Bioinformatics, 38(13), 2022, 3319–3326
https://doi.org/10.1093/bioinformatics/btac308
Advance Access Publication Date: 13 May 2022
Original Paper

Genome analysis

ODGI: understanding pangenome graphs

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

[1]Genomics Research Centre, Human Technopole, Milan 20157, Italy, [2]Quantitative Biology Center (QBiC), University of Tübingen, Tübingen 72076, Germany, [3]Biomedical Data Science, Department of Computer Science, University of Tübingen, Tübingen 72076, Germany and [4]Department of Genetics, Genomics and Informatics, University of Tennessee Health Science Center, Memphis, TN 38163, USA

*To whom correspondence should be addressed.
[†]The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors.

Associate Editor: Peter Robinson

Received on November 9, 2021; revised on March 18, 2022; editorial decision on April 23, 2022

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 Optimized Dynamic Genome/Graph Implementation (ODGI), a novel suite of tools that implements scalable algorithms and has an efficient in-memory representation of DNA pangenome graphs in the form of variation graphs. ODGI supports pre-built graphs in the Graphical Fragment Assembly format. ODGI includes tools for detecting complex regions, extracting pangenomic 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 and implementation: 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/pangenome/odgi/blob/master/guix.scm.

Contact: egarris5@uthsc.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

A pangenome models the full set of genomic elements in a given species or clade (Computational Pan-Genomics Consortium, 2018; Eizenga *et al.*, 2020b; Tettelin *et al.*, 2008). In contrast to reference-based approaches which relate samples to a single genome, these data structures encode the mutual relationships between all the genomes represented (Ballouz *et al.*, 2019). A class of methods to represent pangenomes involves sequence graphs (Hein, 1989; Paten *et al.*, 2017) where homologous regions between genomes are compressed into single representations of all alleles present in the pangenome. In sequence graphs, node labels are genomic sequences with edges connecting those nodes. A bidirected sequence graph can represent both strands of DNA. On this model, variation graphs add the concept of paths representing linear DNA sequences as traversals through the nodes of the graph (Garrison *et al.*, 2018). For example, a path can be a genome, haplotype, contig or read.

Pangenome graphs can be constructed by multiple sequence alignment (Grasso and Lee, 2004; Lee *et al.*, 2002) or by transitively reducing an alignment between sequences to an equivalent, labeled sequence graph (Garrison, 2019; Kehr *et al.*, 2014). Current methods to build these graphs are still under active development (Armstrong *et al.*, 2020; Garrison *et al.*, 2021; Li *et al.*, 2020), 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.

Pangenome graphs let us encode any kind of variation, allowing the generation of comprehensive data systems that builds the basis for the analyses of genome evolution. The Human Pangenome

Reference Consortium (HPRC) and Telomere-to-Telomere (T2T) consortium (Jarvis *et al.*, 2022; Logsdon *et al.*, 2021; Miga *et al.*, 2020; Nurk *et al.*, 2021) have recently demonstrated that high-quality haploid and diploid *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 to analyze pangenomes.

Although pangenome graphs are data structures of utility to researchers (Baaijens *et al.*, 2019; Computational Pan-Genomics Consortium, 2018; Garrison *et al.*, 2018; Hickey *et al.*, 2020; Sibbesen *et al.*, 2021), the scientific community still lacks a toolset capable of operating on gigabase-scale pangenome graphs constructed from whole-genome assemblies. Such an effort began with the VG toolkit (Garrison *et al.*, 2018), but its tools do not efficiently handle pangenome graphs presenting complex motifs that result from repetitive sequences. Here, we refocus the effort with the Optimized Dynamic Genome/Graph Implementation (ODGI) toolkit, a compatible, but independent pangenome graph interrogation and transformation system specifically implemented to handle the data scales encountered when working with pre-built constructed pangenomes comprising hundreds of haplotype-resolved genomes. ODGI offers a set of standard operations on the variation graph data model (Fig. 1), generalizing ‘genome arithmetic’ concepts, like those found in BEDTools (Quinlan and Hall, 2010), to work on pangenome graphs. Furthermore, it provides a variety of tools for graph 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 (Eizenga *et al.*, 2020b; Garrison, 2019). In the variation graph, $V = (N, E, P)$, nodes $N = n_1 \dots n_{|N|}$ contain genomic sequences. Each node n_i has an identifier i and an implicit reverse complement $\overline{n_i}$, and a node strand s corresponds to one of such orientations. Edges $E = e_1 \dots e_{|E|}$ represent ordered pairs of node strands: $e_i = (s_a, s_b)$. Paths $P = p_1 \dots p_{|P|}$ describe walks over node strands: $p_i = s_1 \dots s_{|p_i|}$. 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.

3 Implementation

The ODGI toolkit builds on existing approaches to efficiently store and manipulate pangenome graphs in the form of 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 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 (Eizenga *et al.*, 2020a; Siren *et al.*, 2020). 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 depth of path coverage of some nodes, the so-called node 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, we cannot understand them, and our only option is to build graphs that do not include them. Our goal is to build tools that allow for a wide range of uses of pangenome graphs, including cases with potentially high path depth. 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 build and modify the graph and its paths in parallel.

Algorithm 1: ODGI’s relativistically packed *Node* structure and the *Step* structure used to represent the paths as doubly linked lists.

```
Struct Node contains
    id ∈ ℕ // an identifier
    lock // atomic locking primitive
    sequence = [A|T|G|C|N]+
    // bit-packed vector of edges
    edges = $(x_i, x_j)*$ : $(i, j) \in [1 \dots \Sigma]^2$
    // bit-packed vector of id deltas
    decoding $x_1 \dots x_\Sigma \in \mathbb{N}^\Sigma$
    // bit-packed vector of path steps
    path_steps $[Step_1 \dots Step_n]*$
end
Struct Step contains
    path_id ∈ ℕ // the path's global id
    is_rev ∈ (0, 1) // the step orientation
    is_start ∈ (0, 1) // if first step in path
    is_end ∈ (0, 1) // if last step in path
    prev_δ ∈ [1 ... Σ] // δ-encoded previous node
    prev_rank ∈ ℕ // step rank on previous node
    next_δ ∈ [1 ... Σ] // δ-encoded previous node
    next_rank ∈ ℕ // step rank on next node
end
```

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 (Algorithm 1). 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 pangenome graph property (2). We define the edges and path steps traversing the node in terms of this alphabet of δ’s.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

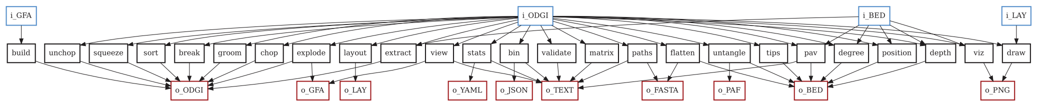

Fig. 1. Overview of the methods provided by ODGI (in black) and their supported input (in blue) and output (in red) data formats (A color version of this figure appears in the online version of this article.)

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).

4 Overview

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 (Section 4.1).
- *odgi view* converts the ODGI data model into GFA file (Section 4.1).
- *odgi viz* provides a linear visualization of the graph (Section 5.1).
- *odgi draw* renders a 2D image of the graph (Section 5.1).
- *odgi extract* excerpts subsets of the graph based on path ranges (Supplementary Section S.3).
- *odgi explode* breaks the graph into connected components (Supplementary Section S.3).
- *odgi squeeze* unifies disjoint graphs (Supplementary Section S.3).
- *odgi chop* breaks long nodes into shorter ones (Supplementary Section S.3).
- *odgi unchop* combines unitig nodes (Supplementary Section S.3).
- *odgi break* removes cycles in the graph (Supplementary Section S.3).
- *odgi prune* removes complex regions (Supplementary Section S.3).
- *odgi groom* resolves spurious inverting links (Supplementary Section S.3).
- *odgi position* lifts coordinates between path and graph positions (Section 5.2).
- *odgi untangle* deconvolutes paths relative to a reference (Section 5.2).
- *odgi tips* finds path end points relative to a reference (Supplementary Section S.2).
- *odgi sort* orders the graph nodes (Section 5.3).
- *odgi layout* establishes a 2D layout (Section 5.3).
- *odgi matrix* derives the pangenome matrix (Supplementary Section S.5).
- *odgi paths* lists and extracts paths in FASTA (Supplementary Section S.5).
- *odgi flatten* converts the graph to FASTA and BED (Supplementary Section S.5).
- *odgi pav* computes presence–absence variations (Supplementary Section S.5).
- *odgi stats* provides numerical properties of the graph (Section 5.4).
- *odgi bin* generates a summarized view of the graph (Supplementary Section S.5).
- *odgi depth* describes node depth over graph and path positions (Section 5.4).
- *odgi degree* describes node degree over graph and path positions (Section 5.4).

Each tool focuses on a small set of related operations. Most read or write the native ODGI format ('og' extension) (Fig. 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 requiring that we build an entirely new set of bioinformatic methods to work in this difficult new pangenomic 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 process itself is lossless. A graph in ODGI format represents everything that is in the input GFAv1 graph, without any loss of information. ODGI does not natively support GFAv2 or rGFA. GFAv2 is similar to GFAv1, but includes process-related annotations of assembly graphs not relevant for pangenome analyses. rGFA embeds a single coordinate hierarchy over the graph that links all sequences into a single base reference genome. This positional model depends on a particular graph induction algorithm Li *et al.* (2020). In contrast, ODGI implements coordinate translation dynamically (e.g. *odgi position* and *odgi untangle*), allowing use of any embedded genome as a reference. Its input graphs can represent any kind of alignment between the genomes. GFAv1 is fully capable of representing many reference genome coordinate systems simultaneously, which supports a reference-agnostic approach that uses the entire pangenome sequence space as a reference system. In doing so, our approach has the advantage of maintaining backward compatibility with existing tools based on genome sequences.

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

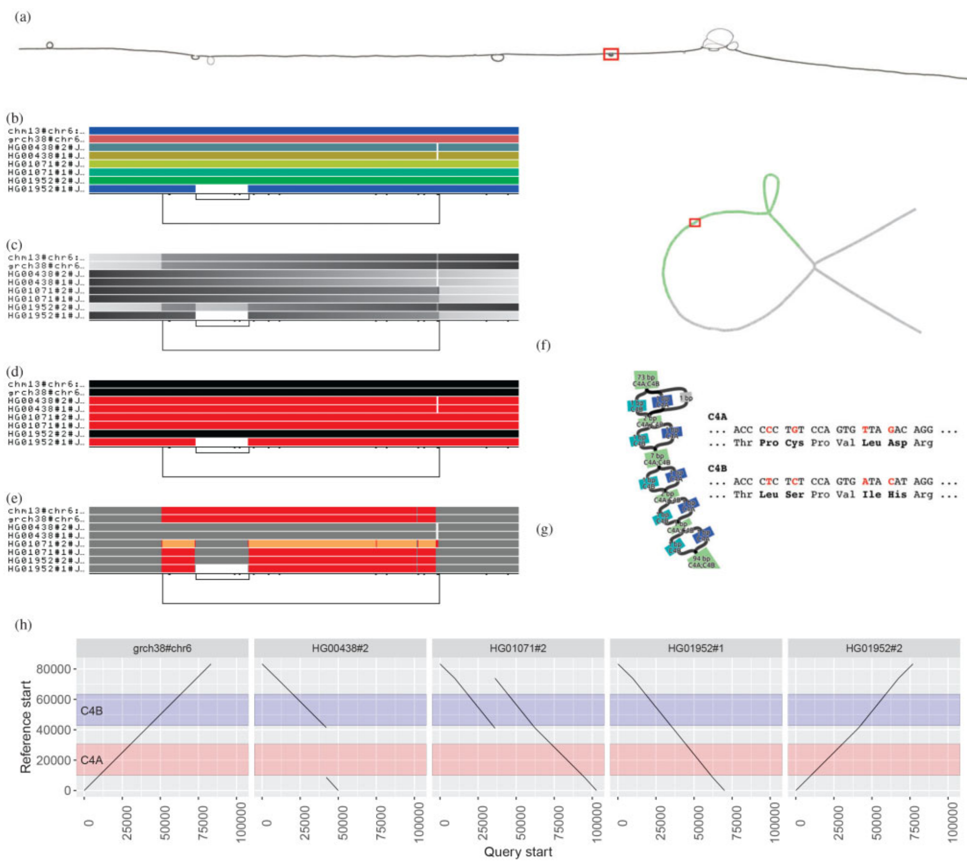

Fig. 2. Visualizing the major histocompatibility complex (MHC) and complement component 4 (C4) pangenome graphs. (**a**) *odgi draw* layout of the MHC pangenome graph extracted from a whole human pangenome graph of 90 haplotypes. The red rectangle highlights the C4 region. (**b–e**) *odgi viz* visualizations of the C4 pangenome graph, where eight paths are displayed: two reference genomes (CHM13 and GRCh38 on the top) and six haplotypes of three diploid individuals. (b) *odgi viz* default modality: the image shows a quite linear graph. The 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). (c) Color by path position. The top two reference genomes and one haplotypes (HG01952#2) go from left to right, while five 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 two reference genomes. (e) *odgi viz* color by node depth: using the Spectra color palette with four levels of node depths, white indicates no depth, while gray, red and yellow indicate depth 1, 2 and greater than or equal to 3, respectively. Coloring by node 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 gray paths have one copy of these genes. HG01071#2 presents three copies of the *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**) Layout of the C4 pangenome graph made with the *Bandage* tool (Wick *et al.*, 2015) and annotated by using *odgi position*. Green nodes indicate the C4 genes (in red). 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 (no black lines in the region annotated in red) (A color version of this figure appears in the online version of this article.)

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 (Section 5.5).

To support interchange with other pangenome tools or text-based processing, *odgi view* converts a graph in ODGI binary format to GFAv1. ODGI utilizes the PanSN (Garrison, 2021) specification to embed sample and haplotype information in the sequence name. This harmonizes the biosample information present in FASTA, GFA, PAF, VCF, BED, BEDPE, SAM/BAM and GFF/GTF formats related to the graph and its embedded genome sequences. By embedding all sequences into a single hierarchical namespace related to fundamental biological groupings in the input (e.g. biosample, individual, pooled group), PanSN allows us to utilize all assemblies in the pangenome as a combined reference coordinate model.

5 Results

Here, we apply our methods to a series of analyses, highlighting how ODGI can assist in exploring the biological features of pangenome graphs. We follow typical analyses that we have found critical to interpreting whole genome alignments represented in the variation graph model.

To simplify our exposition, we will extract small graph regions that are easy to interpret and describe. We focus on a handful of difficult loci from the human pangenome, extracting them from a prototype human pangenome graph built with the Pangenome Graph Builder pipeline (Garrison *et al.*, 2021). Pangenome graphs built from hundreds of haplotype-resolved *de novo* genome assemblies are very large, but it is often only necessary to work with only a small portion of the genomes represented, such as a specific locus (Fig. 2a) or a smaller region (Fig. 2b–g), or even a single gene (Fig. 3). This simplifies the downstream analyses and reduces the resources to work only with the extracted graphs. More on graph extraction and edit operations can be found at Supplementary Section S3.

5.1 Visualizing pangenome graphs

Visualization methods help us quickly gain insight into otherwise opaque biological data. We find visualization essential for understanding pangenome graphs. We pursue a novel approach to visualization with *odgi draw* and *odgi viz*, two tools that provide scalable

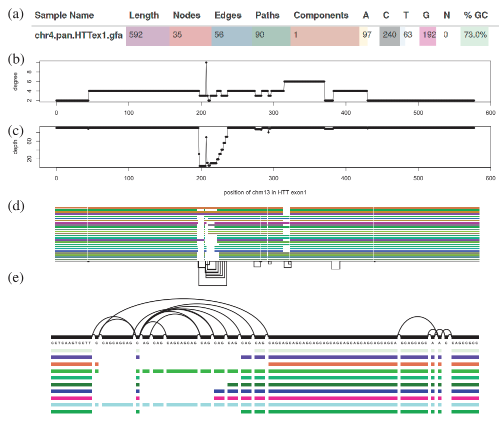

Fig. 3. Features of a 90-haplotype human pangenome graph of the exon 1 huntingtin gene (*HTTexon1*): (**a**) excerpt of vital statistics of the *HTTexon1* graph displayed by MultiQC's ODGI module. (**b**) Per nucleotide node degree distribution of CHM13 in the *HTTexon1* graph. Around position 200 there 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

ways of generating raster images showing the high-level structure of even large pangenome graphs (Fig. 2).

Using *odgi extract*, we extracted the major histocompatibility complex (MHC) locus from a 90-haplotype human chromosome 6 pangenome graph from the HPRC. Specifically, the graph contains the human references GRCh38, CHM13 and the contigs of 44 diploid individuals that encode all possible variations including those in telomeres and centromeres. The MHC genes are involved in antigen presentation, inflammation regulation, the complement system and the innate and adaptive immune responses (Shiina *et al.*, 2009). MHC genes are highly polymorphic, i.e. there are multiple different alleles across individuals in a population. Such variability becomes evident when we apply *odgi draw* to visualize the graph layout of a human MHC pangenome graph (Fig. 2a) (of note, *odgi layout* first generates the drawn projection, see Section 5.3). The visualization displays the graph topology in two dimensions (2D), with structural variation that appears as bubbles in the layout. A 2D rendering can be costly to compute, but we provide an implementation that scales linearly with pangenome sequence size, allowing us to apply it to large pangenome graphs.

The MHC locus includes the complement component 4 (C4) region, which encodes proteins involved in the complement system. In Figure 2a, C4 corresponds to the small bubble highlighted by the red rectangle. As an example use case, we took a closer look at the C4 region of the MHC by extracting it from the full MHC pangenome graph with *odgi extract*. Then, we visualized this subgraph by applying *odgi viz*, which produces binned, linearized renderings in 1 dimension (1D), where the graph is ordered in 1D across the horizontal axis, with each path represented by a row of the vertical axis (Fig. 2b–e). For each path, graph nodes are arranged from left to right, with the colored bars indicating the paths and the nodes they cross. White spaces indicate where paths do not traverse the nodes. Directly consecutive nodes are displayed with no white space between the two. The meaning of the colors depends on how *odgi viz* is executed. By default, path colors are derived from the path names (Fig. 2b), which 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 (see Supplementary Section S1 for a more detailed explanation). 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. In humans, the C4 gene exists as two functionally distinct genes, *C4A* and *C4B*, which both vary in structure and copy number (Sekar *et al.*, 2016). In combination with the observed changes in path self-coverage, which represents copy number of a given path relative to the graph (Fig. 2e), the longer link at the bottom of Figure 2b–e indicates that the copy number status of these genes varies across the haplotypes represented in the pangenome. Moreover, the short nested variation on the left of the locus highlights that *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.

Nevertheless, complex, non-linear graph structures are difficult to interpret in a low number of dimensions. To overcome this limitation, *odgi viz* supports multiple visualization modalities (Fig. 2c–e), making it easy to grasp the properties and shape of the graph. For example, we can color the paths by path position (Fig. 2c), with light gray indicating where paths begin and dark gray where they end. This visualization is suitable for understanding graph node order, as smooth color gradients indicate that the node order respects the linear paths' coordinate systems. Pangenome graphs can represent both strands of the genomic sequences of the DNA. We can display such information by 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. 2d). Furthermore, we can use multiple color palettes to color the paths by how many times they traverse a node, which can be referred to as the path's depth or coverage of the node, the node depth. This highlights that in the C4 pangenome graph, the haplotypes present different number of copies of the C4 genes (Fig. 2e).

5.2 Untangling and navigating the pangenome

The key data in a pangenome graph is a representation of the alignment (i.e. the homology relationships) between genomic sequences. Navigating and understanding the graph requires coordinate systems to link other data to the sequences represented in the graph model. 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 sequences. Such a universal coordinate system allows us to support several kinds of 'lift-over' of coordinates between different sequences in the same or different graphs. As a demonstration, we took the C4 pangenome graph and added to its nodes gene annotation from GRCh38 (in GFF format file) using *odgi position* (Supplementary Section S2.1). The resulting TSV contains pairs of nodes and colors. Taking the graph and the TSV into Bandage (Wick *et al.*, 2015), the actual C4 genes are highlighted (Fig. 2f). Zooming to the nucleotide level, the annotation shows the single nucleotide differences of the *C4A* and the *C4B* genes (Fig. 2g).

odgi position can also translate graph and path positions between or within graphs, emitting the liftovers in BED format. 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. 10 kbp) has the best jaccard metric when compared to the query context. For a more detailed explanation of the *path jaccard* concept see Supplementary Section S2.2.

To obtain a more precise overview of the locus in Figure 2b–e, we applied *odgi untangle* with GRCh38 as a reference. *odgi untangle* segments paths into linear segments by breaking these segments where the paths loop back on themselves. In this way, we obtain information on the position and copy number status of the sequences in the collapsed locus, in BEDPE or PAF format. In the representation in Figure 2h, the orientation of the line indicates if the copy number is in forward or in reverse orientation compared to GRCh38. *odgi untangle* is able to work with any sets of reference sequences, converting the graph to lift-over maps compatible with standard software for projecting annotations and alignments from one genome to another. An explanation of the untangling process is given in Supplementary Section S2.

5.3 Latent graph structure reveals underlying biology

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. *odgi sort* allows the chaining of these sorting algorithms. As many of the algorithms are affected by the initial node order, this allows us to generate sorting pipelines that progressively refine the graph ordering.

We applied several of *odgi sort*'s 1D algorithms to a 90-haplotype human MHC pangenome and a C4 subgraph (Supplementary Fig. S2). The randomly sorted MHC graph (Supplementary Fig. S2a) hides its linear graph structure, whereas our novel path-guided (PG) stochastic gradient descent (SGD) algorithm, PG-SGD, is able to produce a globally linear ordered graph revealing the C4 region (Supplementary Fig. S2b). This exploits path information to order the graph nodes. PG-SGD learns a 1D or 2D organization of the graph nodes that matches nucleotide distances in graph paths (i.e. the sequences embedded in the graph). To scale to large graphs, we learn this projection in parallel via a HOGWILD! approach (Niu *et al.*, 2011). PG-SGD can be seen as an adaptation of SGD-based drawing (Zheng *et al.*, 2019) to pangenome graphs. In parallel, each HOGWILD! thread updates the relative position of pairs of nodes so that their distance in the layout, or their order, better-matches their nucleotide distance in the paths running through the graph. Following standard SGD approaches, the learning rate is reduced as the algorithm progresses, and execution continues until the adjustments to the model fall below a target threshold ϵ.

A PG-SGD sorting of C4 compresses both sides of the variant bubble into one dimension, leading to an interrupted pattern of nodes across the copy-number variable region (Supplementary Fig. S2c). Subsequently applying a topological sort clarifies the graph's latent structure, simplifying interpretation (Supplementary Fig. S2d). To find the best order of graph nodes in 1D, *odgi sort*'s multiple sorting algorithms can be combined into a sorting pipeline to take advantage of the strength of each (results not shown). 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 (Fig. 2a). A detailed description of the node ordering process can be found at Supplementary Section S4. As we have shown above, the node order is crucial to understand the biological features of a pangenome graph.

5.4 Graph features highlight variation

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

As a use case study (Fig. 3), we took a look at the metrics of a 90-haplotype human pangenome graph of the exon 1 huntingtin gene (*HTTexon1*). In particular, we obtained the number of nodes, edges, paths, components, bases, the graph length and the GC content with *odgi stats*. The output pangenome statistics in YAML textual file format was given to MultiQC's (Ewels *et al.*, 2016) newly added ODGI module. As can be seen in Figure 3a, we observe a very high GC content of 73.0% in the *HTTexon1* graph compared to the human genomic mean GC content of 40.9% (Piovesan *et al.*, 2019). This is in accordance with the literature (Neueder *et al.*, 2017). Despite this discovery, the MultiQC module provides an interactive way to comparatively explore statistics of an arbitrary number of graphs.

To investigate in detail which intricate regions in the *HTTexon1* graph are responsible for its genetic variation and high GC content, we took a look at the per nucleotide node degree (Fig. 3b) and node depth (Fig. 3c) distributions of CHM13 by using *odgi depth*'s and *odgi degree*'s BED output, respectively. The results indicate a highly

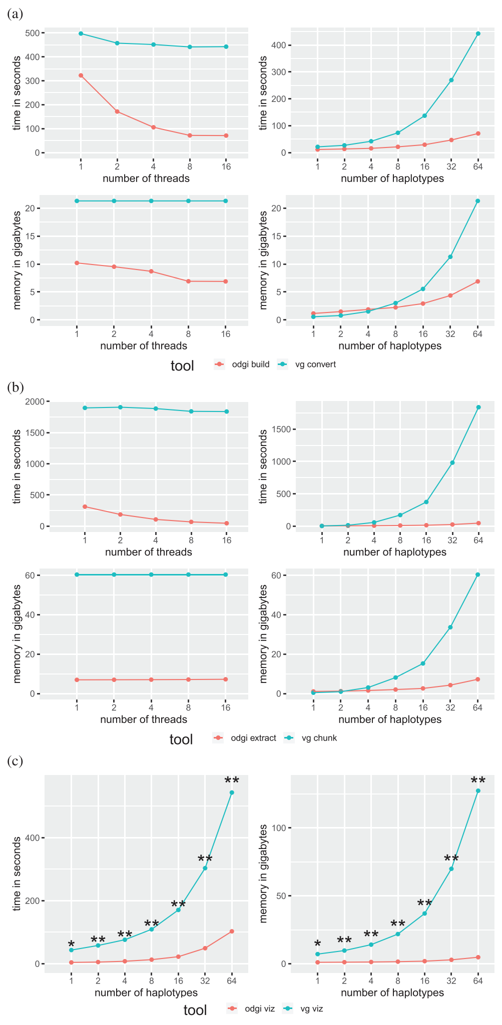

Fig. 4. Performance on a graph of human chromosome 6 from the HPRC. ODGI compares favorably to VG across all routine pangenomic tasks. Evaluations across threads were done using a 64 human haplotype graph. Evaluations across haplotypes were done using 16 threads. (**a**) Performance evaluation when translating a graph into the tools' respective native formats. (**b**) Performance evaluation when extracting the centromeric region from the HPRC graph. (**c**) Performance evaluation when visualizing a graph. Both tools were run with only one thread. *vg viz*: *A 816 MB SVG was produced which cannot be opened by any program. **All produced SVGs only contain an XML header, nothing else

polymorphic region around position 200 in the graph. Figure 3d supports this analysis. Zooming in on this region with *vg viz*, we can clearly identify the typical *HTTexon1* CAG variable repeat region (Fig. 3e). Figure 3b–d highlights 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).

5.5 Performance evaluation

Although many of the operations that ODGI provides are unique, some are common with the existing VG toolkit. We compare with these to highlight the practical performance implications of our graph data structure design. Our results highlight the efficient parallel algorithm implementations enabled by this design.

We compared the efficiency of ODGI (v0.6.3-56-gebc493f 'Pulizia') and VG (v1.37.0 'Monchio') for routine pangenome tasks. In particular, we measured the execution time and memory usage (i) of transforming a GFAv1 file into a tool's native format, (ii) the extraction of a subgraph, (iii) the visualization of a pangenome graph and (iv) the finding of path positions in a pangenome graph. These graph operations are key when it comes to the understanding of pangenome graphs. They are also a set of functions implemented in both toolkits. We ran these operations for a varying number of threads and haplotypes in the graph for a scaling analysis. We ran each evaluation configuration 10 times and report the mean of each run. All evaluations were performed on a VM in the German Network for Bioinformatics Infrastructure (deNBI) cloud with 28 cores and 256 GB of RAM. The presented results are from a 90-haplotype chromosome 6 human pangenome graph built with data from the HPRC. Specifically, the graph contains the human references GRCh38, CHM13 and the contigs of 44 diploid individuals that encode all possible variations including those in telomeres and centromeres. When transforming a GFAv1 file with VG, the static XG file format was used. The tools involved in the evaluation process require the XG format.

In general, ODGI makes comparatively better use of multi-threading and requires much less memory (Fig. 4, Supplementary Table S4) across all operations. ODGI scales much better than VG when working with complex regions of the graph. For example, extracting a difficult centromeric subgraph (Fig. 4b), ODGI is up to 40 times faster and requires 8 times less memory than VG.

Both visualization tools can only make use of a single thread. For a 1 haplotype, graph *vg viz* produces a 816MB SVG which can't be opened by the standard programs to date. For larger graphs, *vg viz* runs through and produces SVGs with only the XML header. This makes it unusable for large graphs.

We also measured the disk space usage of GFAv1, ODGI's and VG's binary formats (Supplementary Table S5). While VG's XG occupies less disk space for smaller graphs, ODGI requires less space for graphs having 32 haplotypes or more. We hypothesize that this indicates the lower marginal cost for additional haplotypes when using ODGI's id delta encoding scheme.

6 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.

By providing a set of standard analysis 'verbs' to interact with pangenome graphs, ODGI enables users to explore and discover important biological features captured in this flexible, inclusive model. 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 the genomic studies at different taxonomic levels and in biomedical research. This progression is already afoot, particularly for targets that involve complex variation, such as cancer (The Computational Pan-Genomics Consortium, 2016), plant pangenomics (Bayer *et al.*, 2020, 2022; Li *et al.*, 2022; Liu *et al.*, 2020; Qin *et al.*, 2021) and metagenomics (Zhong *et al.*, 2021). Also, when studying animals like bovines (Bovine Pan-Genome Consortium, 2022; Leonard *et al.*, 2021; Talenti *et al.*, 2022).

Currently, bacterial pangenomes are best handled by specialized tools like PPanGGolin (Gautreau *et al.*, 2020), PanGraph (Noll *et al.*, 2022) or PanX (Ding *et al.*, 2018). The latter one doesn't build a graphical representation of a pangenome. But, it already has a very developed eco-system, which allows a detailed analysis of bacterial pangenomes using an interactive GUI. Unlike these approaches, which provide a monolithic, integrated solution to understanding pangenomes, ODGI is designed as a low-level toolkit that can work on a generic pangenome graph model frequently used by other existing methods. We hope that this design renders it useful to pangenome analysis pipeline authors. Other pangenome analysis platforms, like PanTools (Sheikhizadeh *et al.*, 2016) provide access to pangenome analyses at the scales we demonstrate with ODGI, but use specialized de Bruijn graph models to achieve this. In contrast ODGI supports the highly generic variation graph model, which has greater representational power than de Bruijn graphs.

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.

The performance analysis shows that ODGI outperforms VG when handling large, complex pangenome graphs. Across the evaluation of key graph operations, ODGI's memory peak was 10GB. This makes it perfectly suited to be run interactively on a recent laptop. We expect that ODGI will be able to handle the next phase of the HPRC, a pangenome graph constructed from 300 individuals, without any problems.

While ODGI does not construct graphs from scratch nor is capable of extending them, it is already the backbone of the Pangenome Graph Builder pipeline (Garrison *et al.*, 2021). Its static, large-scale 1D and 2D 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 pangenome graph browsers are reference-centric (Beyer *et al.*, 2019; Yokoyama *et al.*, 2019), have a limited predefined coordinate system (Durant *et al.*, 2021), or focus primarily on 2D representations (Gonnella *et al.*, 2019; Wick *et al.*, 2015). 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 genomic sequences. 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 for understanding genome variation.

Acknowledgements

The authors 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

The authors gratefully acknowledge support from National Institutes of Health/NIDA U01DA047638 (E.G.), National Institutes of Health/NIGMS R01GM123489 (E.G. and P.P.) and NSF PPoSS Award #2118709 (E.G. and P.P.). S.H. acknowledges funding from the Central Innovation Programme (ZIM) for SMEs of the Federal Ministry for Economic Affairs and Energy of Germany. S.N. acknowledges 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 and 031A532B].

Conflict of Interest: The authors have nothing to declare.

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**, 246–251.

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

Ballouz,S. *et al.* (2019) Is it time to change the reference genome? *Genome Biol.*, **20**, 159.

Bayer,P.E. *et al.* (2020) Plant pan-genomes are the new reference. *Nat. Plants*, **6**, 914–920.

Bayer,P.E. *et al.* (2022) Wheat panache – a pangenome graph database representing presence/absence variation across 16 bread wheat genomes. bioRxiv. https://doi.org/10.1101/2022.02.23.481560.

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

Bovine Pan-Genome Consortium. (2022) Bovine pan-genome consortium. https://njdbickhart.github.io/ (February 2022, date last accessed).

Computational Pan-Genomics Consortium. (2018) Computational pan-genomics: status, promises and challenges. *Brief. Bioinf.*, **19**, 118–135.

Ding,W. *et al.* (2018) panX: pan-genome analysis and exploration. *Nucleic Acids Res.*, **46**, e5.

Durant,E. *et al.* (2021) Panache: a web browser-based viewer for linearized pangenomes. *Bioinformatics*, **37**, 4556–4558.

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

Eizenga,J.M. *et al.* (2020b) Pangenome graphs. *Annu. Rev. Genomics Hum. Genet.*, **21**, 139–162.

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

Garrison,E. (2019) Graphical Pangenomics (Doctoral thesis). https://doi.org/10.17863/CAM.41621.

Garrison,E. (2021) Pansn-spec*:* Pangenome Sequence Naming. https://github.com/pangenome/PanSN-spec (May 2022, date last accessed).

Garrison,E. *et al.* (2018) Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nat. Biotechnol.*, **36**, 875–879.

Garrison,E. *et al.* (2021) The Pangenome Graph Builder. https://github.com/pangenome/pggb (May 2022, date last accessed).

Gautreau,G. *et al.* (2020) PPanGGOLiN: depicting microbial diversity via a partitioned pangenome graph. *PLoS Comput. Biol.*, **16**, e1007732.

GFA Working Group. (2016) Graphical Fragment Assembly *(*GFA*)* Format Specification. https://github.com/GFA-spec/GFA-spec (May 2022, date last accessed).

Gonnella,G. *et al.* (2019) GfaViz: flexible and interactive visualization of GFA sequence graphs. *Bioinformatics*, **35**, 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**, 1546–1556.

Hein,J. (1989) A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. *Mol. Biol. Evol.*, **6**, 649–68.

Hickey,G. *et al.* (2020) Genotyping structural variants in pangenome graphs using the vg toolkit. *Genome Biol.*, **21**, 35.

Jarvis,E.D. *et al.* (2022) Automated assembly of high-quality diploid human reference genomes. bioRxiv. https://doi.org/10.1101/2022.03.06.483034.

Kehr,B. *et al.* (2014) Genome alignment with graph data structures: a comparison. *BMC Bioinformatics.*, **15**, 99.

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

Leonard,A.S. *et al.* (2021) Bovine pangenome reveals trait-associated structural variation from diverse assembly inputs. bioRxiv. https://doi.org/10.1101/2021.11.02.4466900.

Li,H. *et al.*; 1000 Genome Project Data Processing Subgroup. (2009) The sequence alignment/map format and samtools. *Bioinformatics*, **25**, 2078–2079.

Li,H. *et al.* (2020) The design and construction of reference pangenome graphs with minigraph. *Genome Biol.*, **21**, 265.

Li,H. *et al.* (2022) Graph-based pan-genome reveals structural and sequence variations related to agronomic traits and domestication in cucumber. *Nat. Commun.*, **13**, 682.

Liu,Y. *et al.* (2020) Pan-genome of wild and cultivated soybeans. *Cell*, **182**, 162–176.e13.

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

Miga,K.H. *et al.* (2020) Telomere-to-telomere assembly of a complete human X chromosome. *Nature*, **585**, 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**, 392–394.

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

Niu,F. *et al.* (2011) Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *NIPS'11: Proceedings of the 24th International Conference on Neural Information Processing Systems,* 693–701.

Noll,N. *et al.* (2022). Pangraph: scalable bacterial pan-genome graph construction. bioRxiv.

Nurk,S. *et al.* (2022) The complete sequence of a human genome. *Science*, **376**, 44–53.

Paten,B. *et al.* (2017) Genome graphs and the evolution of genome inference. *Genome Res.*, **27**, 665–676.

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

Prezza,N. (2017) A framework of dynamic data structures for string processing. *Leibniz Internatnal Proceedings in Informatics*, 75.

Qin,P. *et al.* (2021) Pan-genome analysis of 33 genetically diverse rice accessions reveals hidden genomic variations. *Cell*, **184**, 3542–3558.

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

Sekar,A. *et al.*; Schizophrenia Working Group of the Psychiatric Genomics Consortium. (2016) Schizophrenia risk from complex variation of complement component 4. *Nature*, **530**, 177–183.

Sheikhizadeh,S. *et al.* (2016) PanTools: representation, storage and exploration of pan-genomic data. *Bioinformatics*, **32**, 487–493.

Shiina,T. *et al.* (2009) The HLA genomic loci map: expression, interaction, diversity and disease. *J. Hum. Genet.*, **54**, 15–39.

Sibbesen,J.A. *et al.* (2021) Haplotype-aware pantranscriptome analyses using spliced pangenome graphs. bioRxiv. https://doi.org/10.1101/2021.03.26.437240.

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

Talenti,A. *et al.* (2022) A cattle graph genome incorporating global breed diversity. *Nat. Commun.*, **13**, 910.

Tettelin,H. *et al.* (2008) Comparative genomics: the bacterial pan-genome. *Curr. Opin. Microbiol.*, **11**, 472–477.

The Computational Pan-Genomics Consortium. (2016) Computational pan-genomics: status, promises and challenges. *Brief. Bioinformatics*, **19**, bbw089.

Wick,R.R. *et al.* (2015) Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, **31**, 3350–3352.

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

Zheng,J.X. *et al.* (2019) Graph drawing by stochastic gradient descent. *IEEE Trans. Vis. Comput. Graph.*, **25**, 2738–2748.

Zhong,C. *et al.* (2021) Integrating pan-genome with metagenome for microbial community profiling. *Comput. Struct. Biotechnol. J.*, **19**, 1458–1466.