Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Genome Analysis

Genome Analysis

ODGI: understanding pangenome graphs

Andrea Guarracino [1†], Simon Heumos [2†], Sven Nahnsen [2,3], Pjotr Prins [4], and Erik Garrison [4*]

[1]Genomics Research Centre, Human Technopole, Milan, Italy
[2]Quantitative Biology Center (QBiC), University of Tübingen, Tübingen, Germany, 72076
[3]Biomedical Data Science, Dept. of Computer Science, University of Tübingen, Tübingen, Germany, 72076
[4]University of Tennessee Health Science Center, Memphis, TN, USA

[*]To whom correspondence should be addressed.
[†]Contributed equally.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: Pangenome graphs provide a complete representation of the mutual alignment of collections of genomes. These models offer the opportunity to study the entire genomic diversity of a population, including structurally complex regions. Nevertheless, analyzing hundreds of gigabase-scale genomes using pangenome graphs is difficult as it is not well-supported by existing tools. Hence, fast and versatile software is required to ask advanced questions to such data in an efficient way.
Results: We wrote ODGI, a novel suite of tools that implements scalable algorithms and has an efficient in-memory representation of DNA variation graphs. ODGI includes tools for detecting complex regions, extracting *loci*, removing artifacts, exploratory analysis, manipulation, validation, and visualization. Its fast parallel execution facilitates routine pangenomic tasks, as well as pipelines that can quickly answer complex biological questions of gigabase-scale pangenome graphs.
Availability: ODGI is published as free software under the MIT open source license. Source code can be downloaded from `https://github.com/pangenome/odgi` and documentation is available at `https://odgi.readthedocs.io`. ODGI can be installed via Bioconda `https://bioconda.github.io/recipes/odgi/README.html` or GNU Guix `https://github.com/ekg/guix-genomics/blob/master/odgi.scm`.
Contact: egarris5@uthsc.edu

1 Introduction

A pangenome models the full set of genomic elements in a given species or clade (Consortium, 2018; Eizenga *et al.*, 2020b). In contrast to reference-based approaches which relate sequences to a single genome, these data structures encode the mutual relationships between all the genomes represented. In pangenome graphs (Paten *et al.*, 2017), homologous regions between genomes are compressed into a single representative of all alleles present in the pangenome. These flexible models let us encode any kind of variation, allowing the generation of comprehensive data systems that builds the basis for the analyses of genome evolution. Although these data structures are of utility to researchers (Consortium, 2018; Garrison *et al.*, 2018; Baaijens *et al.*, 2019; Hickey *et al.*, 2020; Sibbesen *et al.*, 2021), the scientific community still lacks a toolset specifically focused on graph manipulation and interrogation.

The Human Pangenome Reference Consortium (HPRC) and Telomere-to-Telomere (T2T) consortium (Miga *et al.*, 2020; Logsdon *et al.*, 2021; Nurk *et al.*, 2021) have recently demonstrated that high-quality *de novo* assemblies can be routinely generated from third-generation long read sequencing data. We anticipate that *de novo* assemblies of similar quality will become common, leading to demand for methods that allow us to create and explore pangenomes.

Here, we present the Optimized Dynamic Genome/Graph Implementation (ODGI) toolkit, a pangenome graph interrogation and transformation system specifically implemented to handle the data scales encountered when working with pangenomes built from hundreds of haplotype-resolved genomes. ODGI provides a set of standard operations on the variation graph data model, generalizing “genome arithmetic” concepts like those found in BEDTools (Quinlan and Hall, 2010) to work on pangenome graphs, and providing a variety of operations, such as visualization, sorting, and liftover projections, all critical to understand and exploit pangenome graphs.

2 Model

A pangenome graph is a sequence model that encodes the mutual alignment of many genomes (Garrison, 2019; Eizenga *et al.*, 2020b). In the variation graph, $V = (N, E, P)$, nodes $N = n_1 \ldots n_{|N|}$ contain sequences of DNA. Each node n_i has an identifier i and an implicit reverse complement $\bar{n_i}$, and a node strand s corresponds to one such orientations. Edges $E = e_1 \ldots e_{|E|}$ represent ordered pairs of node strands: $e_i = (s_a, s_b)$. Paths $P = p_1 \ldots p_{|P|}$ describe walks over node strands: $p_i = s_1 \ldots s_{|p_i|}$. When used as a pangenome graph, V expresses sequences, haplotypes, contigs, and annotations as paths. By containing both the sequences and information about their relative variations, the variation graph provides a complete and powerful foundation for many bioinformatic applications.

Pangenome graphs can be constructed by multiple sequence alignment (Lee *et al.*, 2002; Grasso and Lee, 2004) or by transitively reducing an alignment between sequences to an equivalent, labeled sequence graph (Kehr *et al.*, 2014; Garrison, 2019). Current methods to build these graphs are still under active development (Li *et al.*, 2020; Armstrong *et al.*, 2020; Garrison *et al.*, 2021), but they have largely settled on a common data model, represented in the Graphical Fragment Assembly (GFA) format (GFA Working Group, 2016). This standardization supports the development of a reference set of tools that operate on the pangenome graph model. Such an effort began with the VG toolkit (Garrison *et al.*, 2018). Here we refocus it with ODGI, a compatible, but independent set of algorithms focused on visualization, interrogation, and transformation of pangenome graphs.

3 Implementation

The ODGI toolkit builds on existing approaches to efficiently store and manipulate variation graphs (Garrison *et al.*, 2018). Similar to other efficient libraries presenting the HandleGraph model (Eizenga *et al.*, 2020a), the implementation of ODGI's tools rests on three key properties which hold for most pangenome variation graphs:

1. They are relatively sparse, with low average node degree.
2. They can be sorted so that most edges go between nodes that are close together in the sort order.
3. Their embedded paths are locally similar to each other.

These properties are used to build efficient dynamic variation graph data structures (Siren *et al.*, 2020; Eizenga *et al.*, 2020a). Sparsity (1) allows us to encode edges E using adjacency lists rather than matrices or hash tables. The local linear structure of the graph (2) lets us assign node identifiers that increase along the linear components of the graph, which supports a compact storage of edges and path steps as relativistic (usually small) differences rather than absolute (always large) integer identifiers. Path similarity (3) allows us to write local compressors that reduce the storage cost of collections of path steps.

ODGI improves on prior efforts, based on issues that arose during our work with high-quality *de novo* assemblies that cover almost all parts of the human genome (Logsdon *et al.*, 2021; Nurk *et al.*, 2021). In particular, we find that it is necessary to support graphs with regions of very high numbers of path traversals (high path depth). Such motifs can occur in collapsed structures generated by ambiguous sequence homology relationships in repeats found in the centromeres and other segmental duplications. If we cannot process such regions, there are only two options: 1) remove such regions, or 2) leave them unaligned. However, neither of these solutions allows us to investigate their biological features. To seamlessly represent such difficult regions, we followed an approach implemented in the dynamic version of the Graph BWT (GBWT) (Siren *et al.*, 2020) and built a node-centric, dynamic, compressed model of the paths. This design supports node-local modification and update of the graph, which lets us operate on paths in parallel.

We store the graph in a vector of node structures, each of which presents a node-local view of the graph sequence, topology, and path layout. Expressed in terms of the variation graph V, ODGI's core $Node$ structure includes a decoder that maps the neighbors of each node to a dense range of integers. For a given $Node_i$ and neighbor $Node_j$, the decoder itself does not store the id of $Node_j$, but rather a compact representation of the relative difference between the node ids: $\delta = Node_i.id - Node_j.id$. This keeps the size of the encoding small, per common variation graph property (2). We define the edges and path steps traversing the node in terms of this alphabet of δ's. The structures in Algorithm 1 describes our encoding.

```
Struct *Node* contains
    **id** $\in \mathbb{N}$ // an identifier
    **lock** // atomic locking primitive
    **sequence** = [A|T|G|C|N]$*$ // DNA
    // bit-packed vector of edges
    **edges** = $(x_i, x_j)* : (i, j) \in [1 \ldots \Sigma]^2$
    // bit-packed vector of id deltas
    **decoding** $x_1 \ldots x_\Sigma \in \mathbb{N}^\Sigma$
    // bit-packed vector of path steps
    **path_steps** $[Step_1 \ldots Step_n]*$
end
Struct *Step* contains
    **path_id** $\in \mathbb{N}$ // the path's global id
    **is_rev** $\in (0, 1)$ // the step orientation
    **is_start** $\in (0, 1)$ // if first step in path
    **is_end** $\in (0, 1)$ // if last step in path
    **prev_**$\delta \in [1 \ldots \Sigma]$ // $\delta$-encoded previous node
    **prev_rank** $\in \mathbb{N}$ // step rank on previous node
    **next_**$\delta \in [1 \ldots \Sigma]$ // $\delta$-encoded previous node
    **next_rank** $\in \mathbb{N}$ // step rank on next node
end
```
Algorithm 1: ODGI's relativistically-packed $Node$ structure and the $Step$ structure used to represent the paths as doubly-linked lists.

Each structure contains the sequence of the node ($Node_i.sequence$), its edges in both directions ($Node_i.edges$), and a vector of path steps that describes the previous and next steps in paths that walk across the node ($Node_i.path\_steps$). For efficiency, $Node_i.sequence$ is stored as a plain string, while the $edges$ and $path\_steps$ are stored using a dynamic succinct integer vector that requires $O(2nw)$ bits for the edges and $O(5nw)$ bits for the path steps, where n is the number of steps on the node and w is $\approx log_2(n)$ (Prezza, 2017).

To allow edit operations in parallel, each node structure includes a byte-width mutex $lock$. All changes on the graph can involve at most two $Node$ structs at a time (both edge and path step representations are doubly-linked). To avoid deadlocks, we acquire the node locks in ascending $Node.id$ order and release them in descending order. In addition to node-local features of the graph, we must maintain some global information. Specifically, we record the start and end of paths, as well as a name to path id mapping in lock-free hash tables. The use of lock-free hash tables lets us avoid a global lock when looking up path or graph metadata, which would quickly become a bottleneck during parallel operations on the graph. By avoiding global locks, we implement many of the operations in ODGI using maximum parallelism available. This approach is key to enable our methods to scale to the largest pangenome graphs that we can currently build (with hundreds of vertebrate genomes).

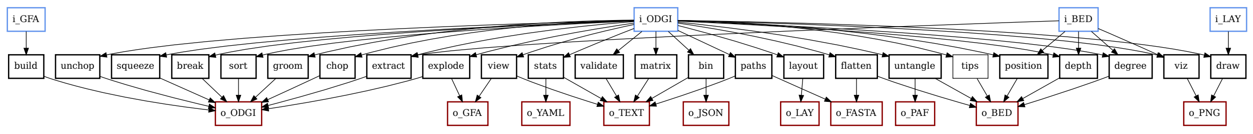


Fig. 1: Methods provided by ODGI (in black) and their supported input (in blue) and output (in red) data formats.

4 Results

ODGI provides a set of interrogative and manipulative operations on pangenome graphs. We have established these tools to support our exploration of graphs built from hundreds of large eukaryotic genomes. ODGI's tools are practical and able to work with high levels of graph complexity, even with regions where paths present very high depth nodes (10^5 to 10^6-fold depth).

ODGI covers common operations that we have found to be essential when working with complex pangenome graphs:

- *odgi build* constructs the ODGI data model from GFA file (§4.1).
- *odgi view* converts the ODGI data model into GFA file (§4.1).
- *odgi viz* provides a linear visualization of the graph (§4.2).
- *odgi draw* renders a 2D image of the graph (§4.2).
- *odgi extract* excerpts subsets of the graph based on path ranges (§4.3).
- *odgi explode* breaks the graph into connected components (§4.3).
- *odgi squeeze* unifies disjoint graphs (§4.3).
- *odgi chop* breaks long nodes into shorter ones (§4.4).
- *odgi unchop* combines unitig nodes (§4.4).
- *odgi break* removes cycles in the graph (§4.4).
- *odgi prune* removes complex regions (§4.4).
- *odgi groom* resolves spurious inverting links (§4.4).
- *odgi position* lifts coordinates between path and graph positions (§4.5).
- *odgi untangle* deconvolutes paths relative to a reference (§4.5).
- *odgi tips* finds path end points relative to a reference (§4.5).
- *odgi sort* orders the graph nodes (§4.6).
- *odgi layout* establishes a 2D layout (§4.6).
- *odgi matrix* derives the pangenome matrix (§4.7).
- *odgi paths* lists and extracts paths in FASTA (§4.7).
- *odgi flatten* converts the graph to FASTA and BED (§4.7).
- *odgi stats* provides numerical properties of the graph (§4.7).
- *odgi bin* generates a summarized view of the graph (§4.7).
- *odgi depth* describes node depth over graph and path positions (§4.7).
- *odgi degree* describes node degree over graph and path positions (§4.7).

Each tool focuses on a small set of related operations. Most read or write the native ODGI format ('og' extension) (Figure 1) and work with standard text based data formats common to bioinformatics. This supports the implementation of flexible and composable graph processing pipelines based on graphs (GFA/ODGI) and standard bioinformatic data types representing positions, genomic ranges (BED), and pairwise mappings (PAF). We use variation graph paths to provide a universal coordinate system, representing annotations and pairwise sequence relationships using the paths as reference and query sequences. Thus, ODGI provides a set of interfaces that let us approach these graphs from the perspective of standard reference- and sequence-based data models. Indeed, by considering all paths in the graph as potential reference or query sequence, we make graphs invisible to downstream tools that operate on collections of sequences or rely on a reference sequence (*e.g.* SAMtools (Li *et al.*, 2009)), enabling interoperability. This approach benefits from the information in the graph without strongly embedding our methods in this difficult new research context.

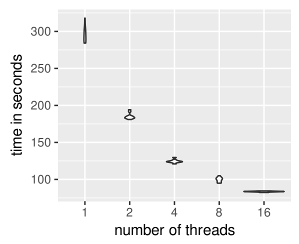
4.1 Building the ODGI model

ODGI maintains its own efficient binary format for storing graphs on disk. We begin by transforming the storage model of the standard GFAv1 (GFA Working Group, 2016) format (in which nodes, edges, and paths are described independently) into the ODGI node-centric encoding with *odgi build*. This construction step can be a significant bottleneck, in particular as the size of the path set of the graph increases.

The ODGI data structure (Algorithm 1) allows algorithms that build and modify the graph to operate in parallel, without any global locks. In *odgi build*, we initially construct the node vector in a serial operation that scans across the input GFA file. Then, we serially add edges in the $Node.edges$ vectors of pairs of nodes. Finally, we create paths in serial, and extend them in parallel by obtaining the mutex $Node.lock$ for pairs of nodes and by adding the path step in their $Node.path\_steps$ vectors. This parallelism speeds ODGI model construction by many-fold when testing against graphs made from assemblies produced by the HPRC (Figure 2).

To support interchange with other pangenome tools or text-based processing, *odgi view* converts a graph in ODGI binary format to GFAv1.

Fig. 2: Performance evaluation of *odgi build* when translating a 90-haplotype graph of human chromosome 6 into ODGI's native format. Build time decreases as parallelism increases, but with diminishing per-thread processing rates.



4.2 Visualizing pangenome graphs

Pangenome graph visualization is one of the first steps to gain insight into the mutual relationship between the sequences in the graph and their variation. We pursue a novel approach to visualization with *odgi viz* and *odgi draw*, two tools which provide scalable ways of generating pictures of the high-level structure of large pangenome graphs.

odgi viz supports a binned, linearized rendering in 1 dimension (1D) (that is, all graph nodes lie on the same axis). This visualization is computed in linear-time and offers a human-interpretable format suitable for understanding the topology and genome relationships in the pangenome graph (Fig. 4). Graph nodes are arranged on a single axis, from left to right, with the colored bar indicating the paths and the nodes they cross. White spaces indicate where paths do not traverse the nodes. The meaning of the colors depends on how *odgi viz* is executed. By default, path colors are derived from a hash of the path name (Fig. 4**b**). Path names are displayed on the left of the paths. The black lines on the bottom indicate the edges connecting the nodes and, therefore, represent the graph topology.

Nevertheless, complex, nonlinear graph structures are difficult to display and interpret in a low number of dimensions. To overcome such a limitation, *odgi viz* supports multiple visualization modalities (Fig. 4**c**-**e**), making it easy to grasp the properties and shape of the graph. Graph node order can affect downstream analyses on pangenome graphs. With *odgi viz* we can color the paths by path position (Fig. 4**c**), with light grey indicating where paths begin and dark grey where they end. This visualization is suitable for understanding graph node order, as smooth color gradients indicate that the graph order respects the linear paths' coordinate systems. Pangenome graphs represent both strands of DNA sequences. *odgi viz* supports also coloring the paths by orientation, with paths colored where their sequence is reverse-complemented (red) or in direct orientation (black) with respect to the sequences of the graph nodes (Fig. 4**d**). Eukaryotic genomes experience gains and losses of genetic material, resulting in copy number variation (CNV) across the population. With *odgi viz*, we can use multiple color palettes to color the paths by path depth, highlighting the different copy number statuses in the genomes represented in the pangenome graph (Fig. 4**e**).

odgi draw extends the visualization in 2 dimensions (2D) (Fig. 4**a**) by rendering the layout built by *odgi layout* (§4.6). A 2D rendering is more costly to compute, but we similarly provide an implementation that scales linearly with pangenome sequence size, allowing us to apply it to large pangenome graphs.

4.3 Extracting or joining regions of interest

Pangenome graphs built from hundreds of haplotype-resolved *de novo* genome assemblies are very large, but it is often necessary to work with only a small portion of the genomes represented, such as a specific *locus* (Fig. 4**a**) or a smaller region (Fig. 4**b-g**), or even a single gene (Fig. 3). This simplifies the downstream analyses and reduces the resources to work only with the extracted graphs. Graph portions can be extracted by using the paths in the graph as coordinate systems to guide the process. For such operation, *odgi extract* allows users to extract specific regions of the graph as defined by query criteria. Regions of interest can be specified by graph nodes or path range(s), also in BED format. Furthermore, it is possible to indicate a list of paths to be preserved completely in the extracted graph.

In *odgi extract*, we begin by collecting all graph nodes that fall within the ranges to extract (and the paths to preserve, if requested). Users can specify the number of steps or nucleotides to expand the selection and include neighboring nodes. Then, edges connecting all selected nodes are added in the subgraph under construction. Finally, the portions of the paths (*i.e.*, the subpaths) walking through the selected nodes are extracted and added to the new subgraph. Subpaths are searched in parallel, created serially, and extended in parallel again thanks to the parallelism enabled by the ODGI data structure (see §4.1), making *odgi extract* a scalable solution to extract also complex subregions presenting nodes with very high path depth.

Pangenome graphs can embed multiple chromosomes as separated connected components (inter-chromosomal structural variants would join the components into bigger ones). *odgi explode* separates the connected components in different ODGI format files, while *odgi squeeze* allows merging multiple graphs into the same ODGI format file, preventing node ID collisions.

4.4 Editing the graph structure

Pangenome graphs can be used in a variety of applications, ranging from read mapping to variant identification and genotyping (Eizenga *et al.*, 2020b). However, graphs presenting complex topology can increase the computational overhead of many downstream analyses. ODGI offers multiple commonly-needed basic operations on the topology of the graph and its nodes.

For simplifying the graph structure, users can use *odgi prune* to take away complex parts as defined by query criteria, while with *odgi break* they can remove cycles in the graph, reducing the complexity of the graph topology. Furthermore, *odgi groom* allows removing spurious inverting links by exploring the graph from the orientation supported by most paths.

To enable efficient sequence alignment against the graph, long nodes can be divided into shorter nodes at a maximum requested size using *odgi chop*. Partial order alignment, which consists of aligning sequences against a directed acyclic graph (DAG), is frequently used in pangenome building pipelines (Garrison *et al.*, 2021), but the current implementations return DAGs with 1-bp long nodes. *odgi unchop* allows joining nodes that can be merged without changing the graph topology, nor the embedded sequences, obtaining an equivalent, but more compact, representation of the graph.

4.5 Untangling and navigating the pangenome

The key data in a pangenome graph is a representation of the alignment (or homology) relationships between the sequences represented. Navigating and understanding the graph requires coordinate systems that we can use to link other data into the graph model, and thus to all genomes in the pangenome. ODGI's tools use the embedded sequences to provide a universal coordinate space that is graph-independent, thereby remaining stable across different graphs built with the same genomes.

A universal coordinate system allows us to support several kinds of "lift-over" of coordinates between different genomes in the same or different graphs. *odgi position* translates graph and path positions between or within graphs, emitting the liftovers in BED format. Coordinates can be specified in BED format, but users can even specify a GFF/GTF file to project the annotations into the pangenome graph (Fig. 4**f-g**).

For a precise translation process when conversing a query position to a reference position in a repeat region, we apply the *path jaccard* context mapping concept. It could be that the found reference node is visited several times by the reference. To ensure a precise translation, we select the reference position whose context (the multiset of $Node.id$s reached within a distance of e.g. 10kbp) has the best jaccard metric when compared to the query context.

Pangenome graphs model alignments of many genomes. With *odgi untangle*, users can extract pairwise alignment information between a given set of "query" sequences and a given set of "target" sequences (used as references). While pangenome graphs may contain looping structures that imply many-to-many alignments between the pangenome sequences, these untangled alignments map each segment of the queries to a single segment in the set of targets. Being able to work with any sets of reference sequences lets us convert the graph to lift-over maps compatible with standard software for projecting annotations and alignments from one genome to another. As an example, by untangling the graph we can study the variation that lies in regions collapsed due to ambiguous alignments over sequence repeats (as shown in Fig. 4**f**). Indeed, to obtain a more precise overview of the *locus* in Fig. 4**b-e**, we can apply *odgi untangle* to segment paths into linear segments by breaking these segments where the paths loop back on themselves. We first discover segment boundaries using standard approaches for detecting repeats in sequence graphs (Pevzner, 2004). We finally "untangle" by finding the target segment that best match each query segment using the *path jaccard* context mapping model. In this way, we obtain information on the position and copy number status of the sequences in the collapsed *locus* (Fig. 4**h**). Moreover, to obtain base-level precise information on the relationships between the repeated sequences, we can align them by using the pairs of regions that came from the untangling to guide the alignment (Guarracino *et al.*, 2021).

odgi tips can identify the break point positions of the contigs relative to the reference(s) in the graph by walking from the ends of each contig until a reference node is found. It could be that the reference visits the node several times. Therefore, for each contig range (a tip) *odgi tips* takes a look at each possible reference window and finds the most similar one using the *path jaccard* concept. The output is a BED file with the best reference hit and position for each of the contigs' ends.

4.6 Sorting the pangenome graph

Pangenome graphs can hide their underlying latent structures, introducing difficulties in the analysis and interpretation. Among the causes of this is the correct ordering of the graph nodes in a convenient number of dimensions. ODGI provides a variety of sorting algorithms to find the best graph node order in 1 or 2 dimensions, allowing us to understand the sparse structures typically found in pangenome graphs and the genetic variation they represent.

To find the best order of graph nodes in 1D, *odgi sort* provides multiple sorting algorithms which can be combined into a sorting pipeline to take advantage of the strength of each. Most notably, nodes can be sorted topologically, randomly, by breaking cycles in the graph (§4.4), by grooming (§4.4), and/or by using a novel path-guided (PG) stochastic gradient descent (SGD) algorithm: PG-SGD. This exploits path information to order the graph nodes. PG-SGD learns a 1D or 2D organization of the graph nodes that matches distances in graph paths. To scale to large graphs, we learn this projection in parallel via a HOGWILD! approach (Niu *et al.*, 2011). Our approach can be seen as an adaptation of SGD-based drawing to pangenome graphs (Zheng *et al.*, 2018). In parallel, each HOGWILD! thread updates node relative positions to best-match their nucleotide distance in the paths running through the graph. Following standard SGD approaches, a learning rate is reduced as the algorithm progresses, and execution continues until the adjustments to the model fall below a target threshold ϵ. ODGI can project vector (in 1D) and matrix (2D) representations of the graph relative to these learned coordinate spaces. Based on this projection, we can trivially sort graph nodes in 1D. Moreover, we support the same concept in 2D in *odgi layout* by providing a 2D implementation of the PG-SGD algorithm.

4.7 Obtaining metrics of the pangenome graph

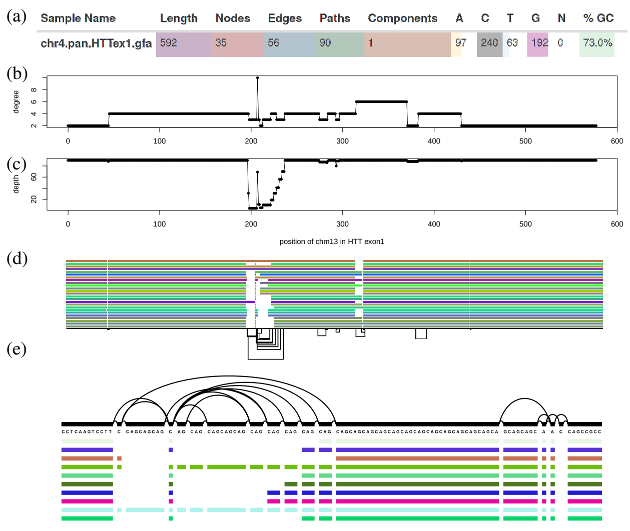
Graphs statistics provide alternative ways to gain insight into pangenomes complexity revealing the overall structure, size, and features of a graph and its sequences.

Pangenome graph topology can be derived by applying *odgi matrix*, obtaining information on graph nodes connections in textual sparse matrix format. To investigate on the genomes encoded in the graph, *odgi paths* allows users to calculate pairwise overlap statistics of groupings of paths and emit all path sequences in FASTA format, and it also allows the generation of a "pangenome matrix" that reports the copy number (presence/absence) of each path over each node. *odgi flatten* generates a linearization of the graph by emitting the pangenome sequence (the concatenation of all node sequences) in FASTA format, and the projection of all paths on the linearized sequence in BED format.

Applying *odgi stats*, users can retrieve metrics describing the graph properties, such as the number of nodes, edges, paths, and graph length. It outputs pangenome statistics in tab-separated values (TSV) or YAML textual file formats. MultiQC's (Ewels *et al.*, 2016) ODGI module provides an interactive way to comparatively explore such statistics of an arbitrary number of graphs.

ODGI also offers more advanced tools for the interrogation of the graphs. To study very large pangenomes, users can use *odgi bin* to summarize the path information into bins of a specified size, generating a summarized view of gigabase scale graphs in TSV or JSON file formats.

Genomes presenting sequences with highly identical repeats result in pangenome graphs with complex motifs that can be detected by *odgi depth* and *odgi degree*, which return the node depth and node degree, respectively, as defined by query criteria. Both tools emit the output in BED format, allowing users to assess the complexity of the graph and detect intricate regions. Indeed, high depth/degree nodes can be the mirror of genetic variation (Fig. 3), but also misassemblies or problems in the pangenome building, making the tools further useful for graph validation.



Sample Name	Length	Nodes	Edges	Paths	Components	A	C	T	G	N	% GC
chr4.pan.HTTex1.gfa	592	35	56	90	1	97	240	63	192	0	73.0%

Fig. 3: Features of a 90-haplotype human pangenome graph of the exon 1 huntingtin gene (*HTTexon1*): **(a)** Excerpt of vital statistics of the *HTTexon1* graph displayed by MultiQC's ODGI module. The very high GC content of 73.0% compared to a human genomic mean GC content of 40.9% (Piovesan *et al.*, 2019) is in accordance with the literature (Neueder *et al.*, 2017). **(b)** Per nucleotide node degree distribution of CHM13 in the *HTTexon1* graph. Around position 200 there is a huge variation in node degree. **(c)** Per nucleotide node depth distribution of CHM13 in the *HTTexon1* graph. The alternating depth around position 200 indicates polymorphic variation complementing the above node degree analysis. **(d)** *odgi viz* visualization of the 23 largest gene alleles, CHM13, and GRCH38 of the *HTTexon1* graph. **(e)** *vg viz* nucleotide-level visualization of 10 gene alleles, CHM13, GRCH38 of the *HTTexon1* graph focusing on the CAG variable repeat region. Figures **(b)-(e)** highlight the variant region around position 200 of CHM13 showing the variable number of glutamine residues of the different individuals as reported by (Nance *et al.*, 1999).

5 Discussion

Pangenome graphs stand to become a ubiquitous model in genomics thanks to their capability to represent any genetic variant without being affected by reference bias (Eizenga *et al.*, 2020b). However, despite this great potential, their spread is impeded by the lack of tools capable of managing and analyzing pangenome graphs easily and efficiently.

ODGI is a state-of-the-art tool suite that enables users to explore and discover the underlying biology in pangenomes graphs, filling the gap that made pangenomic analyses difficult. It provides tools to easily transform, analyze, simplify, validate, and visualize pangenome graphs at large scale. In particular, lifting over annotations and linearizing nested graph structures place the suite as the bridge between traditional linear reference genome analysis and pangenome graphs. With the increased adoption of long read sequencing we expect pangenomic tools to become increasingly common in biomedical research. Particularly for targets that involve complex variation, such as cancer, plant genomics and metagenomics, ODGI will facilitate disentangling, describing and analyzing a much larger set of variation than previously was possible with tools that depend on short reads and reference genomes. Furthermore, users can even consider ODGI as a framework, taking advantage of its algorithms to develop new and more advanced tools that work on pangenome graphs, thus expanding the type of possible pangenomic analyses available to the scientific community.

ODGI is already the backbone of the Pangenome Graph Builder pipeline (Garrison *et al.*, 2021). Its static, large-scale 1D and 2D

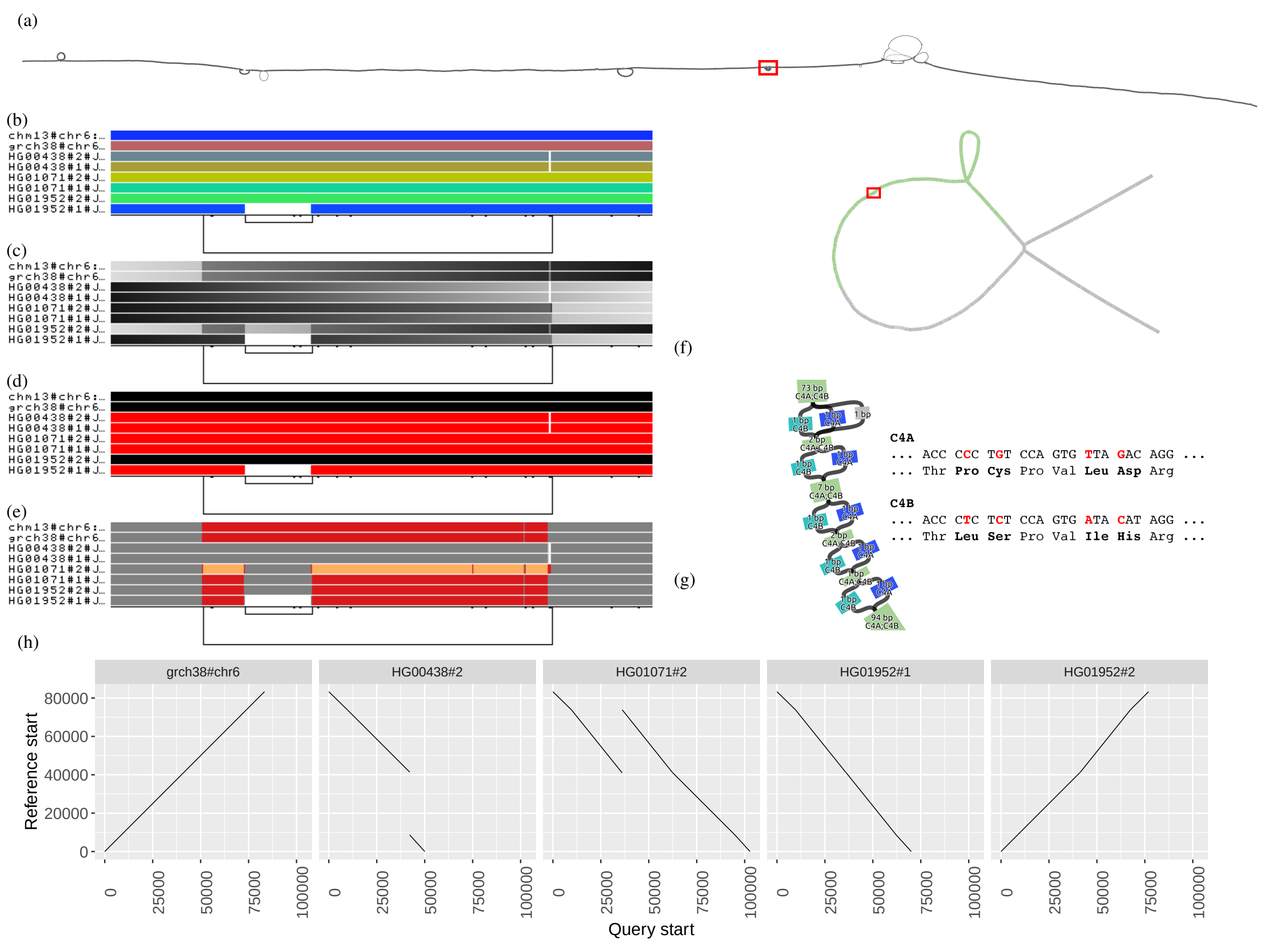


Fig. 4: Visualizing the MHC and C4 pangenome graphs. **(a)** *odgi draw* layout of the MHC pangenome graph extracted from a whole human pangenome graph of 90 haplotypes. The red rectangle highlights the C4 region. **(b-e)** *odgi viz* visualizations of the C4 pangenome graph, where 8 paths are displayed: 2 reference genomes (chm13 and grch38 on the top) and 6 haplotypes of 3 individuals. **(b)** *odgi viz* default modality: the image shows a quite linear graph. The longer links at the bottom indicate the presence of a structural variant (long link) with another structural variant nested inside it (short link on the left). Indeed, human C4 exists as 2 functionally distinct genes, *C4A* and *C4B*, which both vary in structure and copy number (Sekar *et al.*, 2016). The longer link indicates that the copy number status varies across the haplotypes represented in the pangenome. Moreover, *C4A* and *C4B* genes segregate in both long and short genomic forms, distinguished by the presence or absence of a human endogenous retroviral (HERV) sequence, as also highlighted by the short nested link on the left. **(c)** Color by path position. The top two reference genomes and 2 haplotypes (HG01952#2) go from left to right, while 5 haplotypes go in the opposite direction, as indicated by the black color on their left. **(d)** *odgi viz* color by strandness: the red paths indicate the haplotypes that were assembled in reverse with respect to the 2 reference genomes. **(e)** *odgi viz* color by path depth: using the Spectra color palette with 4 level of path depths, white indicates no depth, while grey, red, and yellow indicate depth 1, 2, and greater than or equal to 3, respectively. Coloring by path depth, we can see that the two references present two different allele copies of the C4 genes, both of them including the HERV sequence. The entirely grey paths have one copy of these genes. HG01071#2 presents 3 copies of locus (orange), of which one contains the HERV sequence (gray in the middle of the orange). In HG01952#1, the HERV sequence is absent. **(f)** *Bandage* layout of the C4 pangenome graph, annotated by using *odgi position*. Green nodes indicate the C4 genes. The red rectangle highlights the regions where *C4A* and *C4B* genes differ. **(g)** Annotated *bandage* layout of the C4 region where *C4A* and *C4B* genes differ due to single nucleotide variants leading to changes in the encoded protein sequences. Node labels were annoted by using *odgi position*. **(h)** Visualization of *odgi untangle* output in the C4 pangenome graph: the plots show the copy number status of the sequences in the C4 region with respect to the grch38 reference sequence, making clear, for example, that in HG00438#2, the *C4A* gene is missing.

visualizations of the pangenome graphs allow an unprecedented high-level perspective on variation in pangenomes, and have also been critical in the development of pangenome graph building methods. However, an interactive solution that combines the 1D and 2D layout of a graph with annotation and read mapping information across different zoom levels is still missing. Recent interactive browsers are reference-centric (Beyer *et al.*, 2019; Yokoyama *et al.*, 2019; Durant *et al.*, 2021; Liang and Lonardi, 2021) or focus primarily on 2D (Wick *et al.*, 2015; Gonnella *et al.*, 2018). Our graph sorting and layout algorithms can provide the foundation for future tools of this type. We plan to focus on using these learned models to detect structural variation and assembly errors.

ODGI has allowed us to explore *context mapping* deconvolution of pangenome graph structures via the path jaccard metric. This resolves a major conceptual issue that has strongly guided existing algorithms to construct pangenome graphs. Previously, great efforts have been made to prevent the "collapse" of non-orthologous sequences in the graph topology itself (Li *et al.*, 2020). This has been seen as essential to making these new bioinformatic models interpretable. While our presentation is primarily qualitative, our work demonstrates that we can mitigate this issue by exploiting the pangenome graph not as a static reference, but as a dynamic model of the mutual alignment of many genomes. Because pangenome graphs can contain complete genomes, we are able to query them to polarize the information they contain in easily-interpretable and reusable pairwise formats that are widely supported in bioinformatics. ODGI also projects variation graphs into vector and matrix representations that allow the direct application of machine learning and statistical models to the pangenome. We expect that ODGI will provide a reference interface between pangenomic and genomic approaches to understanding genome variation.

Acknowledgements

We thank members of the HPRC Pangenome Working Group for their insightful discussion and feedback, and members of the HPRC production teams for their development of resources used in our exposition.

Funding

We gratefully acknowledge support from NIH/NIDA U01DA047638 (EG) and NIH/NIGMS R01GM123489 (EG and PP). SH acknowledges funding from the Central Innovation Programme (ZIM) for SMEs of the Federal Ministry for Economic Affairs and Energy of Germany. SN acknowleges Germany's Excellence Strategy (CMFI), EXC-2124 and (iFIT) - EXC 2180 – 390900677. This work was supported by the BMBF-funded de.NBI Cloud within the German Network for Bioinformatics Infrastructure (de.NBI) (031A537B, 031A533A, 031A538A, 031A533B, 031A535A, 031A537C, 031A534A, 031A532B).

Data availability

Code and links to data resources used to build this manuscript and its figures, can be found in the paper's public repository: `https://github.com/pangenome/odgi-paper`.

References

Armstrong, J. *et al.* (2020). Progressive Cactus is a Multiple-Genome Aligner for the Thousand-Genome Era. *Nature*, **587**(7833), 246–251.

Baaijens, J. A. *et al.* (2019). Full-length de novo viral quasispecies assembly through variation graph construction. *Bioinformatics*, **35**(24), 5086–5094.

Beyer, W. *et al.* (2019). Sequence tube maps: making graph genomes intuitive to commuters. *Bioinformatics*, **35**(24), 5318–5320.

Consortium, C. P. (2018). Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, **19**(1), 118–135.

Durant, E. *et al.* (2021). Panache: a web browser-based viewer for linearized pangenomes. *Bioinformatics*.

Eizenga, J. M. *et al.* (2020a). Efficient dynamic variation graphs. *Bioinformatics*, **36**(21), 5139–5144.

Eizenga, J. M. *et al.* (2020b). Pangenome graphs. *Annual Review of Genomics and Human Genetics*, **21**(1), 139–162.

Ewels, P. *et al.* (2016). MultiQC: summarize analysis results for multiple tools and samples in a single report. *Bioinformatics*, **32**(19), 3047–3048.

Garrison, E. (2019). Graphical pangenomics.

Garrison, E. *et al.* (2018). Variation Graph Toolkit Improves Read Mapping by Representing Genetic Variation in the Reference. *Nature Biotechnology*, **36**(9), 875–879.

Garrison, E. *et al.* (2021). The pangenome graph builder. `https://github.com/pangenome/pggb`.

GFA Working Group (2016). Graphical fragment assembly (gfa) format specification. `https://github.com/GFA-spec/GFA-spec`.

Gonnella, G. *et al.* (2018). GfaViz: flexible and interactive visualization of GFA sequence graphs. *Bioinformatics*, **35**(16), 2853–2855.

Grasso, C. and Lee, C. (2004). Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems. *Bioinformatics*, **20**(10), 1546–1556.

Guarracino, A. *et al.* (2021). wfmash: whole-chromosome pairwise alignment using the hierarchical wavefront algorithm. `https://github.com/ekg/wfmash`.

Hickey, G. *et al.* (2020). Genotyping Structural Variants in Pangenome Graphs Using the vg Toolkit. *Genome Biology*, **21**(1), 35.

Kehr, B. *et al.* (2014). Genome alignment with graph data structures: a comparison. *BMC Bioinformatics*, **15**(1).

Lee, C. *et al.* (2002). Multiple sequence alignment using partial order graphs. *Bioinformatics*, **18**(3), 452–464.

Li, H. *et al.* (2009). The sequence alignment/map format and samtools. *Bioinformatics*, **25**(16), 2078–2079. 19505943[pmid].

Li, H. *et al.* (2020). The Design and Construction of Reference Pangenome Graphs with Minigraph. *Genome Biology*, **21**(1), 265.

Liang, Q. and Lonardi, S. (2021). Reference-agnostic representation and visualization of pan-genomes. *BMC Bioinformatics*, **22**(1), 502.

Logsdon, G. A. *et al.* (2021). The structure, function and evolution of a complete human chromosome 8. *Nature*, **593**(7857), 101–107.

Miga, K. H. *et al.* (2020). Telomere-to-Telomere Assembly of a Complete Human X Chromosome. *Nature*, **585**(7823), 79–84.

Nance, M. A. *et al.* (1999). Analysis of a very large trinucleotide repeat in a patient with juvenile Huntington's disease. *Neurology*, **52**(2), 392–394.

Neueder, A. *et al.* (2017). The pathogenic exon 1 HTT protein is produced by incomplete splicing in Huntington's disease patients. *Scientific Reports*, **7**(1), 1307.

Niu, F. *et al.* (2011). Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *arXiv preprint arXiv:1106.5730*.

Nurk, S. *et al.* (2021). The complete sequence of a human genome. *BioRxiv*.

Paten, B. *et al.* (2017). Genome Graphs and the Evolution of Genome Inference. *Genome Research*, **27**(5), 665–676.

Pevzner, P. A. (2004). De novo repeat classification and fragment assembly. *Genome Research*, **14**(9), 1786–1796.

Piovesan, A. *et al.* (2019). On the length, weight and gc content of the human genome. *BMC Research Notes*, **12**(1), 106.

Prezza, N. (2017). A framework of dynamic data structures for string processing. *arXiv preprint arXiv:1701.07238*.

Quinlan, A. R. and Hall, I. M. (2010). Bedtools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, **26**(6), 841–842.

Sekar, A. *et al.* (2016). Schizophrenia risk from complex variation of complement component 4. *Nature*, **530**(7589), 177–183.

Sibbesen, J. A. *et al.* (2021). Haplotype-aware pantranscriptome analyses using spliced pangenome graphs. *BioRxiv*.

Siren, J. *et al.* (2020). Haplotype-aware graph indexes. *Bioinformatics*, **36**(2), 400–407.

Wick, R. R. *et al.* (2015). Bandage: interactive visualization ofde novogenome assemblies: Fig. 1. *Bioinformatics*, **31**(20), 3350–3352.

Yokoyama, T. T. *et al.* (2019). MoMI-G: modular multi-scale integrated genome graph browser. *BMC Bioinformatics*, **20**(1), 548.

Zheng, J. X. *et al.* (2018). Graph drawing by stochastic gradient descent. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, **25**(9), 2738–2748.