Title

Walk-Preserving Transformation of Overlapped Sequence Graphs into Blunt Sequence Graphs with GetBlunted

ISBN

978-3-030-80048-2

Authors

Eizenga, Jordan M
Lorig-Roach, Ryan
Meredith, Melissa M
et al.

Publication Date

2021

DOI

10.1007/978-3-030-80049-9_15

Walk-preserving transformation of overlapped sequence graphs into blunt sequence graphs with GetBlunted

Jordan M. Eizenga*[0000−0001−8345−8356], Ryan Lorig-Roach*[0000−0002−8183−9611], Melissa M. Meredith[0000−0001−5736−3193], and Benedict Paten[0000−0001−8863−3539]

University of California Santa Cruz Genomics Institute
1156 High Street, Santa Cruz, CA 95064
bpaten@ucsc.edu

* Contributed equally

Abstract. Sequence graphs have emerged as an important tool in two distinct areas of computational genomics: genome assembly and pangenomics. However, despite this shared basis, subtly different graph formalisms have hindered the flow of methodological advances from pangenomics into genome assembly. In genome assembly, edges typically indicate overlaps between sequences, with the overlapping sequence expressed redundantly on both nodes. In pangenomics, edges indicate adjacency between sequences with no overlap—often called *blunt* adjacencies. Algorithms and software developed for blunt sequence graphs often do not generalize to overlapped sequence graphs. This effectively silos pangenomics methods that could otherwise benefit genome assembly. In this paper, we attempt to dismantle this silo. We have developed an algorithm that transforms an overlapped sequence graph into a blunt sequence graph that preserves walks from the original graph. Moreover, the algorithm accomplishes this while also eliminating most of the redundant representation of sequence in the overlap graph. The algorithm is available as a software tool, GetBlunted, which uses little enough time and memory to virtually guarantee that it will not be a bottleneck in any genome assembly pipeline.

Keywords: Genome assembly · Graph genome · Pangenomics.

1 Introduction

Genome assembly is the process of determining a sample's full genome sequence from the error-prone, fragmentary sequences produced by DNA sequencing technologies. Sequence graphs have a long history of use in this field [16, 20, 17]. In these graphs, nodes are labeled with sequences derived from sequencing data, and edges indicate overlaps between observed sequences, which may in turn indicate adjacency in the sample's genome (Fig. 1A). The sample genome then

corresponds to some walk through graph. There are several specific sequence graph articulations in wide use, including de Bruijn graphs, overlap graphs, and string graphs. They each present computational and informational trade-offs that make them better suited to certain configurations of sequencing technologies and genome complexity.

The common topological features of genome assembly graphs are driven primarily by the repetitiveness of the underlying genomes. In many species, a large fraction of the genome consists of repeats (for instance, more than 50% of the human genome [11]). Because all copies of a repeat are highly similar to each other, the corresponding nodes in the sequence graph frequently overlap each other. In contrast, the unique regions of the genome have few erroneous overlaps. These two factors tend to create graphs that consist of long non-branching paths (corresponding to the unique regions), which meet in a densely tangled core with a complicated topology (corresponding to the repeats).

Recently, sequence graphs have also emerged into prominence in the growing field of pangenomics, which seeks to analyze the full genomes of many individuals from the same species [4]. In pangenomics, sequence graphs are used to represent genomic variation between individual haplotypes. Sequences in the graph furcate and rejoin around sites of variation so that each individual genome corresponds to a walk through the graph (Fig. 1B). The growth of pangenomics has fueled major advances in both formal algorithms research [21, 12] and practical genomics tools [10, 22].

Pangenome graphs have much simpler topologies than genome assembly graphs. Having fuller knowledge of the constituent genomes makes it possible to distinguish different copies of a repeat. Thus, pangenome graphs tend to be mostly non-branching, much like the portions of assembly graphs that correspond to unique sequences in the genome. Moreover, most of the branching in pangenome graphs consists of localized bubble-like motifs. In contrast to assembly graphs, pangenome graphs have few if any cycles.

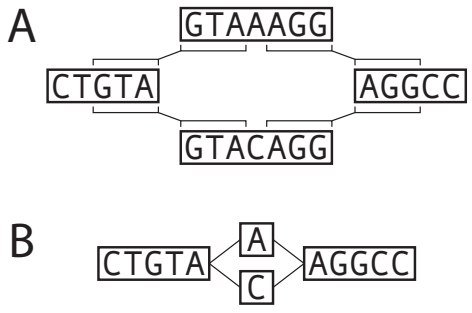

Fig. 1. A: An overlapped sequence graph. **B**: A blunt sequence graph.

Intuitively, the shared basis in sequence graphs should permit the advances in pangenomics to spill over into genome assembly. However, such cross-pollination is stymied by a small difference in the graph formalisms. The edges in assembly graphs indicate sequence overlaps, which are necessary because of the uncertain adjacencies in the underlying genome. In pangenome graphs, the underlying genomes are known, and the edges are *blunt* in that they indicate direct adjacency with no overlap. Blunt sequence graphs can be trivially converted into overlap graphs (with overlaps of length 0), but the reverse requires nontrivial merging operations between the overlapping sequences. As a result, methods have remained siloed within pangenomics despite potential uses in genome assembly.

In this work, we present a method to transform an overlapped sequence graph into a blunt sequence graph. We state the formal guarantees of our formulation and discuss their computational complexity. We then present an algorithm and compare its results to similar methods.

2 Problem statement

In transforming an overlapped sequence graph to a blunt one, we seek to provide two guarantees:

1. All walks in the overlapped graph are preserved in the blunt graph.
2. Every walk in the blunt graph corresponds to some walk in the overlapped graph.

These two properties prohibit the intuitive solution of transitively merging all overlapped sequences. Doing so can result in walks that are not present in the overlapped graph, because walks can transition between nodes that are not connected by an edge via the transitively merged sequences (Fig. 2). Because overlapped sequences cannot be fully merged, it is necessary to retain multiple copies of some sequences in the blunt graph. However, excessive duplication can create problems for downstream analysis, for instance by increasing alignment uncertainty. Thus, we add one further criterion to the above formulation:

3. Minimize the amount of duplicated sequence.

3 Notation

An overlapped sequence graph consists of a set of sequences S and a set of overlaps $O \subset (S \times \{+,-\} \times S \times \{+,-\})$. In this notation, the symbols $+$ and $-$ indicate whether the overlap involves a prefix or suffix (collectively *affix*) of the sequence. This makes the overlapped graph a bidirected graph.

In a bidirected graph, a *walk* consists of a sequence of nodes $s_1 s_2 \dots s_N$, $s_i \in S$ such that 1) each pair of subsequent nodes is connected by an overlap and 2) if s_{i-1} and s_i are connected by an overlap on s_i's prefix, then s_i and s_{i+1}

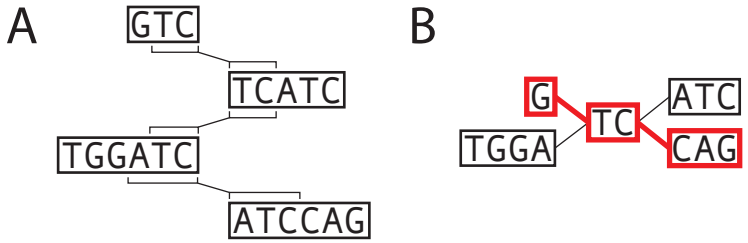

Fig. 2. A: An overlapped sequence graph, and **B**: the blunt sequence graph that results from transitively merging its overlaps. The highlighted walk in the blunt graph does not correspond to any walk in the original overlapped graph.

are a connected by an overlap on s_i's suffix (or vice versa). In the case that a walk traverses a node $s \in S$ from suffix to prefix, we interpret the sequence as its *reverse complement*, which is the sequence of the antiparallel strand of the DNA molecule.

Finally, an *adjacency component* is a collection of affixes (in $S \times \{+, -\}$) that can reach each other via a sequence of adjacent overlaps in O (Fig. 3). This sequence need not form a valid bidirected walk.

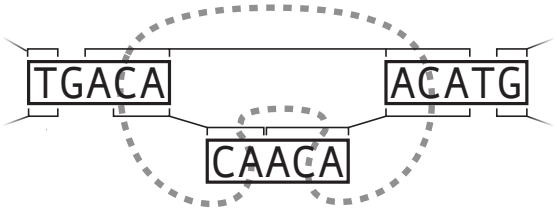

Fig. 3. An adjacency component in a larger sequence graph. Each of the indicated affixes can reach the others by a sequence of overlaps.

4 Methods

To minimize the amount of duplicated sequence, overlapped sequences must be merged. However, we have already mentioned that our criteria prohibit transitively merging all overlaps. We must then minimize the total number of groups within which overlaps are merged transitively, which coincides with the number of times the sequences need to be duplicated.

Consider a group of overlaps that contains $(s_1, s_2, +, -)$ and $(t_1, t_2, +, -)$. For merging to not introduce any walks that are not in the overlapped graph, the

overlaps $(s_1, t_2, +, -)$ and $(t_1, s_2, +, -)$ must also be overlaps in O. Extending this logic, the entire group of overlaps must be contained within a *biclique* subgraph of the adjacency component: two sets of affixes B_1 and B_2 such that every affix in B_1 is connected to every affix in B_2 by an overlap. Thus, we can minimize the number of duplicated sequences by minimizing the number of bicliques needed to cover every overlap edge.

The problem of covering edges with the minimum number of bicliques is known as *biclique cover* (Fig. 4), and it is known to be NP-hard [19]. However, there are domain-specific features of overlapped sequence graphs that often make it tractable to solve large portions of the graph optimally.

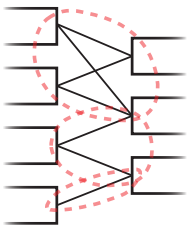

Fig. 4. A biclique cover of an adjacency component with three bicliques.

First, many adjacency components are bipartite. Consider the case that an adjacency component is not bipartite, in which case there is cycle of overlaps between affixes with odd parity. Each overlap indicates high sequence similarity, so an odd cycle means that each sequence is similar to itself, reverse complemented an odd number of times. Such sequences are called DNA palindromes, and they do exist in nature. However, they comprise a small fraction of most real genomes.

Second, most adjacency components are *domino-free*. This property refers to the absence of a particular induced subgraph, the *domino* (Fig. 5). A sufficient condition to prohibit dominoes is for overlapping to be a transitive property. That is, whenever sequence s_1 overlaps sequences t_1 and t_2, and sequence s_2 overlaps t_1, then s_2 also overlaps t_2. In reality, this is not always the case. However, it is very often the case, since overlaps indicate sequence similarity, and similarity is approximately transitive.

These features guided the design of the following algorithm. If an adjacency component is bipartite and domino-free, we compute the biclique cover in polynomial time with the algorithm of Amilhastre, Vilaren, and Janssen [1]. When an adjacency component is bipartite but not domino-free, we instead use the dual graph reduction algorithm of Ene, et al. [7], followed by their lattice-based post-processing if the algorithm does not identify the optimal solution. Finally, if an adjacency component is not bipartite, we first reduce it to the bipartite case by computing an approximate solution to the maximum bipartite subgraph

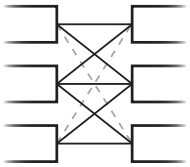

Fig. 5. The domino graph. If either of the dotted edges are present, the induced subgraph is not a domino.

problem using the algorithm of Bylka, Idzik, and Tuza [3]. The maximum bipartite subgraph problem is equivalent to max cut, which is also NP-hard [13]. This process is repeated recursively on the edges that are not included in the bipartite subgraph.

The amount of duplicated sequence is also affected by the manner in which sequences are merged among the overlaps of a biclique. To minimize duplicated sequence, we must maximize matches in the alignment between the overlapped sequences. This is the multiple sequence alignment problem, which is NP-hard. We use the partial order alignment algorithm to approximate the optimal multiple sequence alignment [14]. Partial order alignment also has the advantage that the alignment is expressed as a blunt sequence graph, which can be directly incorporated in the full blunt graph.

5 Implementation

We have implemented the algorithm described here as a genomics tool called GetBlunted. GetBlunted takes as input a GFA file (a common interchange format for sequence graphs [15]) and outputs a GFA containing a blunt graph. In addition, it provides a translation table from sequences in the output to sequences in the input, which can be used to translate analyses performed on the blunt graph into analyses on the overlapped graph. The implementation is written entirely in C++, and it use several auxiliary libraries: GFAKludge is used for manipulating GFA files [5], libbdsg is used to represent sequence graphs [6], and SPOA is used for partial order alignment [24].

6 Results

We compared the performance of GetBlunted to two other tools that transform overlapped sequence graphs into blunt graphs: the gimbricate/seqwish [8, 9] pipeline and Stark [18]. These are, to our knowledge, the only other such tools besides GetBlunted. However, they are not completely comparable. Neither tool provides the guarantees that GetBlunted does for preserving the walk space of the graph. In addition, Stark only works with de Bruijn graphs, a restricted

subset of overlap graphs in which all overlaps are exact matches of a uniform length.

We profiled speed and memory usage on three assembly graphs. The first two are assembly graphs produced by the Shasta assembler [23] for the haploid human cell line CHM13 and for human sample HG002. Both of these were built using Oxford Nanopore reads[1]. The last graph is a de Bruijn graph of Pacific Biosciences HiFi reads of an *Escherichia coli* strain (SRR10382245), which was constructed using jumboDB [2].

All of the bluntifying tools were run on a single core of a c5.9xlarge AWS instance with an Intel Xeon Scalable Processor. Memory usage and compute time were measured with the Unix time tool. The results of the profiling are presented in Table 1. GetBlunted is over 1000 times faster than and comparably memory-intensive to the gimbricate/seqwish pipeline. For de Bruijn graphs, Stark is faster than either tool, although this performance comes at the cost of limited generality.

Assembly	Bluntification Tool	Run Time (min)	RAM (GB)
HG002 Shasta	GetBlunted	0.35	9
	gimbricate/seqwish	917.5	6
CHM13 Shasta	GetBlunted	0.38	4
	gimbricate/seqwish	314.6	6
E. coli de Bruijn	GetBlunted	8.36	26
	gimbricate/seqwish	10.74	4
	Stark	0.65	3

Table 1. Table of speed and memory usage of bluntifing tools run on a single core of an AWS server.

7 Discussion

In this work, we described an algorithm and software tool, GetBlunted, which transforms overlapped sequence graphs into blunt sequence graphs. This provides a route for sequence graph methods developed for pangenomics to be applied to sequence graphs in genome assembly. In both fields, walks through the sequence graph are of primary importance. In genome assembly, some walk through the graph corresponds to the sample genome. In pangenomics, the genomes used to construct the pangenome each correspond to a walk through the graph. GetBlunted provides attractive guarantees that it faithfully preserves the walk space

[1] Publicly available at https://s3-us-west-2.amazonaws.com/miten-hg002/index.html?prefix=guppy_3.6.0/

of the input while also producing parsimonious output. Other comparable methods either do not provide these guarantees or only provide them in limited cases. In addition, GetBlunted is (except in the case of de Bruijn graphs) faster than alternatives that do not provide these guarantees, and it has resource requirements that are easily achievable in any computational environment that is used for genome assembly. In the future, GetBlunted could serve as an step in genome assembly pipelines to improve the quality of their overlap graphs. It could also facilitate direct analyses of assembly graphs in metagenomics applications.

References

1. Amilhastre, J., Vilarem, M.C., Janssen, P.: Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. Discrete Applied Mathematics **86**(2-3), 125–144 (1998)
2. Bankevich, A., Bzikadze, A., Kolmogorov, M., Pevzner, P.A.: Assembling Long Accurate Reads Using de Bruijn Graphs. bioRxiv p. 2020.12.10.420448 (Dec 2020). https://doi.org/10.1101/2020.12.10.420448, https://www.biorxiv.org/content/10.1101/2020.12.10.420448v1, publisher: Cold Spring Harbor Laboratory Section: New Results
3. Bylka, S., Idzik, A., Tuza, Z.: Maximum cuts: Improvements and local algorithmic analogues of the Edwards-Erdos inequality. Discrete Mathematics **194**(1-3), 39–58 (1999)
4. Computational Pan-Genomics Consortium: Computational pan-genomics: status, promises and challenges. Briefings in Bioinformatics **19**(1), 118–135 (2018)
5. Dawson, E.T., Durbin, R.: GFAKluge: A C++ library and command line utilities for the graphical fragment assembly formats. Journal of Open Source Software **4**(33) (2019)
6. Eizenga, J.M., Novak, A.M., Kobayashi, E., Villani, F., Cisar, C., Heumos, S., Hickey, G., Colonna, V., Paten, B., Garrison, E.: Efficient dynamic variation graphs. Bioinformatics (2020)
7. Ene, A., Horne, W., Milosavljevic, N., Rao, P., Schreiber, R., Tarjan, R.E.: Fast exact and heuristic methods for role minimization problems. In: Proceedings of the 13th ACM symposium on Access control models and technologies. pp. 1–10 (2008)
8. Garrison, E.: ekg/gimbricate. https://github.com/ekg/gimbricate (Oct 2020)
9. Garrison, E.: ekg/seqwish. https://github.com/ekg/seqwish (Feb 2021)
10. Garrison, E., Sirén, J., Novak, A.M., Hickey, G., Eizenga, J.M., Dawson, E.T., Jones, W., Garg, S., Markello, C., Lin, M.F., et al.: Variation graph toolkit improves read mapping by representing genetic variation in the reference. Nature Biotechnology **36**(9), 875–879 (2018)
11. Haubold, B., Wiehe, T.: How repetitive are genomes? BMC bioinformatics **7**(1), 1–10 (2006)
12. Jain, C., Zhang, H., Gao, Y., Aluru, S.: On the complexity of sequence to graph alignment. bioRxiv (Jan 2019). https://doi.org/10.1101/522912
13. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Springer (1972)
14. Lee, C., Grasso, C., Sharlow, M.F.: Multiple sequence alignment using partial order graphs. Bioinformatics **18**(3), 452–464 (2002)

15. Li, H., Jackman, S., Myers, E., Gonnella, G., Melsted, P., Turner, I., Heuer, M.L., Wilk, J., Minkin, I., Glusman, G., Shcherbin, E., Garrison, E., Dawson, E., Letcher, B., Huang, S., Bolleman, J.: GFA specification. https://https://github.com/GFA-spec/GFA-spec (2013)
16. Myers, E.W.: Toward simplifying and accurately formulating fragment assembly. Journal of Computational Biology **2**(2), 275–290 (1995)
17. Myers, E.W.: The fragment assembly string graph. Bioinformatics **21**(suppl 2), ii79–ii85 (2005)
18. Nikaein, H.: hnikaein/stark. https://github.com/hnikaein/stark (Jan 2021)
19. Orlin, J., et al.: Contentment in graph theory: covering graphs with cliques. In: Indagationes Mathematicae (Proceedings). vol. 80, pp. 406–424. North-Holland (1977)
20. Pevzner, P.A., Tang, H., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences **98**(17), 9748–9753 (2001)
21. Rautiainen, M., Marschall, T.: Aligning sequences to general graphs in O(V+ mE) time. bioRxiv p. 216127 (2017)
22. Rautiainen, M., Marschall, T.: GraphAligner: rapid and versatile sequence-to-graph alignment. Genome Biology **21**(1), 1–28 (2020)
23. Shafin, K., Pesout, T., Lorig-Roach, R., Haukness, M., Olsen, H.E., Bosworth, C., Armstrong, J., Tigyi, K., Maurer, N., Koren, S., Sedlazeck, F.J., Marschall, T., Mayes, S., Costa, V., Zook, J.M., Liu, K.J., Kilburn, D., Sorensen, M., Munson, K.M., Vollger, M.R., Monlong, J., Garrison, E., Eichler, E.E., Salama, S., Haussler, D., Green, R.E., Akeson, M., Phillippy, A., Miga, K.H., Carnevali, P., Jain, M., Paten, B.: Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes. Nature Biotechnology **38**(9), 1044–1053 (Sep 2020). https://doi.org/10.1038/s41587-020-0503-6, https://www.nature.com/articles/s41587-020-0503-6, number: 9 Publisher: Nature Publishing Group
24. Vaser, R., Sović, I., Nagarajan, N., Šikić, M.: Fast and accurate de novo genome assembly from long uncorrected reads. Genome Research **27**(5), 737–746 (2017)