulerian** if and only if it has **at most two semi-balanced nodes** and **all other nodes are balanced**.

The last point in the above list is important, as the construction process of de bruijn graphs yields an eulerian graph.

Why?

For finding the genomic sequence within the graph, we need to find a starting point, from where we can start our walk.

1. This means, the left end of the sequence (**start**), i.e Node for k-1 mer is semi-balanced i.e has one more outgoing edge than incoming.
2. Similarly, the right end of the sequence (**end**), is also a semi-balanced with one more incoming edge than outgoing. (Special case: if start and end k-1 mer is same, then this is not true).
3. Other nodes are balanced, since number of times k-1 mer occurs as left k-1 mer is equal to number of times it occurs as right k-1 mer.

3.3.2.3 Refining .

Now we shall improving the quality and accuracy of the assembled genome by addressing errors and resolving ambiguities in the graph structure.

There are 3 types of ambiguities still be present in the graph structure after removing the error through the error correction pipeline.

1. Island: Does not connect to the longer sequence at all.
2. Tip: Does not connect back to the longer sequence.
3. Bubbles: Sequence splits into two paths but they merge later in the sequence.

These are illustrated in Figure 7.

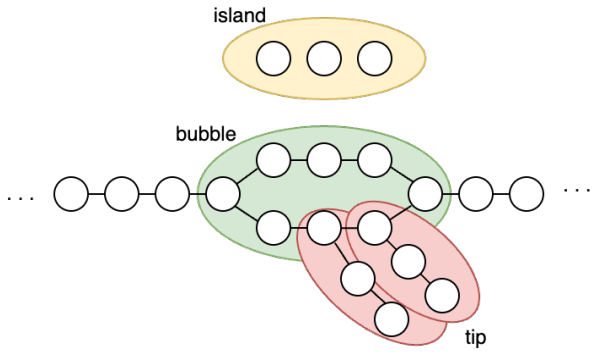

Figure 7: Ambiguities present in de bruijn graph of k-mers

How to resolve?

Tips and islands are easy to resolve using trivial graph techniques. For bubbles we can utilize the fact that graph we form is weighted, i.e we have the count of each k-mer (edge) as well. We can use this heuristic to get the correct path in the bubble.

3.4 OLC - Overlap Layout Consenus

Refer to lecture notes for a detailed write up on this algorithm. These notes cover this topic briefly.

What is an **Overlap Graph**: It is a directed graph, where nodes represent reads and edges represent overlaps between them. It is constructed by calculating overlap between each pair of read. By using overlap we can deduce that they might be connected in the actual genome sequence.

References

[1] Phillip E. Compeau, Pavel A. Pevzner, Glenn Tesler. How to apply de Bruijn graphs to genome assembly. *Nature Biotechnology*, 29(11):987–991, 2011.

[2] Heng Li, Xianchang Feng, Chengsheng Chu. The design and construction of reference pangenome graphs with minigraph. *Genome Biology*, 21:265, 2020.

[3] Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T. Simpson, Paul Medvedev. On the representation of de Bruijn graphs. *Journal of Computational Biology*, 22(5):336–352, 2015.