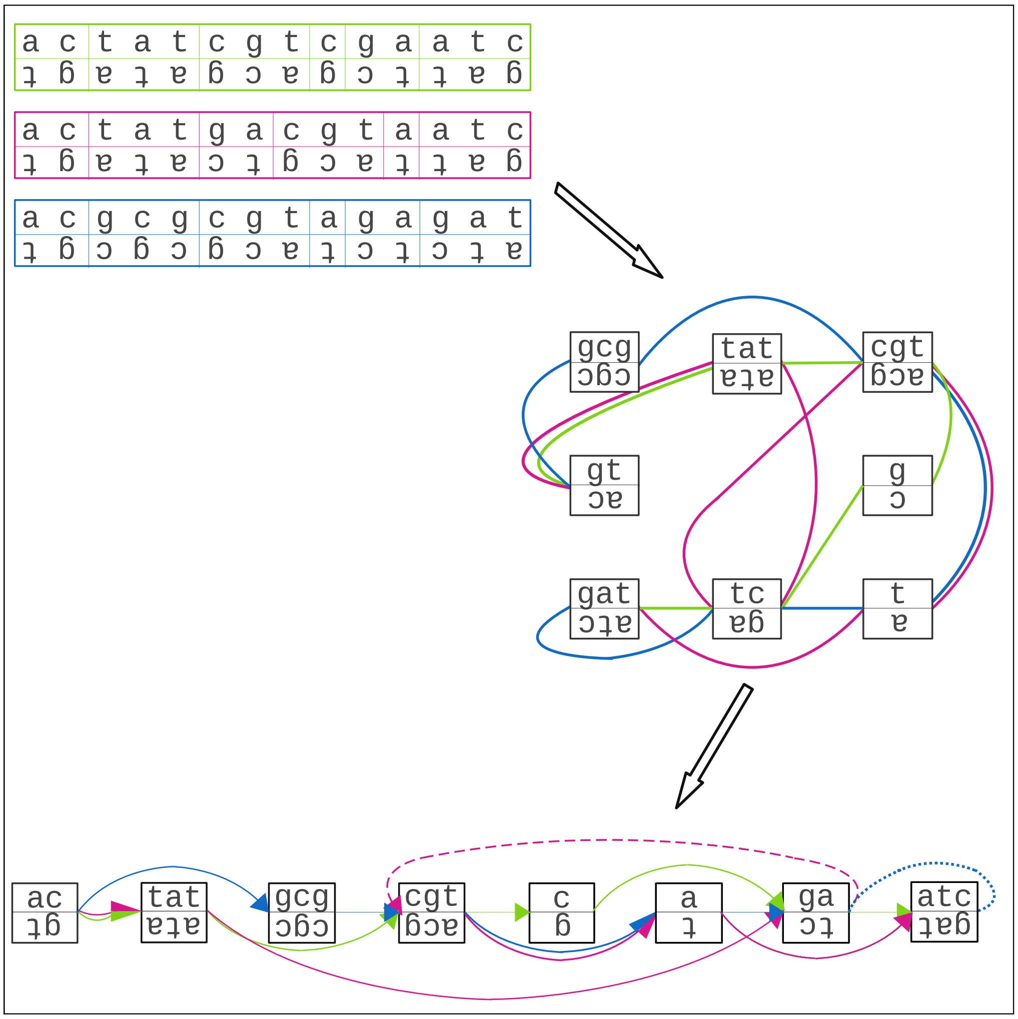
iScience

CellPress
OPEN ACCESS

Article

Linearization of genome sequence graphs revisited



Anna Lisiecka,
Norbert Dojer

dojer@mimuw.edu.pl

Highlights

We propose ALIBI – a new algorithm for linearization of genome sequence graphs

ALIBI yields less feedback arcs and reversing joins than competing methods

ALIBI shows high efficiency and scales well to large graphs

Lisiecka & Dojer, iScience *24*, 102755
July 23, 2021 © 2021 The Authors.
https://doi.org/10.1016/j.isci.2021.102755

iScience

CellPress OPEN ACCESS

Article

Linearization of genome sequence graphs revisited

Anna Lisiecka[1] and Norbert Dojer[1,2,*]

SUMMARY

The need to include the genetic variation within a population into a reference genome led to the concept of a genome sequence graph. Nodes of such a graph are labeled with DNA sequences occurring in represented genomes. Due to double-stranded nature of DNA, each node may be oriented in one of two possible ways, resulting in marking one end of the labeling sequence as in-side and the other as out-side. Edges join pairs of sides and reflect adjacency between node sequences in genomes constituting the graph. Linearization of a sequence graph aims at orienting and ordering graph nodes in a way that makes it more efficient for visualization and further analysis, e.g. access and traversal. We propose a new linearization algorithm, called ALIBI – Algorithm for Linearization by Incremental graph BuIlding. The evaluation shows that ALIBI is computationally very efficient and generates high-quality results.

INTRODUCTION

Reference genomes serve as most important genetic resources for particular populations. They provide coordinate systems for gene annotations, targets for sequencing read mapping and downstream analysis, including variant detection, open chromatin areas and protein binding sites identification, 3-dimensional structure reconstruction, etc. Availability of thousands of individual genomes per species revealed some imperfections of this concept. For example, there are genetic variants that cannot be easily described with respect to the reference genome Horton et al. (2008). Moreover, when used as a target for read mapping, it introduces bias toward the reference alleles Brandt et al. (2015). To overcome these drawbacks, the idea of common representation of a variety of genomes within a population has evolved, leading to the concept of *genome sequence graph*.

Genome sequence graphs are bidirected graphs enhanced with additional structure that allows to represent the relationship within a set of similar genomic sequences (see Figures 1A and 1B). Each node of a sequence graph is labeled with DNA sequence. Each edge is attached to either left or right side of each incident node, representing the 5′ or 3′ end of the sequence, respectively. A directed path *orients* each visited node in one of two possible ways, corresponding to two strands of the DNA fragment labeling the node. Concatenating strand sequences from oriented consecutive nodes on a path yields sequence *represented by* this path. A sequence graph with a collection of its directed paths covering all graph edges represents the set of sequences determined by these paths. Additionally, each edge is assigned a *weight* equal to the number of times it is traversed by paths corresponding to represented genomes.

Genome sequence graphs are an intuitive way to represent the genetic variation, in particular large-scale structural variants (e.g. insertions, deletions, and translocations) in a collection of sufficiently diverse set of genomes. They are applied in several fields, including pangenome modeling (The Computational Pan-Genomics Consortium 2018; Dziadkiewicz and Dojer 2020), improving the quality of read mapping and variant calling (Novak et al., 2017), whole-genome alignment construction (Kehr et al., 2014), or haplotype determination (Paten et al., 2017).

Linearization of a sequence graph aims at reasonable ordering and orienting nodes (see Figure 1C). The result may influence its usability in several aspects, including:

- visual analysis,
- convenience and interpretability of introduced common coordinate system identifying genetic loci of all underlying genomes,
- efficiency of graph-based analysis: searching, simulation, genome comparison etc.

[1]Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland

[2]Lead contact

*Correspondence: dojer@mimuw.edu.pl

https://doi.org/10.1016/j.isci.2021.102755

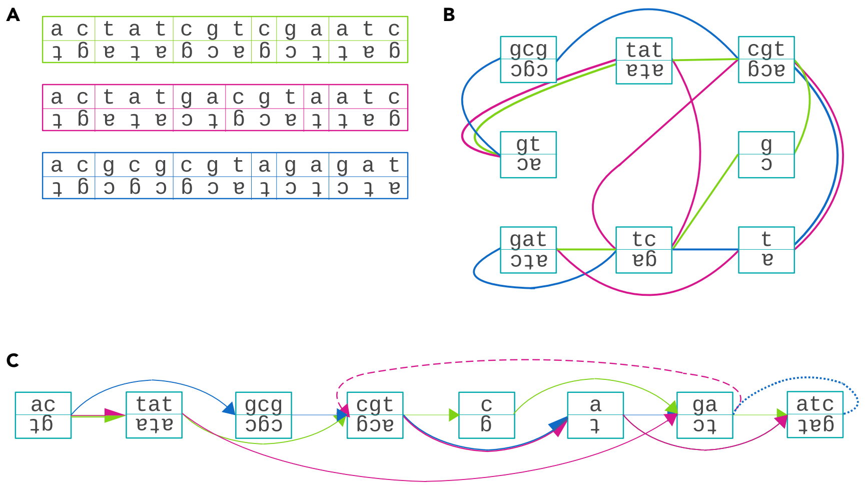


Figure 1. From DNA sequences to linearized sequence graph

(A) Input DNA sequences *ACGCGCGTAGAGAT*, *ACTATCGTCGAATC* and *ACTATGACGTAATC* (each depicted with its reverse complement), divided into fragments aligned with fragments of other sequences.

(B) Sequence graph: nodes represent blocks of aligned DNA fragments, edges join nodes labeled with fragments adjacent in sequences. Paths constituted by green/purple/blue edges represent respective input sequences. For visibility, pairs of nodes labeled with DNA fragments adjacent in multiple sequences are represented by separate edges rather than edge weights.

(C) The same graph after linearization. Nodes are ordered (from left to right) and oriented – the choice of the primary strand results in denominating left sides as in-sides and right sides as out-sides. Consequently, edges are classified as either forward arcs (solid), feedback arcs (dashed) or reversing joins (dotted). The presented linearization minimizes the number of edges of the last two types.

Generally, the more consistent the underlying genomes are with the orientation and ordering of graph nodes, the easier to use the graph is. Ideally, the paths representing genomes should be made up of only *forward arcs*, i.e. edges joining out-sides with in-sides such that the outside node precedes the inside node. Therefore, the quality of linearization may be quantified using the following metrics (see Figure 1C):

- *Weighted feedback arc* (WFA) – the sum of weights of all *feedback arcs*, i.e. backward pointing edges
- *Weighted reversing join* (WRJ) – the sum of weights of all *reversing joins*, i.e. edges joining two in- or two out-sides.

Haussler et al. (2018) proposed an additional metric, called *average cut width* (ACW), i.e. mean number of edges crossing graph cuts placed between any two consecutive nodes (see Figure 2).

A two-step approach to the linearization problem was proposed in the study by Haussler et al. (2018). In the first step, sequence graph is transformed to a directed graph by orienting nodes and ignoring edges joining two in- or two out-sides. In the second step, the nodes of the graph are ordered using either one of well-known heuristics for the feedback arc set problem or an algorithm proposed by authors.

In the current paper, we develop research in this area. We propose a new approach to the linearization problem, in which graph nodes are oriented and ordered jointly. Our algorithm is comprehensively evaluated and compared to previous approaches.

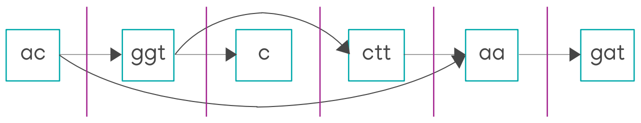


Figure 2. Average cut width (ACW)

Purple vertical lines represent the cuts imposed by the linearization of a sequence graph. Consecutive cuts have width 2, 3, 2, 2, and 1 and the average cut width of this graph is 2.

Algorithm 1. Algorithm for Linearization by Incremental graph BuIlding

```
1.  input genome sequence graph $(V, E)$
2.  output linearized graph $(V', E')$
3.  Set arbitrary orientation on each $v \in V$
4.  $(V', E') \leftarrow (V, \varnothing)$                                ▷ initialize linearized graph
5.  Sort $E$ in descending order of weights
6.  for $e \in E$ do
7.      if $e$ joins nodes belonging to the same connected component then
8.          if $e$ joins two in- or two out-sides then
9.              Add $e$ to $E'$ as reversing join
10.         else                                              ▷ $e$ joins one in- and one out-side
11.             if the out-side node precedes the in-side node then
12.                 Add $e$ to $E'$ as forward arc
13.             else if there is a forward arc path from in-side node to out-side node then
14.                 Add $e$ to $E'$ as feedback arc
15.             else
16.                 Reorder the component such that the out-side node precedes the in-side node
17.                 Add $e$ to $E'$ as forward arc
18.     else                          ▷ $e$ joins nodes belonging to two different connected components
19.         if $e$ joins two in- or two out-sides then
20.             Reverse one of the connected components bridged by $e$
21.         Merge component orders                ▷ out-side node component precedes in-side node component
22.         Add edge $e$ to $E'$ as forward arc
23. end for
```

RESULTS

Overview of Algorithm for Linearization by Incremental graph BuIlding

Tha main idea of our algorithm is to build the linearized graph gradually by adding edges in a decreasing order of weights. We try to establish as many as possible heavy forward arcs and this way reduce both WFA and WRJ. The algorithm starts from the empty graph with arbitrarily oriented nodes and trivially ordered singleton connected components. When a new edge joins two components, their orders are merged. Therefore, at each step of the algorithm, every connected component is separately linearized and, consequently, every edge is classified as reversing join, feedback arc, or forward arc. The pseudocode of Algorithm for Linearization by Incremental graph BuIlding (ALIBI) is given in Algorithm 1.

The justification of our approach, as well as implementation details and complexity analysis, is given in the Methods section. See also Figures 6–8.

Evaluated approaches

We compared ALIBI against two linearization algorithms implemented in vg tool (Garrison et al., 2018). Both vg methods follow the two-step approach proposed by Haussler et al. (2018) and share the algorithm that performs the node orienting step. The node ordering step in these methods are different: Eades is the implementation of a well-known heuristic for the feedback arc set problem of Eades et al. (1993) and FP is the flow procedure proposed in (Haussler et al., 2018) – this algorithm focuses on minimizing average cut width.

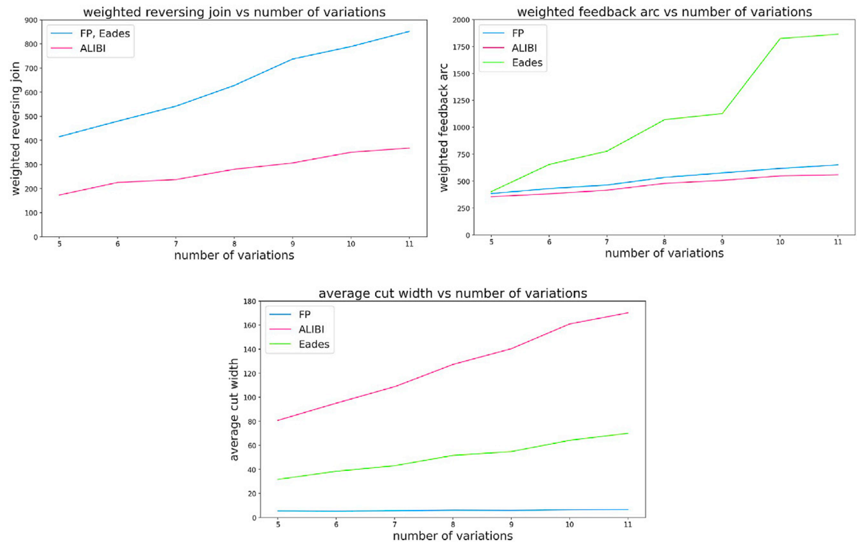


Figure 3. Quality metrics on simulated data: WRJ (top left), WFA (top right) and ACW (bottom)

Performance on simulated data

We prepared simulated data following the procedure of Haussler et al. (2018). Namely, we took a 37287*bp*-long fragment of human genome and applied to it a series of structural variations using the Bioconductor package RSVSim (Bartenhagen and Dugas 2013). Introduced variations included deletions,

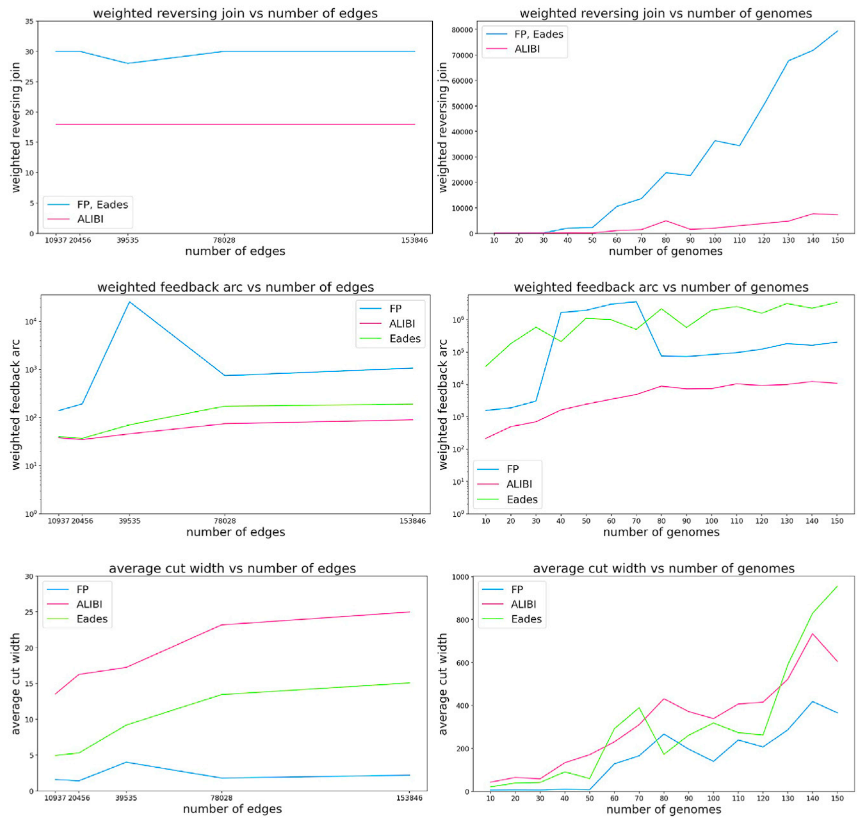


Figure 4. Quality metrics on *E. coli* genomes: WRJ (top), WFA (middle) and ACW (bottom)
Plots present dependence on the size of the graph (left, all graphs are constructed from 4 genomes, GenBank: AP009048, AP012306, CP000948, U00096) and on the number of genomes (right, see Table S1 for the list of genomes).

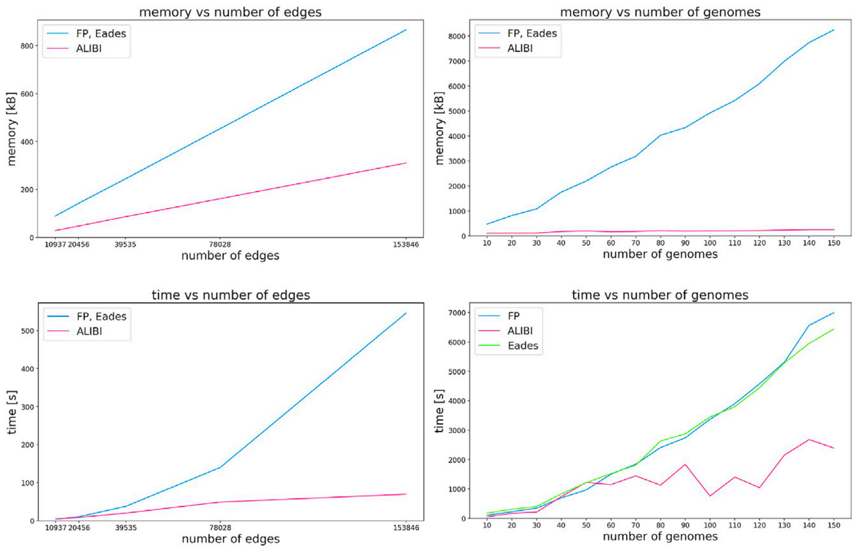


Figure 5. Efficiency of algorithms on *E. coli* genomes: occupied memory (top) and computation time (bottom)
Plots present dependence on the size of the graph (left, all graphs are constructed from 4 genomes, GenBank: AP009048, AP012306, CP000948, U00096) and on the number of genomes (right, see Table S1 for the list of genomes).

insertions, inversions, and duplications of lengths 20, 20, 200, and 500, respectively. Each simulation generated the same number of variations of each type, varying from 5 to 11. For each of these numbers, 10 rearranged genomes were created and passed to the msga command of the vg, which generated a sequence graph in .gfa format.

Figure 3 presents the WRJ, WFA, and ACW results of the algorithms on the simulated data sets. The total weight of reversing joins in ALIBI is approximately half the size of FP and Eades algorithms (both have identical WRJ because they share the algorithm that performs the node orienting step). In terms of feedback arcs, ALIBI is slightly better than FP, while Eades algorithm is surprisingly the weakest. As expected, FP outperforms all other methods in terms of ACW.

Performance on *Escherichia coli* genomes

We also prepared two series of graphs build from real genomes of *Escherichia coli* K-12 strain. In the first series, we took 4 genomes and created graphs using vg msga with the parameter -m restricting the length of sequence labeling nodes set to 32, 64, 128, 256, and 512. In this way, we obtained a series of graphs with similar complexity but varying numbers of nodes and edges. In the second series we build graphs with default parameters of vg msga and the number of *E. coli* genomes varying from 10 to 150 (see Table S1 for the full list of genomes).

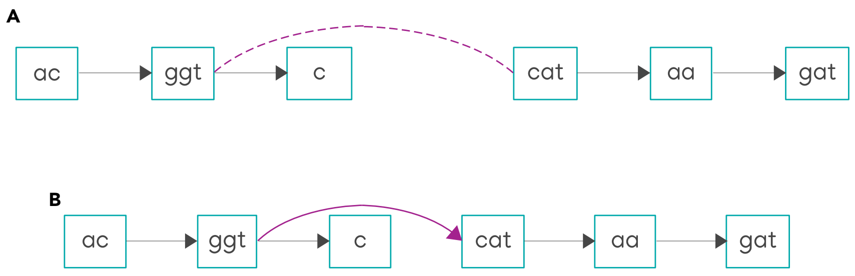


Figure 6. Adding edges joining an out-side with an in-side of nodes from different connected components
(A) The purple edge joins the out-side of the node labeled with "ggt" sequence and the in-side of the node labeled with "cat" sequence.
(B) The edge is added to the graph without any change in node orientations. The nodes from the left component precede the nodes from the right component.

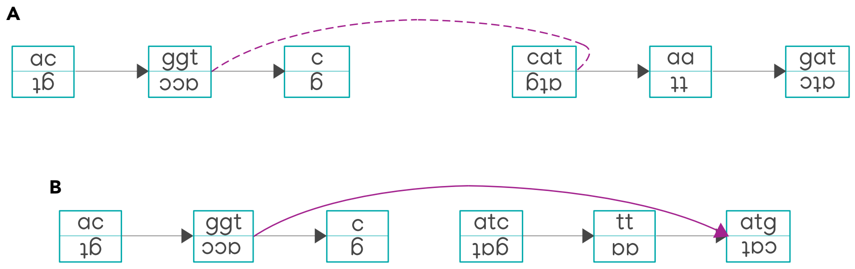


Figure 7. Adding edges joining out-sides of nodes from different connected components

(A) The purple edge joins the out-side of the node labeled with "ggt/acc" sequence and the out-side of the node labeled with "cat/atg" sequence.

(B) After changing the order and orientations of the nodes from the right-hand side connected component (i.e. in-side becomes out-side, out-side becomes in-side, and node sequence changes to its reverse complement), the purple edge joins the out-side of the node labeled with "ggt/acc" sequence and the in-side of the node labeled with "atg/cat" sequence. The purple edge can now be added to the graph.

Figure 4 presents results obtained on *Escherichia coli* data sets. Similarly to simulated data results, FP has the lowest ACW, while ALIBI outperforms the competitors in both WRJ and WFA. However, the differences in WFA are surprisingly extreme here – in some cases, Eades or FP algorithms have WFA over 500 times larger than ALIBI (note the logarithmic scale on the Y axis). Moreover, the relationship between the metrics and the number of genomes is not clear, probably due to highly varying complexity of genome graphs.

Figure 5 summarizes computational efficiency of the algorithms. ALIBI performs best in both experimental settings and with respect to both time and memory resources. All algorithms scale roughly linearly with respect to both number of edges and number of genomes. However, some irregularities are visible in the plots presenting the dependence with respect to the number of genomes, which suggests that the graph complexity significantly affects the computation cost.

DISCUSSION

In the current paper, we proposed ALIBI, a novel linearization algorithm that jointly orients and orders graph nodes. We compared our method with two state of the art algorithms presented in the study by Haussler et al. (2018).

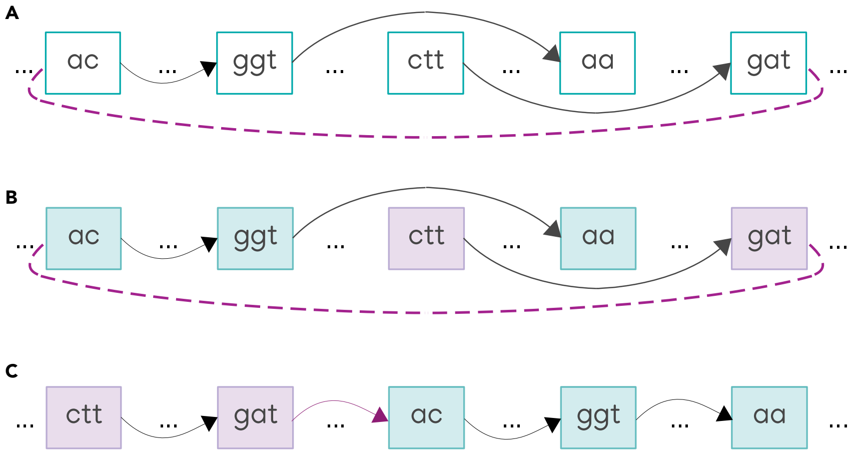


Figure 8. Adding an edge inconsistent with the order on a connected component using Pearce-Kelly algorithm

(A) The purple edge connects the in-side of the node labeled with "ac" sequence and the out-side of the node labeled with "gat" sequence. There is no path from node "ac" to node "gat." Reordering will affect the region between these nodes.

(B) Identification of the nodes from the affected region that are either reachable from node "ac" (light blue nodes) or from which node "gat" is reachable (light purple nodes). The gaps denoted by dots may contain other nodes.

(C) Light purple and light blue nodes are permuted such that all the light purple nodes precede all the light blue nodes and the original order within light purple nodes is preserved, as well as within light blue nodes. Positions of all other nodes remain unaffected.

The evaluation shows that ALIBI substantially outperforms its competitors in two out of three quality metrics: weighted reversing join and weighted feedback arc, while flow procedure of the study by Haussler et al. (2018) achieves best results in terms of average cut widths. Moreover, ALIBI is the fastest and has the lowest memory requirements.

Limitations of the study

The required computational resources of ALIBI and other linearization algorithms depend on the structure of the input graph rather than on the number and size of the underlying genome sequences. Consequently, the computational cost of linearization of graphs build from the same genome data set using different tools may substantially differ.

STAR★METHODS

Detailed methods are provided in the online version of this paper and include the following:

- KEY RESOURCES TABLE
- RESOURCE AVAILABILITY
  - Lead contact
  - Materials availability
  - Data and code availability
- METHOD DETAILS
  - Bidirected graphs and genome sequence graphs
  - Linearization of a bidirected graph
  - Adding edges to the graph
  - Data structure and time complexity

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.isci.2021.102755.

ACKNOWLEDGMENTS

This work was supported by the National Science Centre, Poland, under grant number 2016/21/B/ST6/01471.

AUTHOR CONTRIBUTIONS

Conceptualization, ND; Methodology, ND; Software, AL; Formal Analysis, AL & ND; Investigation, AL; Data Curation, AL; Writing–Original Draft, AL; Writing–Review & Editing, ND; Funding Acquisition, ND.

DECLARATION OF INTERESTS

The authors declare no competing interests.

Received: March 23, 2021
Revised: May 21, 2021
Accepted: June 15, 2021
Published: July 23, 2021

REFERENCES

Bartenhagen, C., and Dugas, M. (2013). RSVSim: an R/Bioconductor package for the simulation of structural variations. Bioinformatics *29*, 1679–1681. https://doi.org/10.1093/bioinformatics/btt198.

Brandt, D.Y.C., Aguiar, V.R.C., Bitarello, B.D., Nunes, K., Goudet, J., and Meyer, D. (2015). Mapping bias overestimates reference allele frequencies at the HLA genes in the 1000 genomes project phase I data. G3 (Bethesda) *5*, 931–941. https://doi.org/10.1534/g3.114.015784.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. (2009). Introduction to Algorithms, 3rd Edition (MIT Press).

Dziadkiewicz, P., and Dojer, N. (2020). Getting insight into the pan-genome structure with PangTree. BMC Genomics *21*, 1–13.

Eades, P., Lin, X., and Smyth, W.F. (1993). A fast and effective heuristic for the feedback arc set problem. Inf. Process. Lett. *47*, 319–323.

Edmonds, J., and Johnson, E.L. (1970). Matching: a well-solved class of integer linear programs. In Combinatorial Structures and Their Applications (Gordon and Breach), pp. 89–92.

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

Haussler, D., Smuga-Otto, M., Eizenga, J.M., Paten, B., Novak, A.M., Nikitin, S., Zueva, M., and Miagkov, D. (2018). A flow procedure for linearization of genome sequence graphs. J. Comput. Biol. *25*, 664–676.

Hopcroft, J.E., and Ullman, J.D. (1973). Set merging algorithms. SIAM J. Comput. *2*, 294–303.

Horton, R., Gibson, R., Coggill, P., Miretti, M., Allcock, R.J., Almeida, J., Forbes, S., Gilbert, J.G., Halls, K., Harrow, J.L., and Beck, S. (2008). Variation analysis and gene annotation of eight MHC haplotypes: the MHC Haplotype Project. Immunogenetics *60*, 1–18. https://doi.org/10.1007/s00251-007-0262-2.

Kehr, B., Trappe, K., Holtgrewe, M., and Reinert, K. (2014). Genome alignment with graph data structures: a comparison. BMC Bioinformatics *15*, 99.

Novak, A.M., Hickey, G., Garrison, E., Blum, S., Connelly, A., Dilthey, A., Eizenga, J., Elmohamed, M.A.S., Guthrie, S., Kahles, A., et al. (2017). Genome graphs. bioRxiv, 101378.

Paten, B., Novak, A.M., Eizenga, J.M., and Garrison, E. (2017). Genome graphs and the evolution of genome inference. Genome Res. *27*, 665–676.

Pearce, D.J., and Kelly, P.H. (2007). A dynamic topological sort algorithm for directed acyclic graphs. J. Exp. Algorithm *11*, 1–7.

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

STAR★METHODS

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Software and algorithms		
vg v1.10.0	Garrison et al. (2018)	https://github.com/vgteam/vg
RSVSim 1.20.0	Bartenhagen and Dugas (2013)	http://www.bioconductor.org/packages/release/bioc/html/RSVSim.html
ALIBI	This paper	https://github.com/anialisiecka/alibi

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources should be directed to and will be fulfilled by the lead contact, Norbert Dojer (dojer@mimuw.edu.pl).

Materials availability

This study did not generate new unique reagents.

Data and code availability

All original code has been deposited at https://github.com/anialisiecka/alibi and is publicly available. Any additional information required to reanalyze the data reported in this paper is available from the lead contact upon request.

METHOD DETAILS

Bidirected graphs and genome sequence graphs

The notion of *bidirected graph* was introduced by Edmonds and Johnson (1970). In a bidirected graph every node has two *sides*, called *left* and *right* and each edge endpoint is incident with one side of a particular node. Formally, each edge in a bidirected graph (V, E) is a tuple (v, s, v', s'), where $v, v' \in V$ and $s, s' \in \{left, right\}$ indicate the incident sides of v and v', respectively. This is conceptually similar to directed graphs, where edges are incident with in- or out-sides of nodes, but in bidirected graphs the sides of edge endpoints are independent (i.e. edges may have one left and one right side, as well as both left or both right sides). Thus we can consider directed graphs as a special case of bidirected graphs.

A *path* in a bidirected graph G is a sequence $e_1, e_2, \ldots, e_k$ of edges such for each pair of consecutive edges $e_i = (v_i, s_i, v_i', s_i')$ and $e_{i+1} = (v_{i+1}, s_{i+1}, v_{i+1}', s_{i+1}')$ we have $v_i' = v_{i+1}$. Path is *directed* if additionally $s_i' \neq s_{i+1}$, i.e. if each node is exited on the other side it is entered. A *connected component* of a bidirected graph is a maximal set of nodes such that each pair of nodes is connected by a (not necessarily directed) path.

Linearization of a bidirected graph

Let $G = (V, E)$ be a bidirected graph and let $V' \subseteq V$. A *linearization* of V' is given by a pair of functions (a, ord), where:

- *nodes orienting function* $a : V' \to \{-1, 1\}$ establishes labeling of node sides as in- or out-sides in the following way:
  - if $a(v) = 1$, then left side of v is labeled as in-side and right side as out-side,
  - if $a(v) = -1$, then left side of v is labeled as out-side and right side as in-side.
- *nodes ordering function* $ord : V' \to \{1, \ldots, |V|\}$ is a bijection establishing the linear order on V' (i.e. node v precedes node v' iff $ord(v) < ord(v')$).

The linearization implies the following classification of edges joining V'-nodes (see Figure 2 (left)):

- *reversing joins* are edges incompatible with node orientations, i.e. joining two in-sides or two out-sides,
- *feedback arcs* are compatible with node orientations but incompatible with their order, i.e. they join out-side of a node v with in-side of a node v' satisfying $ord(v) \geq ord(v')$,
- *forward arcs* are compatible with both orientations and order, i.e. they join out-side of a node v with in-side of a node v' satisfying $ord(v) < ord(v')$.

If the linearized graph has no reversing joins, it is actually a directed graph with sorted nodes. If, additionally, it has no feedback arcs, it is topologically sorted directed acyclic graph.

Adding edges to the graph

The following two theorems give the conditions, under which a new edge may be classified as forward arc.

Theorem 1. Let G_1 and G_2 be two different linearized connected components with sets of nodes V_1 and V_2, respectively. Assume that a new edge e joining a V_1-node with a V_2-node is added to the graph. Then there exists a linearization of $V_1 \cup V_2$ in which e is classified as forward arc and the classification of all edges in both G_1 and G_2 does not change.

Proof. Let (a_1, ord_1) and (a_2, ord_2) be the given linearizations of V_1 and V_2, respectively. We consider two cases:

1. Edge e joins one out- and one in-side. Without loss of generality we may assume that e joins an out-side of a node $v_1 \in V_1$ with an in-side of a node $v_2 \in V_2$. Then the linearization (a, ord) of $V_1 \cup V_2$ given by formulas

$$a(v) = \begin{cases} a_1(v) & \text{if } v \in V_1 \\ a_2(v) & \text{if } v \in V_2 \end{cases} \quad ord(v) = \begin{cases} ord_1(v) & \text{if } v \in V_1 \\ |V_1| + ord_2(v) & \text{if } v \in V_2 \end{cases}$$

satisfies the requirements of the theorem (see Figure 6).

2. Edge e joins two out- or two in-sides. In order to make e a forward arc we have to reverse the order and orientation of nodes in either component (note that reversing doesn't affect the classification of its inner edges). If e joins out-sides of $v_1 \in V_1$ and $v_2 \in V_2$, the following linearization (a, ord) of $V_1 \cup V_2$ does the job (see Figure 7):

$$a(v) = \begin{cases} a_1(v) & \text{if } v \in V_1 \\ -a_2(v) & \text{if } v \in V_2 \end{cases} \quad ord(v) = \begin{cases} ord_1(v) & \text{if } v \in V_1 \\ |V_1| + |V_2| - ord_2(v) + 1 & \text{if } v \in V_2 \end{cases}$$

The case with two in-sides can be handled analogously.

Consider a linearized graph $G = (V, E)$ and two nodes $v_1, v_2 \in V$. We say that v_2 is *forward-accessible* from v_1 (denoted $v_1 \rightsquigarrow v_2$) iff there is a directed path from v_1 to v_2 consisting of forward arcs only.

Theorem 2. Let (a_c, ord_c) be a linearization on a connected component $G_c = (V_c, E_c)$. Assume that a new edge e joining nodes $v_1, v_2 \in V_c$ such that $ord_c(v_1) \leq ord_c(v_2)$ is added to the graph in the way that doesn't affect the classification of E_c-edges. Then:

1. *If e joins two out- or two in-sides, e must be classified as reversing join.*
2. *If e joins out-side of v_2 with in-side of v_1 and $v_1 \rightsquigarrow v_2$, e must be classified as feedback arc.*
3. *Otherwise e may be classified as forward arc.*

Proof. We will separately prove the three statements from the theorem.

1. Since G_c is a connected component, reversing orientation of one node in V_c without violating edge classification implies reversing orientation of all the other V_c-nodes. Consequently, there is no linearization on G_c that reverses the orientation of only one of the nodes v_1, v_2 and saves the classification of all the E_c-edges.
2. Appending e to the path from the assumption results in a cycle, which cannot consist of forward arcs only. Thus e cannot be classified as forward arc without disturbing the classification of the edges on the path from v_1 to v_2.
3. Since the assumption of the statement 1 is not fulfilled, e joins one in- and one out-side. Since the assumptions of the statement 2 are not fulfilled, $v_2 \neq v_1$ and, consequently, $ord_c(v_1) < ord_c(v_2)$. If e joins out-side of v_1 with in-side of v_2, linearization (a_c, ord_c) classifies e as forward arc. Otherwise the nodes must be reordered such that v_2 precedes v_1. This can be done using the approach applied by Pearce and Kelly (2007) in their dynamic algorithm for topologically sorting directed acyclic graphs. The method consists of:

● Identifying two sets of nodes:

$$V_F = \{ v \in V_c \mid v_1 \rightsquigarrow v \wedge ord_c(v) \leq ord_c(v_2) \}$$
$$V_B = \{ v \in V_c \mid v \rightsquigarrow v_2 \wedge ord_c(v_1) \leq ord_c(v) \}$$

Note that V_F and V_B are disjoint, because otherwise we would have $v_1 \rightsquigarrow v_2$.

● Updating the positions of nodes in $V_F \cup V_B$ such that all V_B-nodes precede all V_F-nodes and the original order within both V_F and V_B is preserved (see Figure 8).

The above theorems justify updating the graph with new edges according to the following rules:

1. Edges joining nodes belonging to different connected components are always classified as forward arcs. There are two cases:
   (a) If the edge joins one in- and one out-side, node orientations remain unchanged. Orders on joined components are merged in the way that all the nodes from the component connected with the edge in an out-side precede all the nodes from the other component (see Figure 6).
   (b) If the edge joins two in- or two out-sides, the order and orientations of nodes from one of the connected components are reversed (see Figure 7). After this step, the orders on both components are merged as in the previous case.
2. Edges joining nodes belonging to the same connected component may be classified as forward arcs, feedback arcs or reversing joins. There are three cases:
   (a) If the edge joins two in- or two out-sides then it must be a reversing join. Order and orientations remain unchanged.
   (b) If the edge joins one in- and one out-side and the out-side node precedes the in-side node, the edge is classified as forward arc and no reordering nor reorientation is needed.
   (c) If the edge joins one in- and one out-side and the in-side node precedes the out-side node, the algorithm checks whether there exists a forward arc path from the in-side node to the out-side node. The existence of the path forces the in-side node to precede the out-side node, so the edge is classified as a feedback arc. Otherwise reordering allowing classifying the edge as forward arc is computed using an adapted Pearce-Kelly algorithm (see Figure 8).

Data structure and time complexity

For the purpose of our algorithm we designed a data structure that provides a dynamic representation of node orientations, connected components and orders within components. It is based on the classical disjoint-set data structure of Hopcroft and Ullman (1973), which provides operations:

- FIND-SET(x) – return the representative of the set containing x,
- UNION(x,y) – merge sets containing x and y into a new one.

Implementation using disjoint-set forest with union by rank and path compression yields for both operations amortized cost $\mathcal{O}(\alpha(n))$, where n is the total number of set elements and α is the inverse of the Ackermann function (Cormen et al., 2009).

Connected components are represented by the sets of nodes. The above operations support checking whether the added edge joins nodes from the same or different components and, in the second case, merging components. In order to represent orientations and order we have to augment the disjoint-set structure. Current orientation of a node is encoded as either 1 (same as initial) or -1 (opposite, i.e. implying swapping in- and out-side and reverse complementing the labeling sequence). In order to encode the order within the component, nodes are bijectively assigned integers from some interval $\{i, i+1, \ldots, j\}$, called *positions*.

Adding to each node attributes directly representing its orientation and position within the current component would waste the efficiency of UNION operation, because every node of one of merged components would require at least updating the position attribute. Therefore we introduced attributes that represent these features *locally*, i.e. with respect to nodes' parents in the component tree. Namely, each node has the following attributes:

- *orient* – orientation relative to the parent, either 1 (the same) or -1 (opposite),
- *shift* – preliminary position relative to the parent and orientation, inherited by descendants,
- *reorder_shift* – final position relative to the preliminary position, non-inherited by descendants.

Actual orientation of a node is the product of *orient* attributes of this node and its ancestors in the tree. Actual position of the node is the sum of

- all *shift* attributes of node's ancestors multiplied by their actual orientation and
- the sum of node's *shift* and *reorder_shift* attributes multiplied by its actual orientation.

The attributes are updated during the path compression procedure of the disjoint-set structure. Since updating require constant time per node in the path, the asymptotic cost of the path compression is not affected. Moreover, every calculation of node's orientation or position calls the path compression procedure, so its amortized cost is $\mathcal{O}(\alpha(n))$, where n is the number of nodes in the graph.

When a new edge is added to the graph, determining the respective case from subsection Adding edges to the graph requires calculating component representatives, orientations and positions of the ends of the edge. The subsequent steps modify the structure in the following way:

- In case 1 operation UNION is performed, followed by modifying attributes *shift* and (only in case 1.b.) *orient* in the root node of one component.
- In cases 2.a and 2.b the structure remains unchanged.
- In case 2.c attributes *reorder_shift* are modified in affected nodes when repositioning is required, otherwise the structure remains unaffected.

Consequently, in all cases except 2.c, the total amortized cost of adding the edge to the graph is $\mathcal{O}(\alpha(n))$. Processing case 2.c consists of the following steps:

- Identifying the set V_F of nodes from the affected region that are reachable from the in-side node (i.e. the set of green nodes in Figure 8). To this aim, an adapted depth-first search algorithm is called, which uses only forward arcs that don't exit the affected region. Therefore, for each edge outgoing from each visited node, the algorithm needs to calculate the position of the target node. Consequently, the amortized cost of this step is $\mathcal{O}(|E_F| \cdot \alpha(n))$, where E_F denotes the set of edges outgoing V_F nodes.

- Identifying the set V_B of nodes from the affected region, from which the out-side node is reachable (i.e. the set of blue nodes in Figure 8). Similarly, the amortized cost of this step is $\mathcal{O}(|E_B| \cdot \alpha(n))$, where E_B denotes the set of edges leading to V_B nodes.
- Updating positions of the nodes in $V_F \cup V_B$. This step is dominated by sorting V_F and V_B nodes according to their original positions, which has cost $\mathcal{O}(|V_F|\log|V_F| + |V_B|\log|V_B|)$.

Therefore, the total cost of processing new edge in this case is $\mathcal{O}((|E_F| + |E_B|) \cdot \alpha(n) + |V_F|\log|V_F| + |V_B|\log|V_B|)$ and substantially depends on the size of the affected region and graph structure.