

# Scalable multiple whole-genome alignment and locally collinear block construction with SibeliaZ

Iliia Minkin <sup>1✉</sup> & Paul Medvedev <sup>1,2,3</sup>

Multiple whole-genome alignment is a challenging problem in bioinformatics. Despite many successes, current methods are not able to keep up with the growing number, length, and complexity of assembled genomes, especially when computational resources are limited. Approaches based on compacted de Bruijn graphs to identify and extend anchors into locally collinear blocks have potential for scalability, but current methods do not scale to mammalian genomes. We present an algorithm, SibeliaZ-LCB, for identifying collinear blocks in closely related genomes based on analysis of the de Bruijn graph. We further incorporate this into a multiple whole-genome alignment pipeline called SibeliaZ. SibeliaZ shows run-time improvements over other methods while maintaining accuracy. On sixteen recently-assembled strains of mice, SibeliaZ runs in under 16 hours on a single machine, while other tools did not run to completion for eight mice within a week. SibeliaZ makes a significant step towards improving scalability of multiple whole-genome alignment and collinear block reconstruction algorithms on a single machine.

<sup>1</sup>Department of Computer Science and Engineering, The Pennsylvania State University, 506 Wartik Lab University Park, University Park, PA 16802, USA.

<sup>2</sup>Department of Biochemistry and Molecular Biology, The Pennsylvania State University, 506 Wartik Lab University Park, University Park, PA 16802, USA.

<sup>3</sup>Center for Computational Biology and Bioinformatics, The Pennsylvania State University, 506 Wartik Lab University Park, University Park, PA 16802, USA.

✉email: [ivminkin@gmail.com](mailto:ivminkin@gmail.com)

Multiple whole-genome alignments are the problem of identifying all the high-quality multiple local alignments within a collection of assembled genome sequences. It is a fundamental problem in bioinformatics and forms the starting point for most comparative genomics studies, such as rearrangement analysis, phylogeny reconstruction, and the investigation of evolutionary processes. Unfortunately, the presence of high-copy repeats and the sheer size of the input make multiple whole-genome alignment extremely difficult. While current approaches have been successfully applied in many studies, they are not able to keep up with the growing number and size of assembled genomes<sup>1</sup>. The multiple whole-genome alignment problem is also closely related to the synteny reconstruction problem and to the questions of how to best represent pan-genomes.

There are two common strategies to tackle the whole-genome alignment problem<sup>2</sup>. The first one is based on finding pairwise local alignments<sup>3–7</sup> and then extending them into multiple local alignments<sup>8–11</sup>. While this strategy is known for its high accuracy, a competitive assessment of multiple whole-genome alignment methods<sup>1</sup> highlighted several limitations. First, many algorithms either do not handle repeats by design or scale poorly in their presence, since the number of pairwise local alignments grows quadratically as a function of a repeat's copy number. In addition, many algorithms use a repeat database to mask high-frequency repeats. However, these databases are usually incomplete and even a small amount of unmasked repeats may severely degrade alignment performance. Second, the number of pairwise alignments is quadratic in the number of genomes, and only a few existing approaches could handle more than ten fruit fly genomes<sup>1</sup>. Therefore, these approaches are ill-suited for large numbers of long and complex genomes, such as mammalian genomes in general and the recently assembled 16 strains of mice<sup>12</sup> in particular.

Alternatively, anchor-based strategies can be applied to decompose genomes into locally collinear blocks<sup>13</sup>. These are blocks that are free from nonlinear rearrangements, such as inversions or transpositions. Once such blocks are identified, they can independently be global aligned<sup>13–17</sup>. The problem of constructing blocks from anchors is known as the chaining problem which had been extensively studied in the past<sup>18–20</sup>. All of the methods applicable to datasets consisting of multiple genomes are heuristic since the exact algorithms depend exponentially on the number of genomes. Such strategies are generally better at scaling to handle repeats and multiple genomes since they do not rely on the computationally expensive pairwise alignment.

A promising strategy to find collinear blocks is based on the compacted de Bruijn graph<sup>21–23</sup> widely used in genome assembly. Though these approaches do not work well for divergent genomes, they remain fairly accurate for closely related genomes. For example, Sibelia<sup>23</sup> can handle repeats and works for many bacterial genomes; unfortunately, it does not scale to longer genomes. However, the last three years has seen a breakthrough in the efficiency of de Bruijn graph construction algorithms<sup>24–27</sup>. The latest methods can construct the graph for tens of mammalian genomes in minutes rather than weeks. We therefore believe the de Bruijn graph approach holds the most potential for enabling scalable multiple whole-genome alignments of closely related genomes.

In this paper, we describe an algorithm SibeliaZ-LCB for identifying collinear blocks in closely related genomes. SibeliaZ-LCB is suitable for detecting homologous sequences that have an evolutionary distance to the most recent common ancestor (MRCA) of at most 0.09 substitutions per site. SibeliaZ-LCB is based on the analysis of the compacted de Bruijn graph and uses a graph model of collinear blocks similar to the "most frequent

paths" introduced by<sup>28</sup>. This allows it to maintain a simple, static, data structure, which scales easily and allows simple parallelization. Thus, SibeliaZ-LCB overcomes a bottleneck of previous state-of-the-art de Bruijn graph-based approaches<sup>17,22</sup>, which relied on a dynamic data structure which was expensive to update. Further, we extend SibeliaZ-LCB into a multiple whole-genome aligner called SibeliaZ. SibeliaZ works by first constructing the compacted de Bruijn graph using our previously published TwoPaCo tool<sup>27</sup>, then finding locally collinear blocks using SibeliaZ-LCB, and finally, running a multiple-sequence aligner *spoa*<sup>29</sup> on each of the foundation blocks. To demonstrate the scalability and accuracy of our method, we compute the multiple whole-genome alignment for a collection of recently assembled strains of mice. We also test how our method works under different conditions, including various levels of divergence between genomes and different parameter settings. Our software is freely available at <https://github.com/medvedevgroup/SibeliaZ/>.

## Results

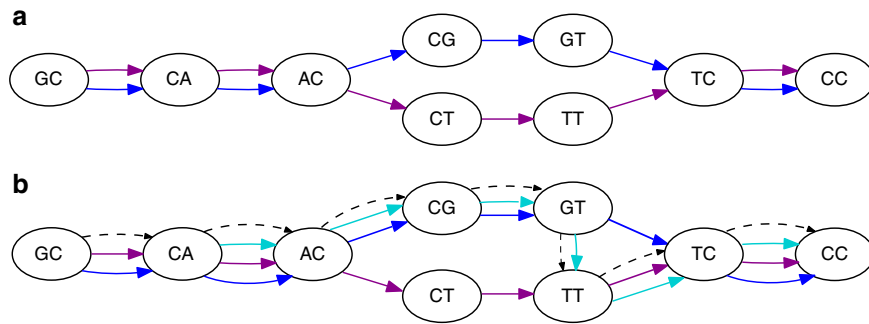
**Algorithm overview.** As described in the introduction, the major algorithmic innovation of this paper is the SibeliaZ-LCB algorithm. SibeliaZ-LCB takes as input a de Bruijn graph built on a collection of assembled genomes. An assembled genome is itself a set of contig sequences. SibeliaZ-LCB identifies and outputs all nonoverlapping blocks of homologous subsequences of the input genomes. A block can be composed of two or more sequences from one or more genomes. In this subsection, we will give a high level overview of SibeliaZ-LCB, leaving the more formal and detailed version for the "Methods".

SibeliaZ-LCB relies heavily on the de Bruijn graph of the genomes. In this graph, the vertices correspond to the  $k$ -mers (substrings of fixed length  $k$ ) of the input. A  $k$ -mer that appears multiple times in the input is represented using just one node. Then,  $k$ -mers that appear consecutively in some input sequence are connected by an edge from the left one to the right one (see Fig. 1a for an example). This way, each genome corresponds to a path in the graph that hops from  $k$ -mer to  $k$ -mer using the edges.

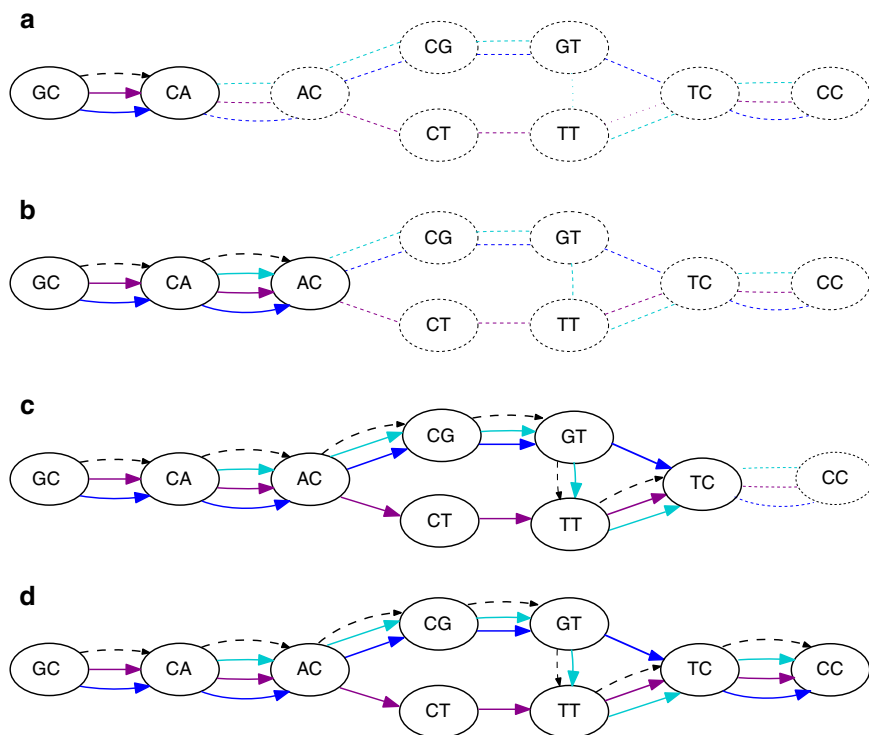
In this graph, two homologous sequences form what is called a chain: an interleaving sequence of parallel edges, which correspond to identical sequences, and "bubbles", which correspond to small variations like single nucleotide variants or indels. However, the concept of a chain is difficult to extend to more than two homologous sequences because the tangled pattern in the graph is difficult to precisely define (see Fig. 1b for an example).

To address this challenge, we introduce the idea that each block has a "carrying path" in the de Bruijn graph that holds the block together. The basic idea is that the homologous sequences forming the block have a lot of shared  $k$ -mers and their corresponding genomic paths go through nearly the same vertices. A carrying path is then a path that goes through the most frequently visited vertices, loosely similar to the notion of a consensus sequence from alignment. Each genomic path from the block then forms a chain with this carrying path (see Fig. 1b for an example).

We do not know the carrying paths in advance but we can use them as a guiding mechanism to find blocks. We start with an arbitrary edge  $e$  in the graph and all other genomic paths that form bubbles with  $e$ . We make  $e$  the starting point of a carrying path and use it along with the other genomic paths to initiate the collection of sequences making up the block corresponding to this carrying path. To extend the carrying path, we look at the edges extending the genomic paths in the current block and take the most common one. The data structures maintaining the genomic paths in the block and the carrying path are then updated and the extension procedure repeats. Figure 2 shows an example of running this algorithm.



**Fig. 1 The de Bruijn graph and an example of a collinear block.** **a** The graph built from strings “GCACGTCC” and “GCACTTCC”, with  $k = 2$ . The two strings are reflected by the blue and magenta walks, respectively. This is an example of a collinear block from two walks. There are four bubbles. The bubble formed by vertices “AC” and “TC” describes a substitution within the block, while three other bubbles are formed by parallel edges. The blue and magenta walk form a chain of four consecutive bubbles. **b** An example of a more complex block, where we have added a third sequence “CACGTTCC” (turquoise) to the input. We can no longer describe the block as a chain of bubbles, as they overlap to form tangled structures. Instead, we consider the path in the graph (dashed black) that shares many vertices with the three collinear walks. This carrying path shares many vertices with the three extant walks, and each walk forms its own chain with it. The task of finding good collinear blocks can then be framed as finding carrying paths that form good chains with the genomic walks.



**Fig. 2 An example of running Algorithm Find-collinear-blocks (Box 1) on the graph from Fig. 1b, starting from edge GC → CC as the seed.** Each subfigure shows the content of the collinear block  $P$  and the carrying path. The collinear walks are solid, the carrying path is dashed, and the rest of the graph is dotted. Subfigure **a** shows the state of these variables after the initialization; subfigures **b–d** show the state after the completion of each phase.

We continue this process until the scoring function that describes how well a carrying path holds the block together falls below zero. At that point, we consider the possibility that we may have overextended the block and should have instead ended it earlier. To do this, we look at all the intermediate blocks we had created during the extension process and output the one that has the highest score. Once a block is an output, we output all its constituent edges as used so that they are not chosen as part of a future block.

In this way, Sibeliaz-LCB finds a single block. Afterwards, we try to find another block by starting from another arbitrary edge.

This process continues until all the edges in the graph are either used or had been tried as potential starters for a carrying path.

**Datasets, tools, and evaluation metrics.** Evaluation of multiple whole-genome aligners is a challenging problem in its own right and we, therefore, chose to use the practices outlined in the Alignathon<sup>1</sup> competition as a starting point. They present several approaches to assess the quality of a multiple whole-genome alignments. Ideally, it is best to compare an alignment against a manually curated gold standard; unfortunately, such a gold

**Table 1** Accession numbers of the assembled mice genomes available at GenBank.

Strain	Accession number
C57BL/6J	GCA_000001635.8
129S1/SvImJ	GCA_001624185.1
A/J	GCA_001624215.1
AKR/J	GCA_001624295.1
CAST/EiJ	GCA_001624445.1
CBA/J	GCA_001624475.1
DBA/2J	GCA_001624505.1
FVB/NJ	GCA_001624535.1
NOD/ShiLtJ	GCA_001624675.1
NZO/HiLtJ	GCA_001624745.1
PWK/PhJ	GCA_001624775.1
WSB/EiJ	GCA_001624835.1
BALB/cJ	GCA_001632525.1
C57BL/6NJ	GCA_001632555.1
C3H/HeJ	GCA_001632575.1
LP/J	GCA_001632615.1

standard does not exist. We, therefore, chose to focus our evaluation on real data.

We evaluated the ability of SibeliaZ to align real genomes by running it on several datasets consisting of a varying numbers of mice genomes. We retrieved 16 mice genomes available at GenBank<sup>30</sup> and labeled as having a “chromosome” level of assembly. They consist of the mouse reference genome and 15 different strains assembled as part of a recent study<sup>12</sup> (Supplementary Table 1). The genomes vary in size from 2.6 to 2.8 Gbp and the number of scaffolds (between 2977 and 7154, except for the reference, which has 377). Their GenBank accession numbers are listed in Table 1. We constructed four datasets of increasing size to test the scalability of the pipelines with respect to the number of input genomes. The datasets contain genomes 1–2, 1–4, 1–8, and 1–16 from Supplementary Table 1, with genome 1 being the reference genome.

To measure accuracy, we used several ground-truth alignments (to be described) and employed the metrics of precision and recall used in the Alignathon and implemented by the mafTools package<sup>1</sup>. For these metrics, alignment is viewed as an equivalence relation. We say that two positions in the input genomes are equivalent if they originate from the same position in the genome of their recent common ancestor. We denote by  $H$  the set of all equivalent position pairs, participating in the “true” alignment. Let  $A$  denote the relation produced by an alignment algorithm. The accuracy of the alignment is then given by recall( $A$ ) =  $1 - |H \setminus A|/|H|$  and precision( $A$ ) =  $1 - |A \setminus H|/|A|$ , where  $\setminus$  denotes set difference.

To evaluate recall, we compared our results against annotations of protein-coding genes. We retrieved all pairs of homologous protein-coding gene sequences from Ensembl and then computed pairwise global alignments between them using LAGAN<sup>31</sup>. The alignment contains both orthologous and paralogous genes, though most of the paralogous pairs come from the well-annotated mouse reference genome. We removed any pairs of paralogous genes with overlapping coordinates, as these were likely mis-annotations, as confirmed by Ensembl helpdesk<sup>32</sup>. We made these filtered alignments as well as the alignments produced by SibeliaZ available for public download from our GitHub repository (see Section “Data availability” for the links).

We define the nucleotide identity of an alignment as the number of matched nucleotides divided by the length of an alignment, including gaps. The distribution of nucleotide identities, as well as

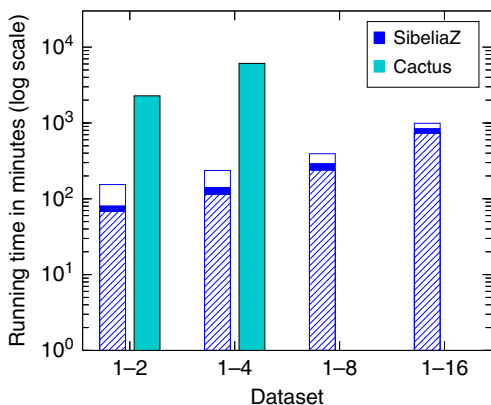
the coverage of the annotation, is shown in Supplementary Fig. 1. In our analysis, we binned pairs of genes according to their nucleotide identity.

Since protein-coding genes only compromise a small portion of the genome, we also computed all-against-all pairwise local alignments between chromosomes 1 of genomes 1–2 and 1–4 using LASTZ<sup>6</sup>, a reliable local aligner known for its accuracy. We only computed alignments between chromosomes of different genomes, i.e., did not include self-alignments, which excludes duplications such as paralogous genes from the alignment. We used default settings of LASTZ except that we made it output alignments of nucleotide identity at least 90%. We then evaluated the recall and precision of our alignments but restricted our alignments to the sequences of chromosome 1. We then treated the LASTZ alignments as the ground truth. The LASTZ alignments are available for download from our repository’s supplemental data section. Note that because the alignment is represented as a set of positions pairs, it is possible to evaluate a multiple whole-genome alignments using pairwise local alignments.

To measure precision, we use the LASTZ alignments on chromosome 1. However, it is computationally prohibitive to compute such alignments with LASTZ for the whole genome. We therefore also use an indirect way to assess precision for the whole genome. For each column in the alignment, we calculate the average number of nucleotide differences<sup>33</sup>. In an alignment of highly similar genomes that has high precision, we expect these numbers to be low (close to 0) for most of the columns in the alignment. Otherwise, it would suggest the presence of unreliable poorly aligned blocks in the alignment. Formally, given a column  $c$  of a multiple whole-genome alignments with  $c_i$  being its  $i$ th element, average number of nucleotide differences is given by  $\pi(c) = \frac{\sum_{1 \leq i \leq |c|} \sum_{i < j \leq |c|} I[c_i \neq c_j]}{\binom{|c|}{2}}$ . The variable  $I[c_i \neq c_j]$  is equal to 1 if both  $c_i$  and  $c_j$  are different valid DNA characters and 0 otherwise;  $|c|$  is the number of rows in column  $c$ .

We benchmarked the performance of SibeliaZ against Progressive Cactus<sup>34</sup>, an aligner based on analysis of the Cactus graphs<sup>35</sup> built from pairwise alignments. We also attempted to run Sibelia<sup>23</sup> (a predecessor of SibeliaZ) and MultiZ + TBA<sup>8</sup>, but these could run to completion within a week on even a single mouse genome. Other multiple aligners<sup>9,10,16</sup> benchmarked in the Alignathon could not handle a dataset of 20 flies and hence are unlikely to scale to a mammalian dataset. We also chose to not run Mercator<sup>14</sup> since it requires a set of gene exons as input and hence solves a different problem: in this paper we focus on computing the whole-genome alignment directly from the nucleotide sequences without using external information. Further details about parameters, versions, and hardware are in Supplementary Note 1 and Supplementary Table 4.

**Running time and memory.** The running times of SibeliaZ and Cactus are shown in Fig. 3 (Supplementary Table 2 contains the raw values). On the dataset consisting of 2 mice, SibeliaZ is more than 10 times faster than Cactus, while on 4 mice SibeliaZ is more than 20 times faster. On the datasets with 8 and 16 mice, SibeliaZ completed in under 7 and 16 h, respectively, while Cactus did not finish (we terminated it after a week). For SibeliaZ, we note that the global alignment with spoa takes 44–73% of the running time, and, for some applications (e.g., rearrangement analysis), this step may be further omitted to save time. Memory is shown in Supplementary Table 2. When it is able to complete, Cactus has better memory performance than SibeliaZ; however, both tools require memory that is well within the range of most modern servers but outside the range of personal computers.



**Fig. 3** Running times of the different pipelines on the mice datasets (on a log scale). Each bar corresponds to a pipeline. The bar of SibeliaZ is split according to its components: spoa (hatch fill), TwoPaCo (solid fill), and SibeliaZ-LCB (empty fill). Cactus is not shown on datasets 1-8 and 1-16 because it did not complete. We used 32 threads for each experiment.

**Table 2** Number of blocks and coverage by the multiple whole-genome alignments computed by SibeliaZ and Cactus from the mice datasets.

Dataset	No. of blocks, SibeliaZ	No. of blocks, Cactus	Coverage, SibeliaZ	Coverage, Cactus
1-2	2,083,258	4,228,063	0.88	0.85
1-4	2,739,821	6,133,662	0.86	0.84
1-8	3,179,619	-	0.89	-
1-16	4,507,109	-	0.88	-

**Table 3** Recall of the orthologous and paralogous basepairs by the multiple whole-genome alignments computed by SibeliaZ and Cactus from the mice datasets, using Ensembl gene annotation as the ground truth. Recall of paralogs by Cactus is not included (see text).

Dataset	Ort. nt. pairs, SibeliaZ	Ort. nt. pairs, Cactus	Par. nt. pairs, SibeliaZ
1-2	0.99	0.99	0.89
1-4	0.98	0.98	0.89
1-8	0.98	-	0.84
1-16	0.98	-	0.83

**Accuracy.** In Tables 2 and 3, we show the properties of the alignments found by SibeliaZ and Cactus. To compute recall, we only used nucleotides from gene pairs having at least 90% identity in the annotation. For the datasets where Cactus was able to complete, SibeliaZ had and similar recall on orthologous pairs. We did not evaluate the results on paralogs by Cactus since it heuristically filters out paralogous alignments<sup>34</sup> as a part of its pipeline. SibeliaZ’s recall decreases only slightly going up to the whole 16 mice dataset, indicating that the recall scales with the number of genomes.

We also measured coverage, which is the percent of the genome sequence that is included in the alignment. The coverage of both tools is roughly the same, but SibeliaZ has only about half the blocks. The different amounts of blocks produced by the tools are likely to be a result of the different approaches to the

formatting of the output. Representation of multiple whole-genome alignments is ambiguous and the same alignment can be formatted in different but mathematically equivalent forms varying by the number blocks.

We further investigate how the recall behaved as a function of nucleotide identity, for the two- and four-mice dataset (Fig. 4). As expected, recall decreases with nucleotide identity, though SibeliaZ’s recall remains above 90% for nucleotides from similar (80–100% identity) orthologous genes. Cactus has a slightly better recall in orthologous genes of lower identity on the two-mice dataset. We note that the gene annotation was constructed<sup>12</sup> using an alignment produced by Cactus which was further processed by annotation software CAT<sup>36</sup>. This fact might give Cactus a slight advantage in this comparison and explain why Cactus has a slightly better recall. Recall on orthologous gene pairs remains consistent in both two- and four-mice datasets for both datasets.

At the same time, we observed a much less consistent picture for paralogous pairs of genes. For example, SibeliaZ was able to recover nearly 90% of the paralogous basepairs belonging to gene pairs of nucleotide identity of 90% but found less than 45% of the base pairs of gene pairs of 80% identity.

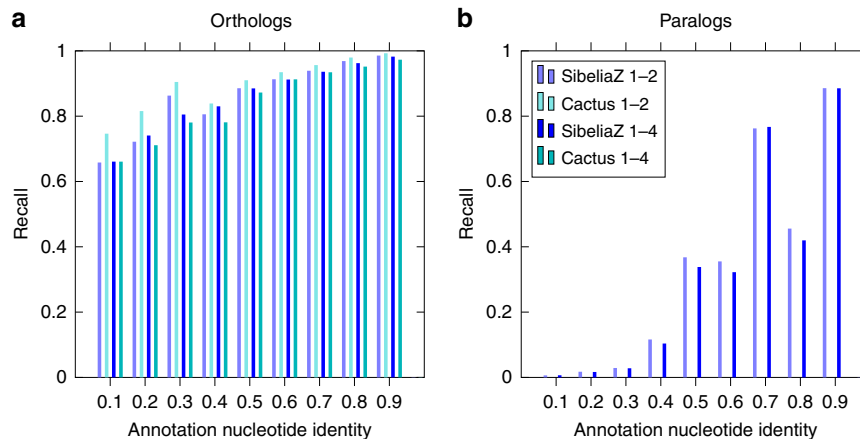
The results of the precision and recall measured with respect to LASTZ alignments are shown in Supplementary Table 3. On the dataset consisting of two genomes, Cactus had a slightly higher recall of 0.97 vs. 0.95 of SibeliaZ. On the other hand, SibeliaZ had better precision: 0.93 against 0.89 of Cactus. With the four genomes, SibeliaZ maintained its recall of 0.95 while the recall of Cactus dropped to 0.92. On this dataset SibeliaZ also had higher precision: 0.96 and 0.90, respectively. Overall, these numbers show that the alignment accuracy of SibeliaZ and Cactus is similar.

Finally, since we could not evaluate genome-wide precision, we use the proxy of the average number of nucleotide differences for the alignment columns (Supplementary Fig. 2). SibeliaZ’s alignment has a high degree of similarity: more than 95% of the alignment columns have  $\pi(c) \leq 0.1$ , which is what we would expect from aligning closely related genomes. Cactus has a slightly lower percentage of highly similar columns, which may simply indicate that it finds more blocks with higher divergence.

We note that the results in this Section evaluate the accuracy of SibeliaZ-LCB and spoa simultaneously; however, since SibeliaZ is targeted at closely related genomes, we expect the global alignment procedure to have a negligible effect on accuracy. This is due to the fact that the global alignment of similar sequences is likely to be unambiguous at homologous nucleotides and robust with respect to different algorithms and their parameters.

**Results on simulated data.** In addition to the real data, we measured the performance of different whole-genome aligners on a larger simulated dataset with small genomic divergence, called “primates” in ref. <sup>1</sup>. In this dataset, the distance from the root to the leaves in the phylogenetic tree is equal to 0.02 substitutions per site. The dataset has four genomes, with four chromosomes each, and each genome is approximately 185 Mbp in size. We did not use the other simulated dataset in ref. <sup>1</sup> since its divergence of around 0.4 substitutions per site is outside of the target range of SibeliaZ.

On this dataset, SibeliaZ pipeline was 20 times faster than Cactus and consumed 2.5 times less memory: SibeliaZ finished in 18 min using 7 GBs of memory, while Cactus took 363 min to finish and used 18 GBs of memory. Sibelia and MultiZ could not finish on the “primates” dataset in a week. Table 4 demonstrate the recall and precision values for the alignments produced by SibeliaZ and Cactus on this dataset. SibeliaZ showed a recall of 95% and a precision of 92%, while Cactus had 98% recall and 95%



**Fig. 4** Recall of nucleotide pairs from homologous genes on the 1-2 and 1-4 mouse datasets. Panels show recall of orthologous (a) and paralogous (b) nucleotide pairs binned according to the nucleotide identity of their respective genes in the annotation. Recall of paralogs by Cactus is not shown (see text).

**Table 4** Recall and precision of the alignments computed by Cactus and SibeliaZ on the “primates” dataset from the Alignment.

Program	Recall	Precision
SibeliaZ	0.95	0.92
Cactus	0.98	0.95

precision. We note that according to ref. <sup>1</sup>, the precision values calculated using this dataset can be considered lower bounds due to the features of the simulation process. Particularly, the ground truth for this dataset is likely to miss some repetitive alignments, hence we believe that the lower precision values of SibeliaZ may be due to this reason.

**Gene families.** We wanted to further understand SibeliaZ’s ability to recall homologous nucleotides from large gene families. Aligning genes having many copies is a challenging task since they generate a tangled de Bruijn graph. To investigate, we took each pair of genes in the two-mice dataset that have greater than 90% nucleotide identity. We then identify any other homologous genes that had a nucleotide identity of at least 90% to one of the genes in the pair. We refer to the number of such genes as the inferred family size of the gene pair, which roughly corresponds to the gene family size in the biological sense. Supplementary Fig. 3 then shows the recall of nucleotide pairs with respect to the inferred family size of their respective genes. The recall shows a lot of variance with respect to the inferred family size but does exhibit a general trend of decreasing with increasing family size. The largest bin (with an inferred family size of 58) corresponds to a single large gene family on the Y chromosome (*PTHR19368*) and actually has a relatively high recall.

This experiment shows that finding all copies of even very similar homologous sequences within long genomes can be a challenging task. Moreover, the high variance we observe indicates that this challenge cannot be reduced to a single factor like family size. A manual inspection of false negatives suggests that the drop in recall may be due to complex substructures of unannotated repeats forming tangled graph structures.

**Effect of parameters and sequence divergence.** SibeliaZ-LCB has four primary parameters that affect its performance. The most critical dependence is on the size of a  $k$ -mer (i.e.,  $k$ ) and the

maximum allowed length of a bubble  $b$ . For a given sequence divergence, the distance between shared  $k$ -mers forming bubbles in homologous regions increases with  $k$ . At the same time, the maximum allowed length of a bubble is  $b$ . If the distance exceeds  $b$ , then SibeliaZ may fail to uncover such regions and result in a lower recall. To avoid this situation, we can either decrease  $k$  or increase  $b$ . Decreasing  $k$  is desirable up to a point, but when  $k$  becomes too low, the de Bruijn graph becomes convoluted and our algorithm becomes more time and memory consuming. Increasing  $b$  can also be done but simultaneously increases the allowable gap length, leading to decreased precision.

Over-alignment is the problem of combining nonhomologous sequences in a single block, which is closely related to low precision<sup>37</sup>. In our case, one can control over-alignment by looking at the  $\pi(c)$  scores, as we have done in our analysis (Supplementary Fig. 2). A higher score indicates that more divergent sequences are included in a block. If the divergence is deemed too high by the user, it is recommended to reduce  $b$ .

To investigate this complex interplay between  $k$  and  $b$  and its relationship to sequence divergence, we used simulations (Supplementary Note 2) to measure recall (Supplementary Fig. 4) and precision (Supplementary Fig. 5) under various combinations. As predicted, recall increases with decreasing  $k$  and with increasing  $b$ , and precision decreases with increasing  $b$ . We note though that the precision varies only a little and remains high. Based on these analyses, we recommend two values of  $k$  for practical usage. For less complex organisms (e.g., bacteria), we recommend  $k = 15$ , since it yields the highest recall. This value is impractical for complex organisms (e.g., mammals) due to runtime, so we recommend setting  $k = 25$  in those cases as it provides a reasonable trade-off between accuracy and required computational resources (we used this for our mice datasets). For the value of  $b$ , we observed that increasing  $b$  lowers the precision at only higher values. Hence, we recommend  $b = 200$  as the default in all cases, as it led to high recall across all tested ranges of  $k$  on our simulated data without lowering precision.

To test the level of divergence which SibeliaZ-LCB can tolerate, we took the default values of  $k = 15$  or  $25$  and  $b = 200$  and plotted the precision vs. recall curve as a function of the root-to-leaf divergence of the dataset (Supplementary Fig. 6). We see that for  $k = 25$  the recall deteriorates significantly for datasets having a root-to-leaf evolutionary distance of more than 0.09 substitutions per site. Based on this, we recommend that for large datasets SibeliaZ-LCB be only used for detecting homologs with an evolutionary distance to the MRCA of at most 0.09 substitutions per site.

The other two parameters that can affect SibeliaZ-LCB's performance are the minimum size of a locally collinear block  $m$  and the abundance pruning parameter  $a$ . These parameters should be set according to the type of data and its intended use. The parameter  $m$  controls the fragmentation of the alignment and the coverage—higher  $m$  results in longer blocks spanning less of the genomes, since short blocks are not reported. We recommend the parameter  $m$  to be set to the length of the shortest homologous sequence of interest to the downstream analysis. We set  $m = 50$  as a default, since this is smaller than 93.1% of the known mice exons<sup>38</sup> and, more generally, we do not expect most applications to be interested in much blocks shorter than 50 nt. In the case that a user is interested in larger homologous units, they can increase  $m$  together with  $b$ . Alternatively, they can use either synteny block generation or alignment chaining algorithms for post-processing the alignments produced by SibeliaZ (see Supplementary Note 3 for relevant references).

The abundance pruning parameter  $a$  is a filtering parameter for  $k$ -mers whose abundance is above  $a$ . Such  $k$ -mers are still considered by SibeliaZ-LCB, but to a smaller extent, resulting in reduced recall in regions with such  $k$ -mers. We recommend setting  $a$  as high as the compute resources allow, keeping in mind that homologous blocks with multiplicity higher than  $a$  are possibly not going to be captured. For the mice dataset, we used  $a = 150$ .

## Discussion

In this paper, we presented a whole-genome alignment pipeline SibeliaZ based on an algorithm for identifying locally collinear blocks. The algorithm analyses the compacted de Bruijn graph and jointly reconstructs the path corresponding to a collinear block and identifies the induced collinear walks. We assume that the collinear walks share many vertices with this carrying path and form chains of bubbles. Each carrying path and the induced block is found greedily, using a scoring function that measures how close it is to all the sequences in the block. We then globally align the collinear blocks to generate the whole-genome alignment.

SibeliaZ builds on the ideas laid down in DRIMM-Synteny<sup>22</sup> and Sibelia<sup>23</sup> that used variants of the de Bruijn graphs for finding synteny blocks (we elaborate on the connection between the whole genome alignment and synteny reconstruction in Supplementary Note 3). Sibelia did not scale beyond bacterial genomes due to its slow graph construction algorithm and the fact that it continuously had to modify the graph. TwoPaCo<sup>27</sup> addressed the former issue and we use it as a standalone module in SibeliaZ. The latter issue was addressed in this paper by SibeliaZ-LCB, which achieves its speed in part because it keeps the underlying graph static.

The main strength of our approach is speed—we achieve drastic speedups compared to the state-of-the-art Progressive Cactus aligner<sup>34</sup> while retaining comparable accuracy. Using a single machine, on 16 mice genomes, SibeliaZ runs in under 16 h, while Progressive Cactus is not able to complete for even 8 mice genomes, within seven days. We note that it is possible for Cactus to construct larger alignments by utilizing a distributed computer cluster<sup>34</sup>. In our study, we concentrated on improving the scalability of the whole-genome alignment when only a single machine is available. In the future, we hope to develop a version of SibeliaZ that will work in the distributed setting as well. Overall, SibeliaZ is the only tool available that can scale to many long, closely related genomes on a single machine.

The biggest limitation of our approach is the limited tolerance to the divergence of input sequences. As suggested by the results

on simulated bacterial data, SibeliaZ works best for aligning genomes having an evolutionary distance to the MRCA of at most 0.09 substitutions per site. Aligning more divergent genomes with SibeliaZ is still possible but it will result in the smaller recall; for such datasets, Cactus remains a better option. In the future, we hope to address this limitation by employing techniques like postprocessing of the output with more sensitive homology finders.

If the alignments themselves are not needed, SibeliaZ-LCB can be run alone (without *spoa*) to construct the collinear blocks. This is most useful in applications stemming from studies of genome rearrangements, which can be applied to study breakpoint reuse<sup>39</sup>, ancestral genome reconstruction<sup>40</sup> and phylogenies<sup>41</sup>. Locally collinear blocks are also a required input for scaffolding tools using multiple references genomes<sup>42–46</sup>. For such applications, the output of SibeliaZ-LCB can be used either directly or after postprocessing by a synteny block generator<sup>22,47</sup>.

There are several remaining open questions of interest. A formal analysis of SibeliaZ-LCB's runtime is relevant, but doing it in a useful way is a challenge. The worst-case running time does not reflect the actual one; moreover, we observed that the actual one depends on multi-thread synchronization, which is challenging to model. However, it would be interesting if such a time analysis can be performed parametrized by the crucial properties of the structure of the input. We also did not investigate how close to an optimal solution our greedy heuristic gets. One way to do this would be to find an optimal carrying path using exhaustive enumeration, but the search space even for a small realistic example is too big. We suspect that a polynomial time-optimal solution is not possible, but the computational complexity of our problem is open.

SibeliaZ is the first multiple whole-genome aligners that can run on a single machine in a reasonable time on a dataset such as the 16 mouse genomes analyzed in this paper. With ongoing initiatives like the Vertebrate Genomes Project and the insect5k, thousands of species will soon have a reference genome available, and the sequencing and assembly of various subspecies and strains will be the next logical step for many comparative genomics studies. For example<sup>48</sup>, currently holds 18 assembled maize genomes, with more to come in the recent future. Similarly, the Solanaceae Genomics Network has recently released the genomes of 13 diverse tomato accessions (<https://solgenomics.net/projects/tomato13/>). Analysis of such datasets is likely to be carried out in single-lab settings with limited compute resources, rather than at large computing centers like EMBL or NCBI. SibeliaZ makes a significant leap toward enabling such studies.

## Methods

**Preliminaries.** First, we will define the de Bruijn graph and related objects. Given a positive integer  $k$  and a string  $s$ , we define a multigraph  $G(s, k)$  as the *de Bruijn graph* of  $s$ . The vertex set consists of all substrings of  $s$  of length  $k$ , called  $k$ -mers. For each substring  $x$  of length  $k + 1$  in  $s$ , we add a directed edge from  $u$  to  $v$ , where  $u$  is the prefix of  $x$  of length  $k$  and  $v$  the suffix of  $x$  of length  $k$ .

Each occurrence of a  $(k + 1)$ -mer yields a unique edge, and every edge corresponds to a unique location in the input. Two edges are *parallel* if they are oriented in the same direction and have the same endpoints. Note that two edges are parallel if and only if they were generated by the same  $(k + 1)$ -mer. This way, we use the notion of parallel edges to refer to a set of identical  $(k + 1)$ -mers in the input strings. Parallel edges are not considered identical. The de Bruijn graph can also be constructed from a set of sequences  $S = \{s_1, \dots, s_n\}$ . This graph is the union of the graphs constructed from the individual strings:  $G(S, k) = \bigcup_{1 \leq i \leq n} G(s_i, k)$ . See Fig. 1 for an example.

The set of all edges in a graph  $G$  is denoted by  $E(G)$ . We write  $(u, v)$  to denote an edge from vertex  $u$  to  $v$ . A *walk*  $p$  is a sequence of edges  $((v_1, v_2), (v_2, v_3), \dots, (v_{|p|-1}, v_{|p|}))$  where each edge  $(v_i, v_{i+1})$  belongs to  $E(G)$ . The length of the walk  $p$ , denoted by  $|p|$ , is the number of edges it contains. The last edge of a walk  $p$  is denoted by the  $\text{end}(p)$ . A *path* is a walk that visits each vertex at most once.

In a de Bruijn graph, a given edge  $x$  was generated by a  $(k + 1)$ -mer starting at some position  $j$  of some string  $s_i$ . To retrieve the position  $j$  of the  $(k + 1)$ -mer that

generated edge  $x$ , we define function  $\text{pos}(x) = j$ . We use the function  $\text{pos}$  to map edges of the graph back to positions of the  $k$ -mers that generated them. For an edge  $x$ , its successor, denoted by  $\text{next}(x)$ , is an edge  $y$  such that both  $x$  and  $y$  are generated by the same string and  $\text{pos}(y) = \text{pos}(x) + 1$ . Note that a successor does not always exist.

A walk  $p = (x_1, \dots, x_{|p|})$  is *genomic* if  $\text{next}(x_i) = x_{i+1}$  for  $1 \leq i \leq |p| - 1$ . In other words, a walk is genomic if it was generated by a substring in the input. The *b*-extension of a genomic walk  $p$  is the longest genomic walk  $q = (y_1, \dots, y_{|q|})$  such that  $y_1 = \text{next}(\text{end}(p))$  and  $|q| \leq b$ . The *b*-extension of a walk  $p$  is uniquely defined and usually has length  $b$ , unless  $p$  was generated by a substring close to an end of an input string. Intuitively, *b*-extension defines edges lying ahead of a walk. As our algorithm works in the seed-and-extend manner, we later use *b*-extensions to find the most appropriate extension of the current block. The concatenation of two walks  $x$  and  $y$  is a walk (if it exists)  $xy$  consisting of edges of  $x$  followed by edges of  $y$ . We use the concatenation operation to extend the genomic walks constituting locally collinear blocks using appropriate *b*-extensions.

**Problem formulation.** In this section, we will define the collinear block reconstruction problem in de Bruijn graphs. A *collinear block* is a set of edge-disjoint genomic walks with length at least  $m$ , where  $m$  is a parameter. We call walks constituting a collinear block *collinear walks*. In order to quantify how well collinear walks correspond to homologous sequences, we will define a *collinearity score* of a collinear block. Our problem will then be to find a set of collinear blocks that are pairwise edge-disjoint and have the largest score.

We capture the pattern of two homologous collinear walks through the concept of chains and bubbles. A *bubble* is a subgraph corresponding to a possible mutation flanked by equal sequences. Formally, a pair of walks  $x$  and  $y$  form a bubble  $(x, y)$  if all of the following holds: (1)  $x$  and  $y$  have common starting and ending vertices; (2)  $x$  and  $y$  have no common vertices except the starting and ending ones; and (3)  $|x| \leq b$  and  $|y| \leq b$ , where  $b$  is a parameter. A *chain*  $c = ((x_1, y_1), (x_2, y_2), \dots, (x_m, y_m))$  is a sequence of bubbles such that  $x = x_1x_2\dots x_m$  and  $y = y_1y_2\dots y_m$  are walks in a de Bruijn graph. In other words, a chain is a series of bubbles where each bubble is a proper continuation of the previous one. Note that two parallel edges form a bubble and a chain arising from equal sequences corresponds to a series of such bubbles. This way, a chain models a pair of sequences that potentially have point mutations and indels. For an example of a bubble and a chain, see Fig. 1.

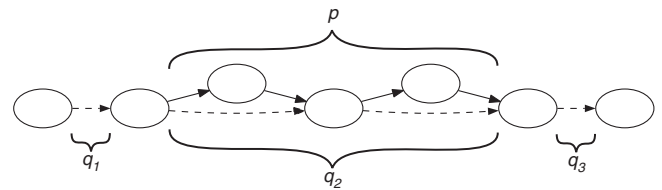
The subgraph resulting from more than two collinear walks can be complex, and there are several ways of capturing it. We note that there are previous studies generalizing the idea of bubbles, see<sup>49–53</sup>, mostly in the context of analyzing assembly graphs. We decided to follow a different approach designed specifically for capturing locally collinear blocks.

Our approach is to give a definition that naturally leads itself to an algorithm. As homologous sequences all originate from some common ancestral sequence  $s_a$ , they should have many common  $k$ -mers and there should be a path  $p_a = G(s_a, k)$  through the graph forming a long chain with each walk  $p$  in the collinear block. We call such a path a *carrying* one. We require the chains to be longer than  $m$  to avoid confusing spuriously similar sequences with true homologs. At the same time, a collinear walk may only partially form a chain with the carrying path, leaving hanging ends at the ends of the carrying path, which is undesirable since it implies that these graphs originate from dissimilar sequences. We note that compared to the previous work on chaining anchors<sup>18–20</sup> our definition of the block will be more relaxed. Namely, we will not require common  $k$ -mers to be present in all copies of a block. In addition, many alignment methods use phylogenetic information for scoring purposes. We decided to not use it since our method targets closely related sequences, such as strains of the same species, where the phylogeny is often unknown.

We formalize this intuition by introducing a scoring function quantifying how well a carrying path describes a collection of the collinear walks. The function rewards long chains formed by the carrying path and a collinear walk and penalizes the hanging ends. Given a carrying path  $p_a$  and a genomic walk  $p$ , let  $q_2$  be the longest subpath of  $p_a$  that forms a chain with  $p$ . Then, we can write  $p_a = q_1q_2q_3$ . Recall that  $m$  is the parameter denoting the minimum length of a collinear block, and  $b$  is the maximum bubble size. We define the score  $f(p_a, p)$  as

$$f(p_a, p) = \begin{cases} 0, & \text{if } |p| < m \\ |p| - (|q_1| + |q_3|)^2, & \text{if } |p| \geq m \text{ and } |q_1|, |q_3| \leq b \\ -\infty, & \text{if } |p| \geq m \text{ and } (|q_1| > b \text{ or } |q_3| > b) \end{cases} \quad (1)$$

The third case forbids walks (i.e., gives them a score of  $-\infty$ ) where the hanging ends are too long, and the first case ignores walks (i.e., gives them a score of 0) that weave through  $p_a$  but are too short. The second case gives a score that is proportional to the length of the part of  $p_a$  that forms a chain with  $p$ . At the same time, it reduces the score if the collinear walks leave hanging ends  $q_1$  and  $q_3$ —the parts of  $p_a$  not participating in the chain. The penalty induced by these ends is squared to remove spuriously similar sequences from from the collinear block. This form of scoring function was chosen because it performed well during the initial stages of development. We do not penalize for the discrepancy between  $p$  and  $q_2$  for the sake of simplicity of the scoring function and avoiding extra computation needed to calculate it. Figure 5 shows an example of computing the score.



**Fig. 5** An example of computing the score of a walk  $p$  (solid) relative to a carrying path  $p_a = q_1q_2q_3$  (dashed). The path  $p$  forms a chain with the subpath  $q_2$  of  $p_a$ , while subpaths  $q_1$  and  $q_3$  form hanging ends. We count the length of  $p$  and subtract lengths of the hanging ends. Thus, the score  $f(p_a, p) = 4 - (1 + 1)^2 = 0$ .

The *collinearity score* of a collinear block is given by

$$f(P) = \max_{p_a \in \mathcal{P}} \sum_{p \in P} f(p_a, p), \quad (2)$$

where  $p_a$  can be any path (not necessarily genomic). In other words, we are looking for a path forming longest chains with the collinear walks and thus maximizes the score. The collinear blocks reconstruction problem is to find a set of collinear blocks  $\mathcal{P}$  such that  $\sum_{p \in \mathcal{P}} f(P)$  is maximum and no two walks in  $\mathcal{P}$  share an edge. Note that the number of collinear blocks is not known in advance. For an example of a complex collinear block in the de Bruijn graph and a carrying path capturing it, refer to Fig. 1b.

**The collinear blocks reconstruction algorithm.** Our algorithm’s main pseudocode is shown in Box 1 and its helper function in Box 2. First, we describe the high-level strategy. The main algorithm is greedy and works in the seed-and-extend fashion. It starts with an arbitrary edge in the graph and tries to extend it into a carrying path that induces a collinear block with the highest possible collinearity score  $P_{\text{best}}$ . If the block has a positive score, then  $P_{\text{best}}$  is added to our collection of collinear blocks  $\mathcal{P}$ . The algorithm then repeats, attempting to build a collinear block from a different edge seed. New collinear blocks cannot use edges belonging to previously discovered collinear blocks. This process continues until all possible edges are considered as seeds. The algorithm is greedy in the sense that once a block is found and added to  $\mathcal{P}$ , it cannot be later changed to form a more optimal global solution.

To extend a seed into a collinear block  $P$ , we first initialize the collinear block with a walk for each unused edge parallel to the seed (including the seed) (lines 7 and 8). These parallel edges represent the different occurrences of the seed string in the input and, hence, form the initial collinear block. We then proceed in phases, where each phase is an iteration of the while loop (lines 9–19). During each phase, the carrying path  $p_a$  is extended using a walk  $r$  of length at most  $b$  (lines 10–14). Next, we try to extend each of the collinear walks in a way that forms chains with the extended  $p_a$  (lines 15–19). The extension of a seed into a collinear block is also a greedy process since we only change  $p_a$  and the walks in  $P$  by extending them and never by changing any edges. Finally, we check that the collinearity score for our extended block is still positive—if it is, we iterate to extend it further, otherwise, we abandon our attempts at further extending the block. We then recall the highest-scoring block that was achieved for this seed and save it into our final result  $\mathcal{P}$  (lines 20–22).

To pick the walk  $r$  by which to extend  $p_a$ , we use a greedy heuristic (lines 10–14). First, we pick the vertex  $t$  which we want to extend to reach (lines 10–12). We limit our search to those vertices that can be reached by a genomic walk from the end of  $p_a$  and greedily chose the one that is most often visited by the *b*-extensions of the collinear walks in  $P$ . Intuitively, we hope to maximize the number of collinear walks that will form longer chains with  $p_a$  after its extension and thereby boost the collinearity score. We then extend  $p_a$  using the shortest *b*-extension of the walks in  $P$  to reach  $t$ . We chose this particular heuristic because it showed superior performance compared to other possible strategies.

Once we have selected the genomic walk  $r$  by which to extend  $p_a$ , we must select the extensions to our collinear walks  $P$  that will form chains with  $p_a$ . This is done by the function Update-collinear-blocks (Box 2). We extend the walks to match  $r$  by considering the vertices of  $r$  consecutively, one at a time. To extend to a vertex  $w$ , we consider all the different locations of  $w$  in the input (each such location is represented by an edge  $x$  ending at  $w$ ). For each location, we check if it can be reached by a *b*-extension from an existing  $p \in P$ . If yes, then we extend  $p$ , so as to lengthen the chain that it forms with  $p_a$ . If there are multiple collinear walks that reach  $w$ , we take the nearest one. If no, then we start a new collinear walk using just  $x$ . Figure 6 shows an example of updating a collinear walk and Fig. 2 shows a full run of the algorithm for a single seed.

Our description here only considers extending the initial seed to the right, i.e., using out-going edges in the graph. However, we also run the procedure to extend the initial seed to the left, using the incoming edges. The case is symmetric and we, therefore, omit the details.

**Other considerations.** For simplicity of presentation, we have described the algorithm in terms of the ordinary de Bruijn graph; however, it is crucial for



**Box 1 | Algorithm Find-collinear-blocks**

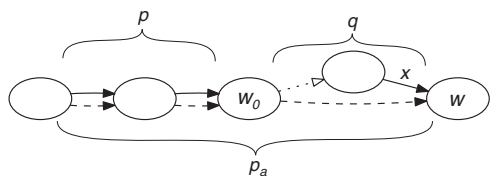
```

Input: strings  $S$ , integers  $k$ ,  $b$  and  $m$ 
Output: a set of edge-disjoint subgraphs of  $G(S, k)$  representing collinear blocks
1:  $\mathcal{P} \leftarrow \emptyset$   $\triangleright$  Collinear blocks
2:  $G \leftarrow G(S, k)$   $\triangleright$  Construct the multigraph
3: for all distinct pairs  $(u, v) \in E(G)$  do  $\triangleright$  Check possible seeds
4:   Initialize the current-carrying path  $p_a$  with  $(u, v)$ 
5:    $P \leftarrow \emptyset$   $\triangleright$  Sorted set of collinear walks forming chains with  $p_a$ 
6:    $P_{\text{best}} \leftarrow \emptyset$   $\triangleright$  Highest-scoring collinear block induced by  $p_a$ 
7:   for edges  $x \in E(G)$  parallel to  $(u, v)$  not marked as used do
8:     Add to  $P$  a new collinear walk consisting of  $x$ 
9:     while  $f(P) \geq 0$  do  $\triangleright$  Extend the carrying path as far as possible
10:     $Q \leftarrow \{q \mid q \text{ is the } b\text{-extension of a } p \in P\}$ 
11:     $w_0 \leftarrow$  last vertex in  $p_a$ 
12:     $t \leftarrow$  a vertex, reachable from  $w_0$  via a genomic walk, that is visited by the most walks of  $Q$ .
13:    Let  $r \in Q$  be the shortest walk from  $w_0$  to  $t$ 
14:    Denote the vertices of  $r$  as  $w_0, w_1, \dots, w_{|r|}$ , with  $w_{|r|} = t$ 
15:    for  $i \leftarrow 1$  to  $|r|$  do
16:      Append  $(w_{i-1}, w_i)$  to the carrying path  $p_a$ 
17:       $P \leftarrow$  Update-collinear-walks( $P, w_i$ )
18:      if  $f(P) > f(P_{\text{best}})$  then
19:         $P_{\text{best}} \leftarrow P$ 
20:    if  $f(P_{\text{best}}) > 0$  then
21:       $\mathcal{P} \leftarrow \mathcal{P} \cup \{P_{\text{best}}\}$ 
22:      Mark edges visited by walks of  $P_{\text{best}}$  as used
23: return  $\mathcal{P}$ 
    
```

**Box 2 | Update-collinear-walks**

```

Input: A sorted set of collinear walks  $P$ , a vertex  $w$ 
Output: Updated set  $P$ 
1: for edges  $x \in E(G)$  ending at  $w$  not marked as used do
2:   Let  $p \in P$  be a walk such that its  $b$ -extension  $q$  contains  $x$  and  $\text{pos}(\text{end}(p))$  is maximized  $\triangleright$  Find a walk extendable with  $x$ 
3:   if such  $p$  exists then
4:     Truncate  $q$  so that  $\text{end}(q) = x$ 
5:     Append  $p$  with  $q$   $\triangleright$  Lengthen the chain that  $p$  forms with  $p_a$ 
6:   else
7:     Add a new walk consisting of the edge  $x$  to  $P$ 
8: return  $P$ 
    
```



**Fig. 6 Illustration for Algorithm Update-collinear-walks (Box 2).** A collinear walk  $p$  (solid) requires an update after the carrying path  $p_a$  is extended with the dashed edge  $(w_0, w)$ . The path  $p_a$  now ends at the vertex  $w$ , which has another incoming edge  $x$ . Since  $x$  is a part of the  $b$ -extension of  $p$  (denoted by  $q$ ),  $p$  can be appended with  $q$  to form a longer chain and boost the collinearity score.

running time and memory usage that the graph is compacted first. Informally, the compacted de Bruijn graph replaces each non-branching path with a single edge. Formally, the vertex set of the compacted graph consists of vertices of the regular de Bruijn graph that have at least two outgoing (or ingoing) edges pointing at (incoming from) different vertices. Such vertices are called junctions. Let  $\ell = v_1, \dots, v_n$  be the list of  $k$ -mers corresponding to junctions, in the order they appear in the underlying string  $s$ . The edge set of the compacted graph consists of edges  $\{v_1 \rightarrow v_2, v_2 \rightarrow v_3, \dots, v_{n-1} \rightarrow v_n\}$ . We efficiently construct the compacted graph using our previously published algorithm TwoPaCo<sup>27</sup>.

This transformation maintains all the information while greatly reducing the number of edges and vertices in the graph. This makes the data structures smaller and allows the algorithm to fast-forward through non-branching paths, instead of considering each  $(k + 1)$ -mer one by one. Our previous description of the algorithm remains valid, except that the data structures operate with vertices and edges from the compacted graph instead of the ordinary one. The only necessary change is that when we look for an edge  $y$  parallel to  $x$ , we must also check that  $y$  and  $x$  spell the same sequence. This is always true in an ordinary graph but not necessarily in a compacted graph.

An important challenge of mammalian genomes is that they contain high-frequency  $(k + 1)$ -mers, which can clog up our data structures. To handle this, we modify the algorithm by skipping over any junctions that correspond to  $k$ -mers occurring more than  $a$  times; we call  $a$  the abundance pruning parameter. Specifically, prior to constructing the edge set of the compacted de Bruijn graph, we remove all high abundance junctions from the vertex set. The edge set is constructed as before, but using this restricted list of junctions as the starting point. This strategy offers a way to handle high-frequency repeats at the expense of limiting our ability to detect homologous blocks that occur more than  $a$  times.

The organization of our data in memory is instrumental to achieving high performance. To represent the graph, we use a standard adjacency list representation, annotated with position information, and other relevant data. We also maintain a list of the junctions in the paragraph above in the order they appear in the input sequences, thereby supporting next() queries. The walks in the collinear block  $P$  are stored as a dynamic sorted set, implemented as a binary search tree. The search key is the genome/position for the end of each walk. This allows performing a binary search in line 2 of Algorithm Update-collinear-walks.

Another aspect that we have ignored up until now is that DNA is double-stranded and collinear walks can be reverse-complements of each other. If  $s$  is a

string, then let  $\bar{s}$  be its reverse complement. We handle double strandedness in a natural way by using the comprehensive de Bruijn graph, which is defined as  $G_{\text{comp}}(s, k) = G(s, k) \cup G(\bar{s}, k)$ <sup>27</sup>. Our algorithm and corresponding data structures can be modified to work with the comprehensive graph with a few minor changes which we omit here.

Our implementation is parallelized by exploring multiple seeds simultaneously, i.e., parallelizing the for a loop at line 3 of Algorithm Find-collinear-blocks. This loop is not embarrassingly parallelizable, since two threads can start exploring two seeds belonging to the same carrying path. In such a case, there will be a collision on the data structure used to store used marks. To address this issue, we process the seeds in batches of fixed size. All the seeds within a batch are explored in parallel and the results are saved without modifying the “used” marks. Once the batch is processed, a single arbiter thread checks if there is any overlap in the used marks of the different threads. If there is, it identifies the sources of the conflict and reruns the algorithm at the conflicting seeds serially. Since most seeds do not yield valid carrying paths, such conflicts are rare. Once there is no conflict, the arbiter updates the used main data structures with the results of the batch. This design allows the computation result to be deterministic and independent of the number of threads used.

**Reporting summary.** Further information on research design is available in the Nature Research Reporting Summary linked to this article.

### Data availability

Table 1 contains the list of GenBank accession numbers of the mice genomes we used in our experiments (Figs. 3 and 4, Supplementary Figs. 1–3). The nine simulated datasets we generated (Supplementary Figs. 4–6), ground-truth alignments for the mouse data (Fig. 4, Supplementary Figs. 1–3), and alignments produced by SibeliaZ and Progressive Cactus (Fig. 4, Supplementary Figs. 2 and 3) are available for download at <https://github.com/medvedevgroup/SibeliaZ/blob/master/DATA.txt>.

### Code availability

Our tool is open source and freely available at <https://github.com/medvedevgroup/SibeliaZ>.

Received: 28 September 2019; Accepted: 29 October 2020;

Published online: 10 December 2020

### References

- Earl, D. et al. Alignathon: a competitive assessment of whole-genome alignment methods. *Genome Res.* **24**, 2077–2089 (2014).
- Dewey, C. N. & Pachter, L. Evolution at the nucleotide level: the problem of multiple whole-genome alignment. *Hum. Mol. Genet.* **15**, R51–R56 (2006).
- Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. Basic local alignment search tool. *J. Mol. Biol.* **215**, 403–410 (1990).
- Altschul, S. F. et al. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res.* **25**, 3389–3402 (1997).
- Schwartz, S. et al. Human–mouse alignments with blastz. *Genome Res.* **13**, 103–107 (2003).
- Harris, R. S. *Improved Pairwise Alignment of Genomic DNA*. (The Pennsylvania State University, 2007).
- Kent, W. J. Blat—the blast-like alignment tool. *Genome Res.* **12**, 656–664 (2002).
- Blanchette, M. et al. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Res.* **14**, 708–715 (2004).
- Dubchak, I., Poliakov, A., Kislyuk, A. & Brudno, M. Multiple whole-genome alignments without a reference organism. *Genome Res.* **19**, 682–689 (2009).
- Angiuoli, S. V. & Salzberg, S. L. Mugsy: fast multiple alignment of closely related whole genomes. *Bioinformatics* **27**, 334–342 (2011).
- Paten, B. et al. Cactus: algorithms for genome multiple sequence alignment. *Genome Res.* **21**, 1512–1528 (2011).
- Lilue, J. et al. Sixteen diverse laboratory mouse reference genomes define strain-specific haplotypes and novel functional loci. *Nat. Genet.* **50**, 1574 (2018).
- Darling, A. C., Mau, B., Blattner, F. R. & Perna, N. T. Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome Res.* **14**, 1394–1403 (2004).
- Dewey, C. N. *Aligning Multiple Whole Genomes with Mercator and MAVID*. 221–235 (Humana Press, Totowa, NJ, 2008).
- Paten, B., Herrero, J., Beal, K., Fitzgerald, S. & Birney, E. Enredo and pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome Res.* **18**, 1814–1828 (2008).
- Darling, A. E., Mau, B. & Perna, N. T. Progressivmauve: multiple genome alignment with gene gain, loss and rearrangement. *PLoS ONE* **5**, e11147 (2010).
- Minkin, I., Pham, H., Starostina, E., Vyahhi, N. & Pham, S. C-sibelia: an easy-to-use and highly accurate tool for bacterial genome comparison. *F1000Research* <https://f1000research.com/articles/2-258> (2013).
- Myers, G. & Miller, W. Chaining multiple-alignment fragments in sub-quadratic time. in *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '95*, 38–47 (Society for Industrial and Applied Mathematics, USA, 1995).
- Abouelhoda, M. I. & Ohlebusch, E. Chaining algorithms for multiple genome comparison. *J. Discret. Algorithms* **3**, 321–341 (2005).
- Ohlebusch, E. & Abouelhoda, M. I. *Chaining Algorithms and Applications in Comparative Genomics*. (Handbook of Computational Molecular Biology, 2006).
- Raphael, B., Zhi, D., Tang, H. & Pevzner, P. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Res.* **14**, 2336–2346 (2004).
- Pham, S. & Pevzner, P. Drimm-synteny: decomposing genomes into evolutionary conserved segments. *Bioinformatics* **26**, 2509–2516 (2010).
- Minkin, I., Patel, A., Kolmogorov, M., Vyahhi, N. & Pham, S. Sibelia: A scalable and comprehensive synteny block generation tool for closely related microbial genomes. in (eds Darling, A. & Stoye, J.) *Algorithms in Bioinformatics*. 215–229 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2013).
- Marcus, S., Lee, H. & Schatz, M. C. Splitmem: a graphical algorithm for pan-genome analysis with suffix skips. *Bioinformatics* **30**, 3476–3483 (2014).
- Chikhi, R., Limasset, A. & Medvedev, P. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics* **32**, i201–i208 (2016).
- Baier, U., Beller, T. & Ohlebusch, E. Graphical pan-genome analysis with compressed suffix trees and the burrows-wheeler transform. *Bioinformatics* **32**, 497–504 (2016).
- Minkin, I., Pham, S. & Medvedev, P. Twopaco: an efficient algorithm to build the compacted de bruijn graph from many complete genomes. *Bioinformatics* **33**, 4024–4032 (2017).
- Cleary, A., Kahanda, I., Mumey, B., Mudge, J. & Ramaraj, T. Exploring frequented regions in pan-genomic graphs. in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, 89–97 (Association for Computing Machinery, New York, NY, USA, 2017).
- Vaser, R., Sović, I., Nagarajan, N. & Šikić, M. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Res.* **27**, 737–746 (2017).
- Sayers, E. W. et al. GenBank. *Nucleic Acids Res.* **48**, D84–D86 (2019).
- Brudno, M. et al. Lagan and multi-lagan: efficient tools for large-scale multiple alignment of genomic dna. *Genome Res.* **13**, 721–731 (2003).
- Perry, E. Personal communication (2018).
- Tajima, F. Statistical method for testing the neutral mutation hypothesis by dna polymorphism. *Genetics* **123**, 585–595 (1989).
- Armstrong, J. et al. Progressive alignment with cactus: a multiple-genome aligner for the thousand-genome era. Preprint at <https://www.biorxiv.org/content/early/2019/10/15/730531> (2019).
- Paten, B. et al. Cactus graphs for genome comparisons. *J. Comput. Biol.* **18**, 469–481 (2011).
- Fiddes, I. T. et al. Comparative annotation toolkit (cat)-simultaneous clade and personal genome annotation. *Genome Res.* **28**, 1029–1038 (2018).
- Schwartz, A. S. & Pachter, L. Multiple alignment by sequence annealing. *Bioinformatics* **23**, e24–e29 (2007).
- Sakharkar, M. K., Perumal, B. S., Sakharkar, K. R. & Kanguane, P. An analysis on gene architecture in human and mouse genomes. *Silico Biol.* **5**, 347–365 (2005).
- Pevzner, P. & Tesler, G. Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. *Proc. Natl Acad. Sci. USA* **100**, 7672–7677 (2003).
- Kim, J. et al. Reconstruction and evolutionary history of eutherian chromosomes. *Proc. Natl Acad. Sci. USA* **114**, E5379–E5388 (2017).
- Luo, H. et al. Phylogenetic analysis of genome rearrangements among five mammalian orders. *Mol. Phylogenet. Evolut.* **65**, 871–882 (2012).
- Kolmogorov, M. et al. Chromosome assembly of large and complex genomes using multiple references. *Genome Res.* **28**, 1720–1732 (2018).
- Kim, J. et al. Reference-assisted chromosome assembly. *Proc. Natl Acad. Sci. USA* **110**, 1785–1790 (2013).
- Kolmogorov, M., Raney, B., Paten, B. & Pham, S. Ragout—a reference-assisted assembly tool for bacterial genomes. *Bioinformatics* **30**, i302–i309 (2014).
- Chen, K.-T. et al. Multi-car: a tool of contig scaffolding using multiple references. *BMC Bioinform.* **17**, 469 (2016).
- Aganezov, S. & Alekseyev, M. A. Multi-genome scaffold co-assembly based on the analysis of gene orders and genomic repeats. in (eds Bourgeois, A., Skums, P., Wan, X. & Zelikovsky, A.) *Bioinformatics Research and Applications*. 237–249 (Springer International Publishing, Cham, 2016).
- Proost, S. et al. i-adhore 3.0—fast and sensitive detection of genomic homology in extremely large data sets. *Nucleic Acids Res.* **40**, e11–e11 (2011).
- Portwood, J. L. et al. Maizegdb 2018: the maize multi-genome genetics and genomics database. *Nucleic Acids Res.* **47**, D1146–D1154 (2018).

49. Onodera, T., Sadakane, K. & Shibuya, T. Detecting superbubbles in assembly graphs. in (eds Darling, A. & Stoye, J.) *Algorithms in Bioinformatics*. 338–348 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2013).
50. Sung, W., Sadakane, K., Shibuya, T., Belorkar, A. & Pyrogova, I. An  $\mathcal{O}(m \log m)$ -time algorithm for detecting superbubbles. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **12**, 770–777 (2015).
51. Iliopoulos, C. S., Kundu, R., Mohamed, M. & Vayani, F. Popping superbubbles and discovering clumps: Recent developments in biological sequence analysis. in (eds Kaykobad, M. & Petreschi, R.) *WALCOM: Algorithms and Computation*. 3–14 (Springer International Publishing, Cham, 2016).
52. Brankovic, L. et al. Linear-time superbubble identification algorithm for genome assembly. *Theor. Comput. Sci.* **609**, 374–383 (2016).
53. Paten, B., Novak, A. M., Garrison, E. & Hickey, G. Superbubbles, ultrabubbles and cacti. in (ed Sahinalp, S. C.) *Research in Computational Molecular Biology*. 173–189 (Springer International Publishing, Cham, 2017).

## Acknowledgements

We would like to thank Mikhail Kolmogorov for useful suggestions on the empirical evaluation of our algorithm; Robert Harris for his help with running MultiZ; Son Pham for introducing us to the problem; and the Ensembl support team for helping us with retrieving the gene annotations.

This work has been supported in part by NSF awards DBI-1356529, CCF-1439057, IIS-1453527, and IIS-1421908 to P.M. The research reported in this publication was supported by the National Institute Of General Medical Sciences of the National Institutes of Health under Award No. R01GM130691. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

## Author contributions

Conceptualization: I.M., Methodology: I.M. and P.M., Software: I.M., Validation: I.M. and P.M., Writing—original draft: I.M. and P.M., Writing—review and editing: I.M. and P.M., and Funding acquisition: P.M.

## Competing interests

The authors declare no competing interests.

## Additional information

**Supplementary information** is available for this paper at <https://doi.org/10.1038/s41467-020-19777-8>.

**Correspondence** and requests for materials should be addressed to I.M.

**Peer review information** *Nature Communications* thanks the anonymous reviewers for their contribution to the peer review of this work.

**Reprints and permission information** is available at <http://www.nature.com/reprints>

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2020