Hierarchically Visualizing Metagenome Assembly Graphs with MetagenomeScope

Marcus Fedarko*
University of Maryland
College Park, Maryland
mfedarko@umd.edu

Jay Ghurye
University of Maryland
College Park, Maryland
jayg@cs.umd.edu

Todd Treangen
University of Maryland
College Park, Maryland
treangen@umd.edu

Mihai Pop
University of Maryland
College Park, Maryland
mpop@umiacs.umd.edu

ABSTRACT

Manual inspection of sequence assembly graphs can be useful not only as a debugging tool when developing assembly software, but also as a way to uncover interesting biological patterns, such as structural differences between the two or more haplotypes being analyzed in a genomic or metagenomic experiment. Current tools for visualizing these graphs, however, emphasize a high-level representation, based on force-directed layouts, aimed at revealing the broad level quality of an assembly rather than its small scale structure. As a result, it is difficult for users to piece together a unified understanding of both the high-level structure of the assembly graph and the detailed patterns found within these graphs.

We present a new strategy for displaying genome assembly graphs that emphasizes the expected linear structure of genome assemblies and allows a multi-level exploration of the structure of the graph. This approach is implemented in MetagenomeScope, an interactive web-based tool.

We detail the novel layout algorithm employed by MetagenomeScope and provide a qualitative comparison with the main tools currently used to explore genome assembly graphs. We demonstrate that MetagenomeScope provides a set of unique capabilities that enable the effective visual exploration of assembly graphs in the context of several common workflows, such as genome finishing or the discovery of structural variants within assemblies.

CCS CONCEPTS

• **Human-centered computing → Visualization systems and tools**; • **Applied computing → Computational genomics**;

KEYWORDS

Visualiation, Metagenomics, Assembly Graph

ACM Reference Format:
Marcus Fedarko, Jay Ghurye, Todd Treangen, and Mihai Pop. 2018. Hierarchically Visualizing Metagenome Assembly Graphs with MetagenomeScope. In *Proceedings of University of Maryland Department of Computer Science (UMDCS)*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*The student for whom this undergraduate honors thesis is presented.

UMDCS, Spring 2018, College Park, Maryland USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modern approaches for genome assembly—the computational process of reconstructing a genomic sequence from the many small DNA fragments "read" by a sequencing instrument—rely on graph-theoretic algorithms. Specifically, the assembly problem is formulated as the traversal of a graph that encodes the links between DNA segments whose adjacency is implied by the sequenced reads. Repeats, sequencing errors, mutations, and other genomic features complicate the structure of this assembly graph, giving rise to branching and cyclic paths which make it difficult to automatically identify the traversal of the graph that spells out the correct genome sequence. Resolving this complexity frequently necessitates manual inspection of the graph and even the generation of additional experimental data, in a process known as finishing, in order to complete the assembly process [14].

Effective visualization of assembly graphs can substantially speed up the finishing process. More importantly, however, in many cases the assembly graph contains information about interesting biological signatures. In the case of eukaryotic genomes, the assembly represents the combination of two distinct haplotypes, and structural differences between these haplotypes (an important genetic feature) appear as "bubbles" in the graph. In the case of metagenomic data resulting from the sequencing of complex microbial mixtures, the assembly comprises multiple haplotypes (genomes of closely related strains co-existing in a sample), and the assembly graph can reveal structural genomic variants associated with interesting biological phenomena such as antigenic drift or lateral gene transfer.

Current tools for visualizing assembly graphs are limited in their ability to display a hierarchical, semi-linearized overview of a graph. Many tools employ force-directed layout algorithms for positioning contigs (nodes in the graph which represent fragments of DNA) and the edges between them, which can result in visualizations that generate an appealing high-level representation of the graph but make analysis of details (for example, “bubble”-like regions) intractable. This is a particular downside for metagenomic assemblies, where analysis of these fine-grained details can provide valuable biological insights.

MetagenomeScope addresses the limitations of current assembly visualization tools through a novel hierarchical layout process that results in a semi-linear representation of the assembly graph, and

through the automatic detection of certain structural patterns in the graph. Our tool is implemented as a client-side web application and provides an array of controls for interacting with assembly graphs.

2 RELATED WORK

Several tools were developed in recent years for the visualization of assembly graphs. These include Bandage [16], ABySS-Explorer [12], and Ray Cloud Browser [7]. Many of these tools are primarily targeted at the needs of users of their corresponding assembly programs: ABySS-Explorer was designed to support ABySS [15] and Ray Cloud Browser supports Ray [2]. Bandage, though, was developed with similar general applications in mind as MetagenomeScope. Below we highlight some of the key differences between our work and these prior approaches.

2.1 Layout Differences

Bandage, ABySS-Explorer, and Ray Cloud Browser all use force-directed layout algorithms for depicting assembly graphs, which can complicate the inspection of small-scale details in their visualizations. MetagenomeScope uses Graphviz' [6] *dot* tool to perform hierarchical layout, with an extra graph linearization step based on identified structural patterns of contigs. (Section 3.1 provides an in-depth description of MetagenomeScope's layout process.)

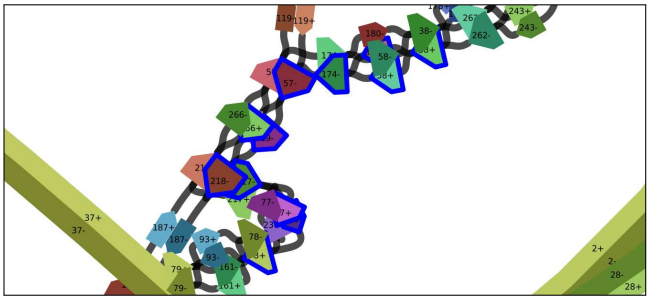

Figure 1: Screenshot captured in Bandage v0.8.1 of a specified region of an E. coli assembly graph. The graph was drawn as a "double graph" in Bandage's linear layout mode with otherwise default settings. Selected contigs are outlined in blue by Bandage.

To demonstrate the differences between the two approaches we highlight a region of a Velvet [17] *Escherichia coli* assembly graph (`E_coli_LastGraph`) distributed with Bandage, rendered in Bandage (Fig. 1) and MetagenomeScope (Fig. 2). In both cases, the visualization has been zoomed to fit the selected contigs 38+, 180+, 58+, 174-, 57-, 266+, 219-, 218-, 217-, 77+, 236-, and 78+ (the + or - sign indicates the inferred strand of the DNA).[1] Bandage's representation of this region is visually dense, and does not reveal much insight into the contigs' relative positions in the assembly

[1]As can be seen in Figs. 1 and 2, the identifiers used for contigs that lack a definite orientation differ slightly between Bandage and MetagenomeScope. MetagenomeScope omits + signs, and uses the - sign (where applicable) as a prefix instead of as a suffix. So the same list of contig identifiers in MetagenomeScope is 38, 180, 58, -174, -57, 266, -219, -218, -217, 77, -236, and 78.

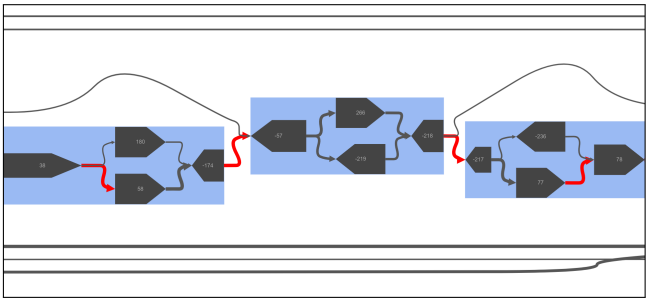

Figure 2: Screenshot captured in MetagenomeScope of the same region of the same graph as in Fig. 1. The collections of contigs highlighted in blue have been flagged as "bubbles," regions of the assembly graph exhibiting a converge→diverge→converge pattern [13]. The thicknesses of edges in the display relate to the number of links supporting the corresponding connection between two contigs; edges highlighted in red are supported by a significantly large amount of links. See section 3.1.2 for details on how edge weights are visually represented in MetagenomeScope.

graph. MetagenomeScope's representation, however, clearly highlights the connections between these contigs, as well as the relative multiplicities of the edges between them.

2.2 Application Environments

Bandage and ABySS-Explorer are desktop applications, while Ray Cloud Browser and MetagenomeScope are web applications.

Web applications like MetagenomeScope and Ray Cloud Browser reduce the amount of work needed for end users to view visualizations, since the installation process is replaced with merely opening a window in a web browser. This functionality makes the tools easier to use; however, it limits the complexity of the graphs that can be rendered due to limits in the computational resources made available by web browswer or available on a user's computer. Such resource limitations are most relevant for the computation of the graph layout. Ray Cloud Browser relies on server-side code for this task while MetagenomeScope performs this task off-line in its preprocessing script.

3 METHODS

3.1 Hierarchical graph layout

MetagenomeScope relies on the layout algorithms implemented in the Graphviz package [6], specifically the *dot* hierarchical layout engine, modified as follows. First, the graph is decomposed into a collection of structural patterns as described in more detail in section 3.2. These patterns are then grouped together throughout the layout process in order to retain the visual presentation of these small-scale details.

We initially structured these patterns by defining them as "clusters," using a feature of the language supported by *dot*. However, we noticed that *dot* tended to route edges through the borders defined by the clusters, thereby making the graph difficult to interpret (see Fig. 3 for an example).

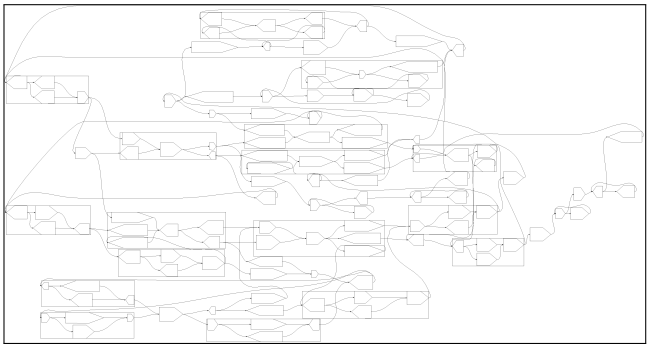

Figure 3: *dot*'s layout of a sample assembly graph (`sample_LastGraph`) provided with Bandage. The input DOT file used for this layout was created without using the structural pattern backfilling technique described in section 3.1. (This figure, along with Fig. 4, was generated using `dot -Tpng` with the lines `rotate=90;` and `dpi=50;` added to the header of its DOT file in order to rotate the output image and decrease its resolution to fit within this document.)

To address this issue, we modified the invocation of *dot* by laying out the contents of each structural pattern in isolation (calling *dot* once for each structural pattern) and saving the relative positions of the contigs within each pattern, as well as the dimensions of each pattern's bounding box. In a final invocation, we represent each structural pattern as a single node, using the dimensions determined in the earlier layout step, then "backfill" the original contigs and edges within the corresponding regions in the graph. This process is possible because MetagenomeScope limits contigs to being in at most one pattern, so the set of contigs contained in a given pattern is guaranteed to be disjoint from the sets of contigs contained in other patterns in the graph.

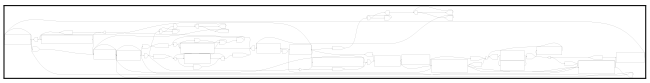

Figure 4: *dot*'s layout of the same assembly graph as in Fig. 3. The input DOT file used for this layout was created using the backfilling technique described in section 3.1. Notice the linearity of this layout compared with that in Fig. 3 that has been introduced by preventing edges from being routed through contig groups.

We found that this technique not only addressed the routing of edges through structural patterns, but also significantly helped linearize *dot*'s layout for many graphs containing structural patterns. An example of this effect is shown in Fig. 4. For comparative purposes, versions of the DOT files specifying layouts without backfilling being used (as in Fig. 3) can be generated using the `-nbdf` option in MetagenomeScope's preprocessing script, and versions of the DOT files specifying layouts with backfilling (as in Fig. 4) can be generated using the `-pg` option.

3.1.1 Contig Scaling. Contigs' sizes are scaled in the displayed graph by their length. Due to the wide range of contig sizes found within a typical assembly, MetagenomeScope scales contigs such that the area occupied by a contig's symbol in a visualization of a connected component of the assembly graph is proportional to the contig's relative length in that connected component. Contig lengths are also scaled logarithmically beforehand in order to maintain compactness while adequately highlighting differences in contig sizes.[2] In order to help visually elucidate significant relative differences in contig length, MetagenomeScope also adjusts contigs' proportions (keeping area constant) based on their relative length.

3.1.2 Edge Scaling. The thicknesses of edges in the graph are also scaled to highlight the number of links connecting a pair of contigs. As edge weights are difficult to evaluate visually, we also add color to flag edges that are unusually "thick" or unusually "thin" when compared to other edges in a given connected component of the graph. These outlier edges are detected using Tukey fences as detailed in [18], and their thickness is set as either the maximum or minimum thickness available, based on whether the outlier edge in question was a high or low outlier. These outlier edges are also specially colored to indicate their relatively abnormal high or low multiplicity. This approach aids in providing a quick visual indication of which edges have relatively high or low multiplicities, setting MetagenomeScope apart from tools like Bandage [16] which do not incorporate edge weight metadata into the visualizations.

The thicknesses of edges not determined to be outliers in their connected components are set relatively; since outlier edges are not considered in these calculations, relative scaling in this context should be less vulnerable to extreme values skewing the dataset.

3.2 Detection and Highlighting of Structural Patterns

Certain structures in genome assembly graphs correspond to either errors (e.g., "bubbles" caused by sequencing errors) or biological phenomena such as differences between co-assembled haplotypes. As described earlier, the layout algorithm employed by MetagenomeScope can use these patterns to generate a layout that is visually informative. We limit the scope of pattern identification so that single contigs are assigned to at most one pattern. This enables the hierarchical layout of the graph, as each pattern can thus be replaced by a single virtual node (as detailed in section 3.1).

MetagenomeScope automatically detects four types of structural patterns present in many assembly graphs: "bubbles," "frayed ropes," "chains," and "cyclic chains."

Bubbles and frayed ropes indicate regions of the graph exhibiting a diverge-converge or converge-diverge pattern, respectively [10]. Bubbles can indicate sequencing errors or real polymorphisms between co-assembled haplotypes, and frayed ropes can be a signature of repetitive sequences.

We define "chains" as any sequence of two or more unambiguously linked contigs within an assembly graph.

"Cyclic chains" are defined as chains that form a cycle, with the purview of "chain" slightly expanded in this case to include single-contig cycles. These types of patterns potentially indicate

[2]We use a base 10 logarithm by default, although the base is set as a configurable variable (`CONTIG_SCALING_LOG_BASE`) in MetagenomeScope's source code.

the presence of tandem repeats in the underlying sequence, or represent complete circular chromosomes or organelles.

MetagenomeScope can also accept patterns defined by an external file, thereby allowing users of the tool to visualize complex patterns or to study specific biological phenomena that are currently not captured by the patterns described above.

The patterns, whether identified automatically by MetagenomeScope or generated by an external tool, are higlighted with a different background in the graph, and can be dynamically collapsed and uncollapsed by the user in MetagenomeScope's viewer interface. Figs. 5 and 6 demonstrate this functionality, showing how it helps to simplify the region of the graph being visualized.

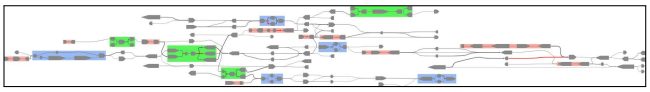

Figure 5: Third largest connected component of a human metagenome assembly graph (accession ID SRS049950), visualized in MetagenomeScope. Bubbles are highlighted blue, frayed ropes are colored green, and chains are colored red.

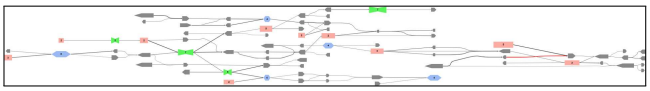

Figure 6: Same component of the SRS049950 assembly graph as in Fig. 5, with all identified structural patterns collapsed.

In addition to their role when laying out the graph, MetagenomeScope uses these identified patterns while drawing the graph in the viewer interface. During the drawing process for a given connected component of a graph, MetagenomeScope first draws the colored bounding boxes of each pattern in the component, followed by all contigs in the component, and ending with all edges in the component. This behavior helps the user obtain some limited understanding of the graph while it is being drawn; for particularly large connected components where displaying the graph can take more than a few seconds, this feature can provide the user with an initial overview of the graph's density of patterns.

3.3 SPQR Tree Decomposition

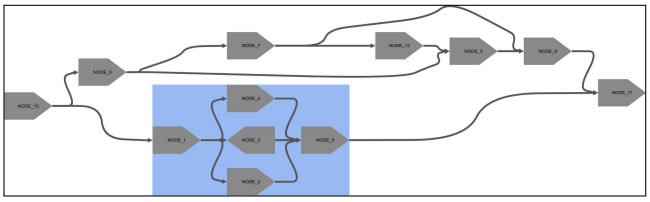

Figure 7: Assembly graph based on Fig. 2a in [13], visualized in MetagenomeScope's "standard mode."

The SPQR tree data structure [1] provides a decomposition of a biconnected graph into its triconnected components. Each triconnected component in the original graph corresponds to an individual node in the SPQR tree (here referred to as "metanodes" in order to alleviate confusion with contigs). SPQR trees have been proposed as a means for hierarchically decomposing assembly graphs [11], as well as for identifying complicated bubble-like regions in these graphs [13].

MetagenomeScope supports the use of SPQR trees as a means for decomposing biconnected components within an assembly graph into smaller, iteratively expandable regions. This functionality is available in MetagenomeScope's SPQR "decomposition mode," a distinct method of visualization from the "standard mode" that is organized around the subgraph patterns described above.

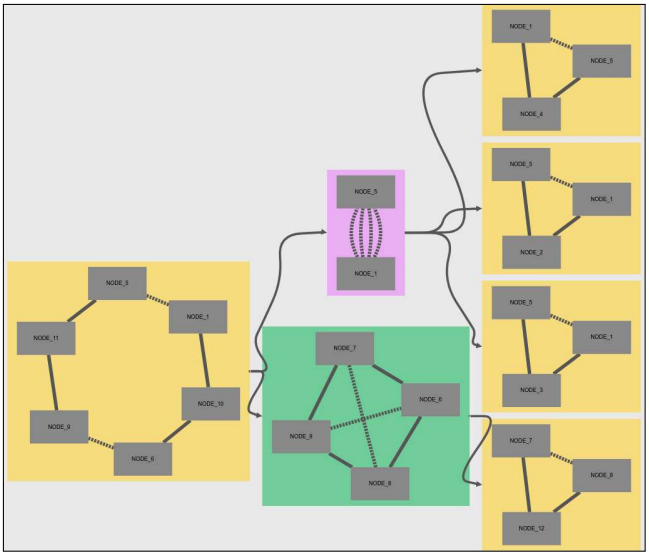

Figure 8: Same graph as in Fig. 7, presented as a fully uncollapsed SPQR tree in MetagenomeScope's explicit SPQR decomposition mode. The children of every non-leaf metanode in the SPQR tree can be iteratively uncollapsed and collapsed in MetagenomeScope's viewer interface, providing a hierarchical means of dynamically altering the graph's complexity.

There are two "sub-modes" of MetagenomeScope's decomposition mode. In both cases, the specified connected component of the assembly graph is drawn—like in MetagenomeScope's standard mode—with the main distinction that every biconnected component in the currently-drawn connected component is collapsed to the root metanode of its respective SPQR tree. These two sub-modes differ in the ways in which these SPQR trees are uncollapsed, and in which information about the biconnected components' structure is thus iteratively revealed.

The first of these sub-modes is "implicit" SPQR tree visualization, in which right-clicking (or tapping with two fingers, for touch-enabled devices) on a metanode within a given biconnected component's SPQR tree merges the child metanodes of the clicked metanode into the currently displayed biconnected component, providing

an iteratively more detailed view of the structure of the biconnected component in question. This sub-mode helps to provide the user with a high-level overview of the graph's structure.

The second of these sub-modes is "explicit" tree visualization. In this sub-mode, right-clicking (or tapping with two fingers) on a metanode within a SPQR tree reveals the child metanodes of the clicked metanode, thus revealing an additional section of the literal SPQR tree structure. Fig. 8 shows an example of a fully uncollapsed tree in MetagenomeScope's explicit SPQR decomposition mode. This sub-mode helps provide a visual explanation of the precise structure of biconnected components' SPQR trees. We posit that this functionality may be useful from a didactic standpoint, when explaining SPQR trees to those unfamiliar with the data structure.

3.4 Scaffold Visualization

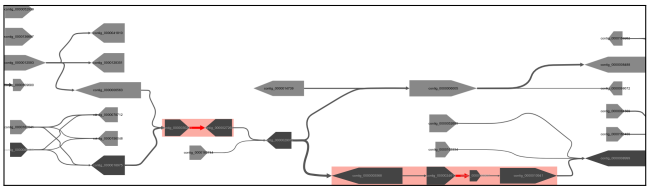

Figure 9: Region of a biofilm assembly graph visualized in MetagenomeScope. Contigs contained within a selected scaffold are colored darker than other contigs to indicate their selection status.

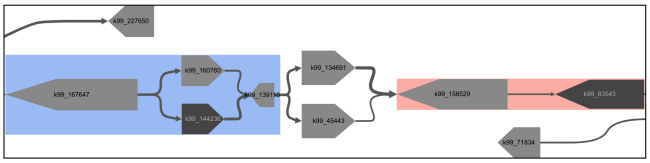

Figure 10: Example of an erroneously generated scaffold in the SRS049950 assembly graph, visualized in MetagenomeScope. The discontiguous nature of the scaffold is immediately apparent from cursory visual inspection.

A scaffold consists of a set of contigs within an assembly graph that are joined together into a path [8]. Scaffolding is usually performed by stand-alone software tools relying on complementary sources of information, such as mate-pairs or physical mapping data. To help users evaluate the correctness of the results produced by such tools, MetagenomeScope allows users to provide scaffold information through an AGP file. The scaffold information is then overlaid onto the graph, as shown in Fig. 9. This information can help identify inconsistencies, such as the situation shown in Fig. 10 where the contigs determined to be adjacent by the scaffolding software are not connected by a valid path within the assembly graph. In this particular example, the visual inspection of the scaffolds allowed us to determine that such mistakes were associated with bubble-like patterns, and closer inspection of the scaffolding code revealed a bug in the way node orientations were processed in such regions.

3.5 Finishing Tools

An assembly graph is an inherently intermediate structure. The goal of assembly is the reconstruction of the complete genome sequence of an organism from the complexity represented within the assembly graph. In most cases, this goal cannot be achieved in a fully automated fashion, and human intervention is necessary, often supported by additional experiments meant to disambiguate the path taken by the genome through the assembly graph. This process is commonly referred to as finishing.

To aid in this endeavor, MetagenomeScope's viewer interface includes functionality that allows the interactive selection of paths through an assembly graph. A path manually selected by the user can then be exported in AGP or CSV format suitable for further processing. Fig. 11 shows a demonstration of a finishing process in progress: immediately after selecting the contig labelled `k99_180779` during a finishing process, many other contigs become available for traversal along the path. Selecting one of these contigs adds that contig to the path and repeats the process at the next branching point in the path. The finishing process ends automatically when no contigs can be added to the path.

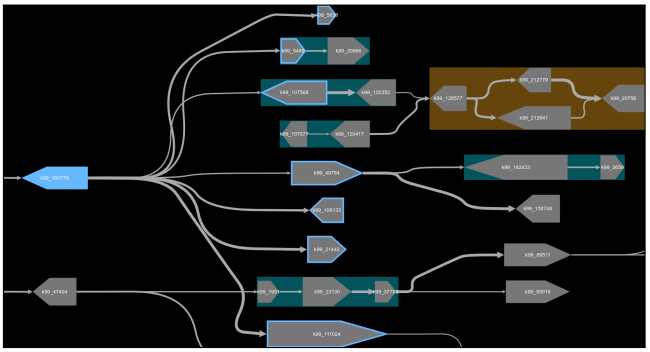

Figure 11: Example of visual feedback provided to the user in MetagenomeScope's interactive finishing tools. The contigs with a light blue border represent alternate paths originating from a specific contig, which is also colored light blue to indicate its "visited" status in the path. This figure also demonstrates MetagenomeScope's configurability in its color settings: all of the colors used in the viewer interface's graph visualizations can be modified, and these settings can be exported and imported in order to be reused. Here, the viewer interface's default colors have been inverted using MetagenomeScope's "Invert all color settings" option.

In the case that the user selects a contig that starts an unambiguous path, MetagenomeScope automatically pursues the path forward as far as possible until a branch is reached. This "autofinishing" functionality drastically reduces the amount of user effort involved in resolving a path, only requiring the user to provide input at branches in the graph rather than at every contig in a path.

We note that Bandage supports similar functionality for selecting and exporting paths of contigs; however, it offers little visual guidance during the path construction process regarding the contigs that are "available" to extend a path. Furthermore, Bandage does

not support autofinishing, and requires the user to interact with every contig added to a path.

4 IMPLEMENTATION

MetagenomeScope is composed of two software components: a command-line preprocessing script that performs layout and structural pattern detection on an input assembly graph, producing a SQLite database file, and a client-side web "viewer interface" that can visualize these database files.

This modular design provides an advantage in the analysis of large assembly graphs. Database files generated by the preprocessing script can be visualized an arbitrary number of times, without repeatedly incurring the computational costs of layout and structural pattern detection. Additionally, the viewer interface and the database files output by the preprocessing script can be hosted on a server, allowing end users to view assembly graphs using only a web browser.

4.1 Preprocessing Script: Laying Out the Graph

MetagenomeScope's preprocessing script highlights and groups together contigs contained in basic structural patterns in the assembly graph, and performs layout on the graph's connected components using Graphviz' [6] *dot* tool for hierarchical layout. Many hierarchical layout algorithms, including *dot*, use heuristics to circumvent the inherent intractability of Sugiyama-style hierarchical graph drawing [4]. Although these layout approaches work relatively quickly for most small graphs, on large graphs they can take a longer amount of time compared to other layout algorithms such as the force-directed algorithm employed by Bandage [16]. We posit, however, that the relative quality of hierarchical layouts in many cases justifies this increase in computation time.

The preprocessing script is primarily written in Python, with some C++ code that interfaces with the Open Graph Drawing Framework [3] to generate SPQR trees. The script supports Linux and macOS systems.

4.2 Viewer Interface: Supporting Interaction with the Graph

MetagenomeScope's web viewer interface uses Cytoscape.js [5] to visualize assembly graphs described by database files generated by its preprocessing script. Database files are processed on the client side using sql.js [9]. The entirely client-side nature of MetagenomeScope's viewer interface reduces the need for any involved server-side operations, moving the onus of computation to the user's web browser. This has the effect of mitigating the costs of actually hosting an instance of MetagenomeScope's viewer interface, making collaborative visualization of assembly graphs easier. This also allows MetagenomeScope's viewer interface to be used without being hosted on a server—if it is downloaded to a device, it can be accessed as a local file from a web browser on that device.

In addition to standard controls for interactive graph manipulation such as panning, zooming, and selection, the viewer interface includes a variety of novel controls to support exploratory analysis of the assembly graph; many of these are discussed in section 3.

The viewer interface supports modern internet browsers on ordinary computers and on smartphones/tablets.

4.3 Source Code and License

MetagenomeScope's source code is released under the GNU General Public License (GPL), version 3. The source code is publicly available on GitHub at https://github.com/marbl/MetagenomeScope.

5 CONCLUSION

We have presented MetagenomeScope, a tool for visualizing and interacting with assembly graphs. MetagenomeScope relies on a hierarchical layout algorithm in order to visualize graphs in a way that captures the linear structure of genomic graphs and highlights graph patterns that may represent biological features.

MetagenomeScope also provides many novel features to augment exploratory analysis of these graphs. Structural pattern collapsing, scaffold visualization, manual finishing controls, and SPQR tree decomposition—to name a few such tools—provide functionality not currently available in other tools for visualizing genome assembly graphs.

MetagenomeScope is, however, just a first step towards better interactive visualizations of assembly graphs. The ability to detect a richer set of graph patterns and to hierarchically organize them would allow for a multi-level representation able to simultaneously capture the large-scale structure of the graph and drill down to explore interesting motifs. Better layout algorithms developed specifically for handling the unique characteristics of the assembly graphs could reduce the computational cost of layout operations, perhaps even allowing their execution directly in the browser. Further performance improvements could be obtained through parallelization or through the use of special hardware such as GPU processors.

Metagenomic assembly is an inherently difficult process. Although it is likely to remain a hard problem for quite some time, we believe that visualization tools like MetagenomeScope can provide a means of simplifying the process—and a means of helping users explore their data—in the meantime.

ACKNOWLEDGMENTS

The authors were supported in part by the NIH, grant R01-AI-100947, the NSF, grant IIS-1117247, and the Navy Research Laboratories, cooperative agreement N00173162C001, all to MP.

Travel by MF to the 25th International Symposium on Graph Drawing and Network Visualization to present on this work was supported by the Rita Colwell Travel Fellowship.

REFERENCES

[1] G. Di Battista and R. Tamassia. 1989. Incremental planarity testing. In *30th Annual Symposium on Foundations of Computer Science*. 436–441. https://doi.org/10.1109/SFCS.1989.63515

[2] Sébastien Boisvert, François Laviolette, and Jacques Corbeil. 2010. Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. *Journal of computational biology* 17, 11 (2010), 1519–1533.

[3] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. 2014. The Open Graph Drawing Framework (OGDF). In *Handbook of Graph Drawing and Visualization*, Roberto Tamassia (Ed.). CRC Press, Chapter 17.

[4] Markus Eiglsperger, Martin Siebenhaller, and Michael Kaufmann. 2004. An efficient implementation of Sugiyama's algorithm for layered graph drawing. In *International Symposium on Graph Drawing*. Springer, 155–166.

[5] Max Franz, Christian T. Lopes, Gerardo Huck, Yue Dong, Onur Sumer, and Gary D. Bader. 2016. Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics* 32, 2 (2016), 309. https://doi.org/10.1093/bioinformatics/btv557

[6] Emden R Gansner and Stephen C North. 2000. An open graph visualization system and its applications to software engineering. *Software: Practice and Experience* 30, 11 (2000), 1203–1233.

[7] Elenie Godzaridis, Sebastien Boisvert, Fangfang Xia, Mikhail Kandel, Steve Behling, Bill Long, Carlos P Sosa, François Laviolette, and Jacques Corbeil. 2013. Human Analysts at Superhuman Scales: What Has Friendly Software To Do? *Big data* 1, 4 (2013), 227–236.

[8] Sergey Koren, Todd J. Treangen, and Mihai Pop. 2011. Bambus 2: scaffolding metagenomes. *Bioinformatics* 27, 21 (2011), 2964. https://doi.org/10.1093/bioinformatics/btr520

[9] Ophir Lojkine, Alon Zakai, et al. 2017. sql.js. http://github.com/kripken/sql.js. (2017).

[10] Jason R. Miller, Sergey Koren, and Granger Sutton. 2010. Assembly algorithms for next-generation sequencing data. *Genomics* 95, 6 (2010), 315–327. https://doi.org/10.1016/j.ygeno.2010.03.001

[11] Gene Myers, Mihai Pop, Knut Reinert, and Tandy Warnow. 2017. Next Generation Sequencing (Dagstuhl Seminar 16351). *Dagstuhl Reports* 6, 8 (2017), 91–130. https://doi.org/10.4230/DagRep.6.8.91

[12] C. B. Nielsen, S. D. Jackman, I. Birol, and S. J. M. Jones. 2009. ABySS-Explorer: Visualizing Genome Sequence Assemblies. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (Nov 2009), 881–888. https://doi.org/10.1109/TVCG.2009.116

[13] Jurgen F. Nijkamp, Mihai Pop, Marcel J. T. Reinders, and Dick de Ridder. 2013. Exploring variation-aware contig graphs for (comparative) metagenomics using MaryGold. *Bioinformatics* 29, 22 (2013), 2826.

[14] Adam M. Phillippy, Michael C. Schatz, and Mihai Pop. 2008. Genome assembly forensics: finding the elusive mis-assembly. *Genome Biology* 9, 3 (2008), R55. https://doi.org/10.1186/gb-2008-9-3-r55

[15] Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones, and Inanç Birol. 2009. ABySS: a parallel assembler for short read sequence data. *Genome research* 19, 6 (2009), 1117–1123.

[16] Ryan R. Wick, Mark B. Schultz, Justin Zobel, and Kathryn E. Holt. 2015. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics* 31, 20 (2015), 3350. https://doi.org/10.1093/bioinformatics/btv383

[17] Daniel R Zerbino and Ewan Birney. 2008. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research* 18, 5 (2008), 821–829.

[18] Qi Zhou, Shaonan Li, Xiaopeng Li, Wei Wang, and Zhiguo Wang. 2006. Detection of outliers and establishment of targets in external quality assessment programs. *Clinica chimica acta* 372, 1-2 (2006), 94–97.