A Review of Genome Graph Tools

Manuscript

Njagi Mwaniki and you if you wanna ;)

Contents

1 Introduction **1**

2 Review of tools **2**

2.1 Alignment/Mapping and Assembly 2

2.1.1 Indexing 2

2.1.2 Assembly 2

2.2 Interfaces and APIs 3

2.3 Data Interchange 3

2.3.1 FASTG 3

2.3.2 GFA 4

2.3.3 rGFA (reference GFA) 4

3 Discussion **4**

4 Appendix **5**

4.1 Formats 5

4.2 Algorithms 5

1 Introduction

Most kinds of evolution are based on changes in the DNA of an organism. In fact, change is so inherent in DNA in that it even happens in everyday cell division (somatic cells) making it impossible to separate variation from genomes. This means that not only is the DNA of each species different from the other but that of each individual is different.

Despite this, through the use of a linear reference genome, most methods, up until recently treated DNA within a species as if it were static across time or updating at regular intervals, homogenous between individuals, and not polymorphic.

This led to among other problems the reference bias problem<citation needed (cn)> as well as the reference not being a functional genome<cn> meaning that if we could somehow get it into a cell that the cell would actually survive and replicate.

In this way, the reference genome is an illusion that works but as (individualized) medicine, vaccine research and other fields require finer-grained results from genetics it begins to show its flaws.

To study DNA in a way that factors in changes that occur across time and between individuals we need a structure that exposes the changes(variation) and collapses redundancies. On top of this, it has to support all the operations possible with linear methods such as assembly, querying, and visualization in a space-efficient and time-efficient manner.

The natural alternative is a graphical reference genome<citation needed>. These represent genomes through a series of nodes and edges as a means of expressing the variation. However, their adoption isn't as widespread because of the computational complexity as well as the novelty of the approach. There are four major overlapping bioinformatics steps that need to be carried out after sequencing(reading the DNA strand) to getting the actionable results: Assembly: recreating the original genetic sequence Alignment/Mapping/indexing/querying: can be part of the assembly or can be done later to query the purpose of a section of DNA Visualization: A way to see these changes

2 Review of tools

2.1 Alignment/Mapping and Assembly

The alignment problem involves finding where a substring lies within a larger string either exactly or approximately (fuzzy) within a larger one.

Sequence alignment started with Needleman and Wunsch (1970) followed by local alignment (Smith and Waterman, 1981), where the alignment can be between any substrings of the two sequences then semi-global alignment (Sellers, 1980) where one sequence (query) is entirely aligned to a substring of the other (reference).

Alignment problems grow with the input size (what is the computational complexity of it?) therefore in the case of graphs, the alignment problem gets rather complicated for variant calling. In this case, there was the development of the GBWT a Burrows-Wheeler Transform generalized for graphs that... used in seqwish allowing it to be orders of magnitude faster than VG.

2.1.1 Indexing

Indexing involves coming up with a way to reduce the search space so that searching can be faster. The building of an index is, however, a non-trivial process. Vg which performed a BWT and used the FM-index takes... to index the human genome. The indexing problem is non-trivial in genomes generally but gets even more complicated when it comes to graphs. Indexes like the FM-Index and BWT fail to hold and there's the need for indexes such as the BWT used in seqwish allowing it to be orders of magnitude faster than VG.

2.1.2 Assembly

The process of assembly involves finding overlapping sections of a sequence and linking them somehow. Most of the tools that do this use a graph of some kind.

The most common kind is the de Bruijn graph. A DBG <describe a DBG here> break sequence into k-mers and find overlaps These tools

Minia (2013) used a de Bruijn graph and a bloom filter (for quick search) to come up with a probabilistic de Bruijn graph is obtained by inserting all the nodes of a de Bruijn graph (i.e all k-mers) in a Bloom filter. Go into how a bloom filter gets rid of false positives?

Bcalm2 (2016) built a de Bruijn graph but in low memory allowing assembly of genomes of a laptop

Bifrost (2019) brings in parallelism (can take advantage of multiple cores) and graph coloring (associating of the nodes with metadata such as where they come from) on a cDBG.

GraphAligner (2019) Use a bit parallel shift and algorithm

bit-parallelism (Rautiainen et al., 2019) studies the semi-global sequence-to-graph alignment problem. That is, we seek to find a path in a directed, node-labeled graph that has minimum edit distance to the query sequence.

Shift-And algorithms (Baeza-Yates and Gonnet, 1992; Domolki, 1964, 1968) for exact string matching to graphs Myers' bit-vector alignment algorithm (Myers, 1999) to graphs, which proceeds along the same lines as the Shift-And algorithm, but requires some further algorithmic insights to handle nodes with an in-degree greater than one

SPAdes (different tools) This is done in the assembly stage and the variant calling stage.

Odgi

Reads Practical Graphical Pangenomics

2.2 Interfaces and APIs

The field of genome graphs is growing at an alarming rate as evidenced by the ever-growing number of tools. There is, therefore, a need to have a common way of how the tools interact with the data they operate on.

Libhandlegraph, which has python bindings and now being ported to Rust, is a declarative approach towards graphs that defines an interface between which tools interact with the data below. The ides is to treat the graph as a larger structure to which we have pointers to called handles (similar to Unix file handles) through which we manipulate the graph. In C++ and Python, this is done using the class abstraction while in Rust the trait abstraction is used. This defines a common set of attributes and operations through which we can manipulate the graph.

We can then use libhandlegraph as a layer between an underlying graph implementation and genome graph manipulation tools we plan on building.

Tools built on top of this are PackedGraph (low memory) and HashGraph (high-performance hash tables)

2.3 Data Interchange

The aim here is to come up with a widely adopted plaintext specification for presenting graphs be they reference genome graphs or assembly graphs.

DNA data can be serialized into binary formats by tools such as VG but they're most useful for interchange. For this, there are plain text formats. The de facto plaint text format for DNA data interchange is FASTQ for raw sequences and FASTA for assembled sequences.

The world of assembly has some end to end graph assembly tools but the need for tools that specialize in different parts of the assembly process are needed.

Contemporary tools store and exchange DNA data in FASTA (or FASTQ for unassembled sequences) files which represent a linear string and variation is mainly represented in VCF. These formats cannot hold for the interchange of graphs.

A few attempts have been made for a number of formats beginning with FASTG (a play on FASTA) all way to GFA.

We then need different ways of transferring the results of tools between each other for the building of pipelines. There are different ways of doing this. There are binary formats of and text-based formats. The text-based ones are

In the view of graphical references being a step up from linear references (or if the graph was built from VCF or incrementally), we may want to associate each node with the linear genome from which it came.

Extension to GFA with 3 additional tags that indicate the origin of a segment

2.3.1 FASTG

Based on a sequence digraph.

"The traditional assembly representation had two mechanisms for representing branching and uncertainty: (1) ambiguous base codes, according to the IUPAC scheme, and (2) base quality scores (via an ancillary quality score file, in one--to--one correspondence with the FASTAfile). Both are limited in their expressive capability, and while (1) is exceptionally straightforward and easy to understand, the interpretation of (2) is less clear, particularly in regard to indels. Finding an appropriate replacement will involve a balance between expressive capability and complexity."

A format for representing genome assemblies. It's like FASTA but the G stands for Graph.

An assembly format prior to GFA. Vertex means edge Edge means adjacency

```
>x:y;
ACGTGAGAT
```

Here x represents a DNA sequence and an edge in the graph. The edge is in turn followed by edge y. There exists an adjacency from edge x to edge y

To facilitate inversions the format allows for adjacencies between forward and reverse complement edges by use of the prime character eiej' follow ei either follow the reverse complement of ej

The header line >Edge:Neighbours:Properties; Where Edge is the name given to this edge/sequence Neighbors is a list of edges (or their reverse complements, indicated by a ') that follow this edge or the reverse complement of this edge(indicated by a preceding~) Properties is a list of optional properties associated with this edge (discussed later in this document)

FASTG puts sequences on arcs/edges. It is unable to describe a simple topology such as A->B; C->B; C->D without adding a dummy node, which breaks the theoretical elegance of assembly graphs. <heng li's blog>

Reads FASTG spec ASQG `https://github.com/jts/sga/wiki/ASQG-Format`

2.3.2 GFA

Able to represent a graph at all stages in the assembly Similar to ASQG Consists of four types of lines Header (H) header Segment (S) segment or node Link (L) bidirected edge Contained (C) containment

Spec `https://gfa-spec.github.io/GFA-spec/` Describes sequence graphs

Reads The design and construction of reference pangenome graphs Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences Dear assemblers, we need to talk ... together by Pall Melsted The Graphical Fragment Assembly (GFA) format in acgt.me A proposal of the Graphical Fragment Assembly format

2.3.3 rGFA (reference GFA)

GFA extended for reference (pan)genomes An extension to GFA with 3 additional tags that indicate the origin of the segment to give us a unique stable coordinate system as an extension to the linear reference coordinate

Each segment is associated with one origin Which forbids collapsing of different nodes from one region as would be with a cDBG in the graph by design

rGFA disallows overlaps between edges and forbids multiple edges (more than one edges between the same pair of vertices). Reads The design and construction of reference pangenome graphs Mapping GAF An extension to Pairwise mApping Format

Reads The design and construction of reference pangenome graphs Visualization Bandage (2015) Uses a force-directed layout, strength is aesthetic appeal and clearly communicating components but annotation and

navigation aren't possible. The major issue is the runtime scalability Force-directed layout has quadratic or even cubic costs with respect to graph size <cite pantograph docs>.

Takes in a variety of formats LastGraph (Velvet) FASTG (SPAdes) Trinity.fasta ASQG GFA

Good for a cursory visualization of the graph Standalone MoMI-G MOdular Multi-scale Integrated Genome graph browser Built for the visualization of structural variants (SVs) as a variation graph Has a desktop version & web component Pantograph A web browser for the visualization of graphs

Reads: MoMI-G: modular multi-scale integrated genome graph browser Bandage: interactive visualization of de novo genome assemblies

3 Discussion

Because viruses are small and don't have chromosomes we recommend the use of colored cDBG for the initial graph assembly.

4 Appendix

4.1 Formats

- cDBG compacted de Bruijn Graph
- rGFA reference
- PAF Pairwise mApping Format
- GAF Graphical mApping Format
- ASQG

4.2 Algorithms

- BWT
- GBWT