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

Ilia Minkin*[1] and Paul Medvedev†[1, 2, 3]

[1]Department of Computer Science and Engineering, The Pennsylvania State University
[2]Department of Biochemistry and Molecular Biology, The Pennsylvania State University
[3]Center for Computational Biology and Bioinformatics, The Pennsylvania State University

Abstract

Multiple whole-genome alignment is a fundamental and challenging problems in bioinformatics. Despite many ongoing successes, today's methods are not able to keep up with the growing number, length, and complexity of assembled genomes. Approaches based on using compacted de Bruijn graphs to identify and extend anchors into locally collinear blocks hold the potential for scalability, but current algorithms still do not scale to mammalian genomes. We present a novel algorithm SibeliaZ-LCB for identifying collinear blocks in closely related genomes based on the analysis of the de Bruijn graph. We further incorporate it into a multiple whole-genome alignment pipeline called SibeliaZ. SibeliaZ shows drastic run-time improvements over other methods on both simulated and real data, with only a limited decrease in accuracy. On sixteen recently assembled strains of mice, SibeliaZ runs in under 12 hours, while other tools could not run to completion for even eight mice, given a week. SibeliaZ makes a significant step towards improving scalability of multiple whole-genome alignment and collinear block reconstruction algorithms and will enable many comparative genomics studies in the near future.

1 Introduction

Multiple whole-genome alignment is 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 (Earl *et al.*, 2014). 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 — we elaborate on the connection in Supplementary Note 1.

There are two common strategies to tackle the whole-genome alignment problem (Dewey and Pachter, 2006). The first one is based on finding pairwise local alignments (Altschul *et al.*, 1990, 1997; Schwartz *et al.*, 2003; Harris, 2007; Kent, 2002) and then extending them into multiple local alignments (Blanchette *et al.*, 2004; Dubchak *et al.*, 2009a; Angiuoli and Salzberg, 2011;

*ium125@psu.edu
†pashadag@cse.psu.edu

Paten *et al.*, 2011). While this strategy is known for its high accuracy, a competitive assesment of multiple whole-genome alignment methods (Alignathon, Earl *et al.* (2014)) 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 (Earl *et al.*, 2014). Therefore, these approaches are ill-suited for large numbers of long and complex mammalian genomes, such as the recently assembled 16 strains of mice (Lilue *et al.*, 2018).

Alternatively, anchor based strategies can be applied to decompose genomes into locally collinear blocks (Darling *et al.*, 2004). These are blocks that are free from non-linear rearrangements, such as inversions or transpositions. Once such blocks are identified, they can independently be globally aligned (Darling *et al.*, 2004; Dewey, 2007; Paten *et al.*, 2008; Darling *et al.*, 2010; Minkin *et al.*, 2013a). 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 (Raphael *et al.*, 2004; Pham and Pevzner, 2010; Minkin *et al.*, 2013b) (we note that the de Bruijn graph is also used in genome assembly, and we elaborate on the connection in Supplementary Note 1). Though these approaches do not work well for divergent genomes, they remain fairly accurate for closely related genomes. For example, Sibelia (Minkin *et al.*, 2013b) can handle repeats and works for many bacterial genomes; unfortunately, it does not scale to larger genomes. However, the last three years has seen a breakthrough in the efficiency of de Bruijn graph construction algorithms (Marcus *et al.*, 2014; Chikhi *et al.*, 2016; Baier *et al.*, 2016; Minkin *et al.*, 2017). 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 alignment of closely related genomes.

In this paper, we describe a novel algorithm SibeliaZ-LCB for identifying collinear blocks in closely related genomes, where the evolutionary distance to the closest common ancestor is no more than 9 PAM units (equivalently, 0.085 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 Cleary *et al.* (2017). 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 (Pham and Pevzner, 2010; Minkin *et al.*, 2013a), 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 previoulsy published TwoPaCo tool (Minkin *et al.*, 2017), then finding locally collinear blocks using using SibeliaZ-LCB, and finally, running a multiple-sequence aligner (spoa, Lee *et al.* (2002)) on each of the found 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 use simulations to test how our method works under different conditions, including various levels of divergence between genomes and different parameter settings.

2 Results

2.1 Accuracy evaluation metrics and strategy

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 (Earl *et al.*, 2014) competition as a starting point. They present several approaches to assess the quality of a multiple whole-genome alignment. Ideally it is best to compare an alignment against a manually curated gold standard; unfortunately, such a gold standard does not exist. The first approach to deal with the lack of a gold standard is to generate simulated genomes for which the true alignment is known, though evolution simulators have their own limitations. The second approach is to use statistical measures (Prakash and Tompa, 2007; Kim and Ma, 2011; Sela *et al.*, 2015) to evaluate the quality of the alignment without external ground-truth information. The third approach is to use external biological information, like gene annotations and homologies, to evaluate the alignments. This way an alignment can be evaluated in a biological context, but the external information usually represents only a fraction of the homology relationships between all genome basepairs. The fourth approach is to evaluate the quality of downstream analysis that uses the alignment; however, these results tend to be specific to the type of downstream analysis and do not generalize well.

Following the footsteps of the Alignathon project, we first evaluated SibeliaZ on simulated data. We employed their precision and recall metrics, as implemented by the mafTools package (Earl *et al.*, 2014). 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 accuracy on real data, we relied on the third approach described above and compared our results against annotations of protein-coding genes. In addition to computing recall as defined above, we also measured coverage, which is the percent of the genome sequence that is included in the alignment. We could not measure the precision directly, since the annotation only included genes. A good proxy for precision could have been a statistical analysis of the alignment blocks (i.e. the second approach), implemented by the the PSAR tool (Kim and Ma, 2011). However, Earl *et al.* (2014) showed that PSAR score does not have a linear correlation with the true precision. Instead, we define our own score, corresponding to the percentage of nucleotide pairs that are identical in an alignment column. In an alignment of highly-similar genomes that has high precision, we expect that the column scores in the alignment blocks are very high. Formally, given a column C of a multiple whole-genome alignment, its score $f(C)$ is computed as follows:

$$f(C) = \sum_{x \in C} \sum_{y \in C} I[x = y] / \binom{|C|}{2}$$

The variable $I[x = y]$ in the formula above is equal to 1 only if both x and y are the same valid DNA characters and 0 otherwise.

2.2 Evaluated tools

We benchmarked the performance of SibeliaZ against three other programs. First, we compared SibeliaZ with its predecessor Sibelia (Minkin *et al.*, 2013b), the state-of-the-art de Bruijn-graph-based synteny finder. To do so, we created a pipeline analogous to SibeliaZ, consisting of the

original Sibelia and the global aligner spoa. Although Sibelia is a program for finding synteny, it is highly flexible and it is possible to tune its parameters to mimic the multiple whole-genome alignment setting. Second, we compared against Progressive Cactus (Paten *et al.*, 2011) and MultiZ+TBA (Blanchette *et al.*, 2004), two multiple whole-genome aligners that were shown to have superior sensitivity compared to other approaches and could scale to the large flies datasets in the Assemblathon. Other multiple aligners (Dubchak *et al.*, 2009b; Darling *et al.*, 2010; Angiuoli and Salzberg, 2011) 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 (Dewey, 2007) 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. The details about the tools' parameters, as well as the exact command lines used, are given in Supplementary Note 2.

2.3 Sixteen mice genomes

We evaluated the ability of SibeliaZ to align real genomes by running it on several datasets consisting of varying number of mice genomes. We retrieved 16 mice genomes available at GenBank (Benson *et al.*, 2017) 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 (Lilue *et al.*, 2018) (Table S1). The genomes vary in size from 2.6 to 2.8 Gbp and the number of scaffolds (between 2,977 and 7,154, except for the reference, which has 377). We constructed 4 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 Table S1, with the genome 1 being the reference genome. We compared against Cactus, since Sibelia does not support such large datasets and Multiz+TBA did not finish on even the smallest dataset (we terminated the program after a week).

The running times of SibeliaZ and Cactus are shown on Figure 1 (Table S2 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 8 and 12 hours, respectively, while Cactus did not finish (we terminated it after a week). For SibeliaZ, we note that the global alignment with spoa takes 42-62% of the running time, and, for some applications (e.g. rearrangement analysis), this step may be omitted to save time.

To evaluate the recall of the results, we retrieved all pairs of homologous protein-coding gene sequences from Ensembl and then computed pairwise global alignments between them using LAGAN (Brudno *et al.*, 2003). The alignment contains both orthologous and paralogous genes, though most of the paralagous 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 (Perry, 2018). We made these filtered alignments available for public download from our github repository[1]. We define the nucleotide identity of an alignment as the number of matched nucleotides divided by the length of the shorter gene. The distribution of nucleotide identities as well as the coverage of the annotation is shown in Figure S1. In our analysis, we binned pairs of genes according to their nucleotide identity. We used this annotation alignment to evaluate the recall of SibeliaZ and Cactus.

[1] `https://github.com/medvedevgroup/SibeliaZ/blob/master/DATA.txt`

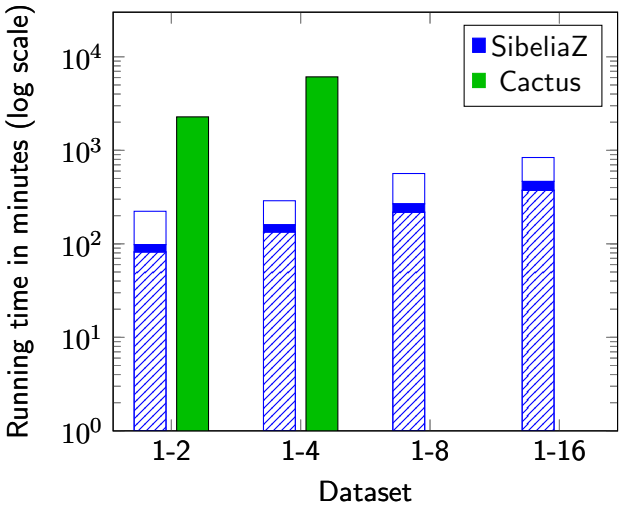

Figure 1: Running times of the different pipelines on the mice datasets (on a log scale). Each bar corresponds to a pipeline. The bar of SibeliaZ is split according to its components: spoa (hatch fill), TwoPaCo (solid fill), and SibeliaZ-LCB (empty fill). Raw numbers are shown in Table S2.

In Table 1, 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 has significantly better recall on paralogous pairs, but slightly worse recall on orthologous pairs. The coverage of both tools is roughly the same, but SibeliaZ has only about half the blocks, indicating a less fragmented alignment. SibeliaZ's coverage and recall decreases only slightly going up to the whole 16 mice dataset, indicating that the recall scales with the number of genomes.

Dataset	N. of blocks		Coverage		Recall			
					Ort. nt. pairs		Par. nt. pairs	
	SibeliaZ	Cactus	SibeliaZ	Cactus	SibeliaZ	Cactus	SibeliaZ	Cactus
1-2	2,031,729	4,228,063	0.88	0.85	0.96	0.97	0.78	0.19
1-4	2,587,182	6,133,662	0.86	0.84	0.96	0.97	0.80	0.03
1-8	3,059,048	-	0.87	-	0.96	-	0.73	-
1-16	4,373,981	-	0.86	-	0.95	-	0.71	-

Table 1: Properties of the multiple whole-genome alignments computed by SibeliaZ and Cactus from the mice datasets.

We further investigated how the recall behaved as a function of nucleotide identity, for the two-mice dataset (Figure 2). As expected, recall decreased with nucleotide identity, though SibeliaZ's recall remained above 90% for nucleotides from similar (80-100% identity) orthologous genes. Cactus generally retains slightly higher recall, except for the case of nucleotides from highly similar ($\geq 90\%$) paralogous genes. In this case, Cactus' recall drops to below 20%, while SibeliaZ's recall is 78%.

We also observed that for both Cactus and SibeliaZ, the recall for paralagous nucleotides is lower than for orthologous ones (Figure 2). Based on manual inspection, we hypothesize that the problem is likely due to the complex repeat structures embedded inside these genes. These repeat

structures of high and varying multiplicity make alignment of those genes challenging, despite their high nucleotide identity.

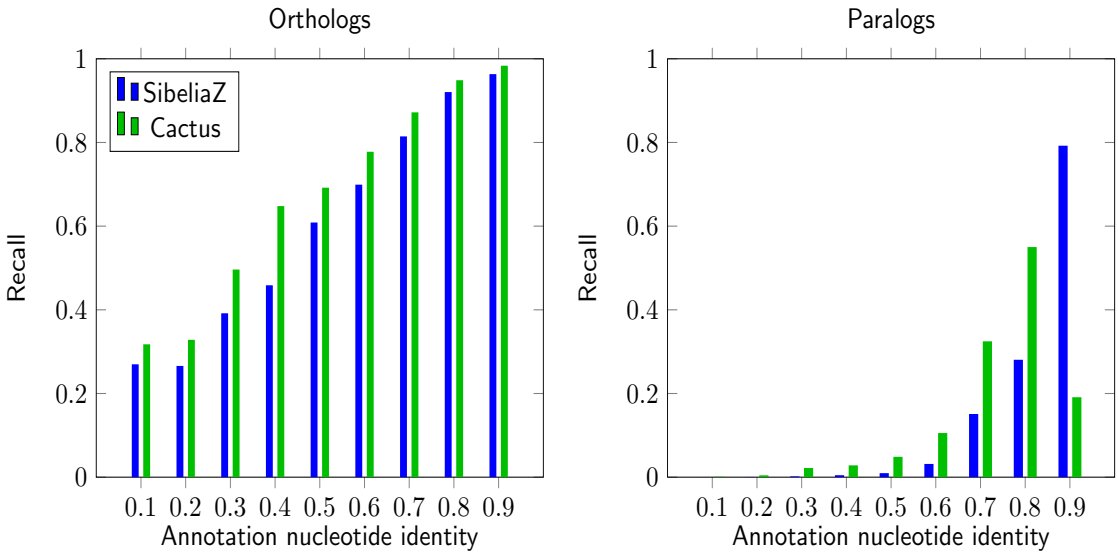

Figure 2: Recall of orthologous (left panel) and paralogous (right panel) nucleotide pairs, on the 1-2 mouse dataset.

Since the gene annotation constitutes only a fraction of all the homologous nucleotides in the genomes, it cannot be used to access the precision. However, as a sanity check, we compute the scores $f(C)$ for the alignment columns (Figure S2). SibeliaZ's alignment has a high degree of similarity: 90% of the alignment columns have a score $f(C)$ a of 0.9 or above, which is what we would expect from aligning closely related genomes.

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. Figure S3 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 inferred family size of 72) corresponds to a single large gene family on the Y chromosome (PTHR19368) and actually has relatively high recall.

2.4 Generation of simulated data

We used simulated data in order to understand the role of a dataset's genomic distance and of our parameter settings, directly measure the precision of the tools, and, in general, to compare SibeliaZ against tools which could not scale to the mice genomes. We used ALF (Dalquen *et al.*, 2011) for simulation because it simulates point mutations as well as genome-wide events such as inversions, translocations, fusions/fissions, gene gain/loss, and lateral gene transfer. Furthermore, ALF is useful for benchmarking as it also produces an alignment which represents the true homology

between the genomes, making it possible to directly assess the precision and recall. We simulated nine datasets, each one consisting of 10 bacterial genomes. Each genome is composed o 1500 genes and of size approximately 1.5 Mbp. We used such relatively small datasets to allow us to efficiently explore the parameter space. Each of the 9 datasets corresponded to a different parameter for distance from the root to leaf species, which we varied from 1 to 24 PAM units. PAM units are a standard measure of divergence between genomes used in the literature. The PAM range used here corresponds to a substitution rate from 0.01 to 0.2. For genome-wide events, we used ALF's default rates. Links to download the the simulation parameter files, the simulated genomes, and their alignments are are available at the github repository[2].

To measure performance on a larger simulated dataset, we also used a simulated dataset from Alignathon with small root-to-leaf divergence, called "primates" in (Earl *et al.*, 2014). In this dataset, the distance from the root to the leaves in the phylogenetic tree is equal to 0.02 substitutions per site, or approximately 2 PAM units. 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 (Earl *et al.*, 2014) since its divergence is outside the target range of SibeliaZ.

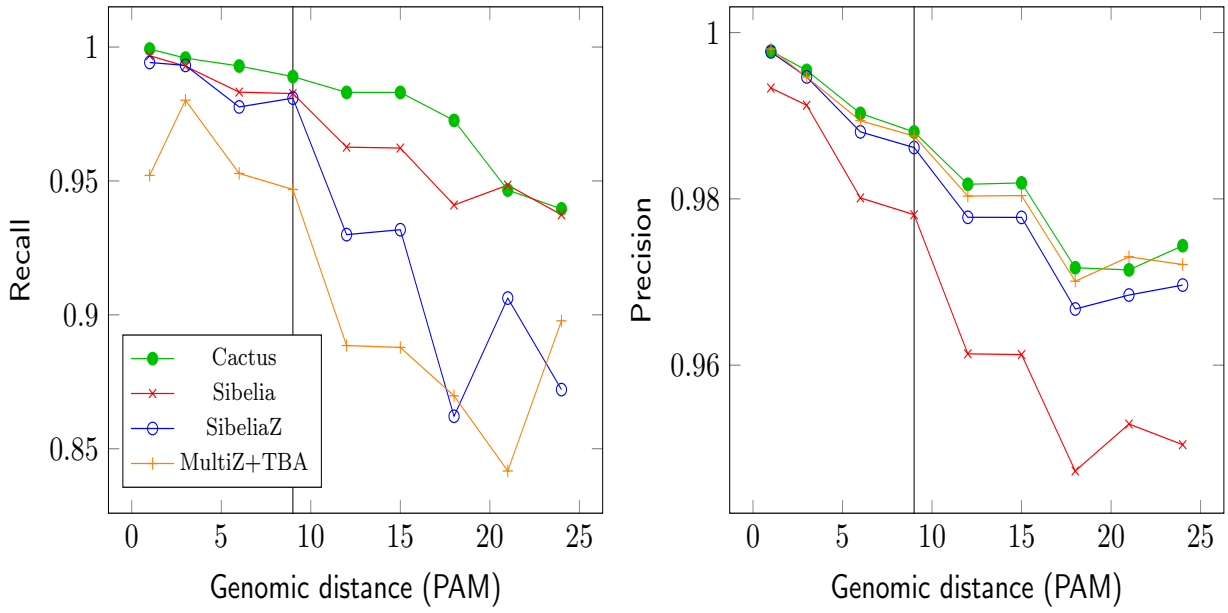

Figure 3: Alignment accuracy of the different pipelines on the simulated bacterial datasets. The vertical line at 9 PAM denotes the target range of SibeliaZ.

2.5 Effect of divergence and k-mer size

SibeliaZ-LCB is based on the analysis of genomic k-mers using the de Bruijn graph. When the divergence between genomes is high, the frequency of shared k-mers is not sufficiently high to recover homologous regions. To test the level of divergence which SibeliaZ can tolerate, we looked at SibeliaZ's precision and recall for the various simulated datasets (Figure 3). For these experiments, $k = 15$ was used. This should give the best attainable accuracy, since lower values would generate many genomic locations with identical k-mers due to simple chance. Figure 3 shows that the recall

[2] `https://github.com/medvedevgroup/SibeliaZ/blob/master/DATA.txt`

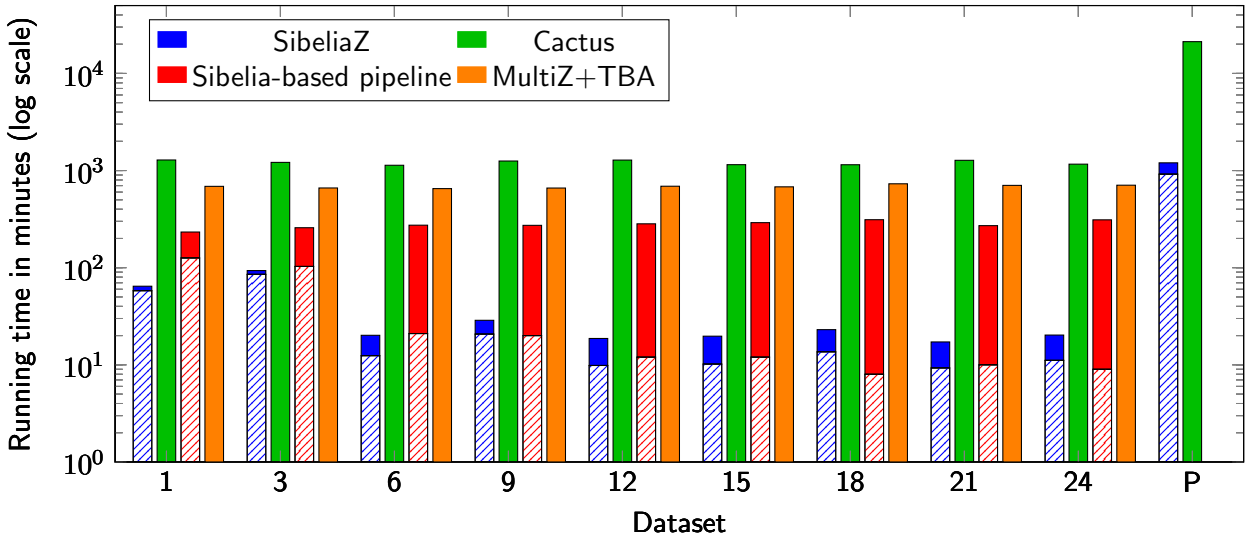

Figure 4: Running times of the different pipelines on the simulated datasets. Each bar corresponds to a pipeline. The hatched part of SibeliaZ and the Sibelia pipeline indicates the part of the running time taken up by spoa. Numerical labels correspond to the datasets simulated by ALF and denote the genomic divergence, while "P" denotes the "primates" dataset from Alignathon.

deteriorates significantly for PAM values greater than 9, hence we do to recommend SibeliaZ for such divergent datasets.

The overall runtime (Figure 4 and Table S3) generally decreases with increasing divergence, albeit only slightly after PAM of 6. Memory use also generally decreases up to PAM of 12, after which it remains constant (Table S4). However, the divergence of the genomes does not seem to consistently affect the graph construction (TwoPaCo) and block finding (SibeliaZ-LCB) runtime and memory usage. The overall decrease in these resources' utilization is due to the global alignment component of SibeliaZ (spoa). This is expected, since higher divergence results in blocks that are shorter and hence easier for spoa to align.

Figure S4 shows the effects of k on recall and precision on the datasets within our target PAM range. As expected, increasing k leads to lower recall, more so for larger PAM. Precision remains mostly unchanged. However, for large datasets such as the mice, smaller values of k lead to denser graphs and hence more time- and memory-consuming performance. We therefore recommend two values of k for the practical usage. For a small dataset, $k = 15$ is the best choice since it yields the highest recall (we used this for the bacterial data). This value is impractical for large datasets 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 the mouse and primates data). In general, we recommend the user to use Figures 3 and S4 to guide the application of SibeliaZ to their own dataset.

2.6 Comparison against other tools on simulated data

SibeliaZ was substantially faster than other tools on all datasets (Figure 4 and Table S3). Using the 9 PAM dataset as an example, SibeliaZ was nine times faster than the Sibelia pipeline, 43 times faster than Cactus and 23 times faster than Multiz+TBA. On the larger primates dataset, SibeliaZ was 17 times faster than Cactus and consumed 33% less memory, while both Sibelia and MultiZ+TBA could not finish in a week. Comparing only the collinear block reconstruction times

(i.e. excluding spoa), SibeliaZ-LCB (including graph construction) is at least 17 times faster than Sibelia.

The speed improvements come at a small cost to accuracy (Figure 3). Focusing on SibeliaZ intended range of ≤ 9 PAM units, SibeliaZ's recall is 1-2 percentage points lower than Cactus, about the same as Sibelia, and 2-4 points higher than MultiZ+TBA. SibeliaZ's precision was about the same as Cactus and Sibelia, and 0.5-1 percentage points higher than MultiZ+TBA. On the primates, SibeliaZ had the same precision as Cactus (95%) but slightly lower recall (95% versus 97%). Overall, our simulations indicate that Cactus is still the preferred tool when the datasets are smaller and there is enough access to compute resources.

3 Methods

3.1 Preliminaries

First, we will define the de Bruijn graph and related objects. Given a positive integer k, 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 multiedge, and every multiedge corresponds to a unique location in the input. Two edges are *parallel* if they have the same endpoints. Parallel edges are not considered identical. The de Bruijn graph can also be constructed from a set of sequences $S = \{s_1, \ldots, s_n\}$. 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. 5 for an example.

The set of a multiedges in a graph G is denoted by $E(G)$. We write (u, v) to denote a multiedge from vertex u to v. A *walk* p is a sequence of multiedges $((v_1, v_2), (v_2, v_3), \ldots, (v_{|p|-1}, v_{|p|}))$ where each multiedge (v_i, v_{i+1}) belongs to $E(G)$. The length of the walk p, denoted by $|p|$, is the number of multiedges it contains. The last multiedge of a walk p is denoted by $end(p)$ and the first one by $start(p)$. A *simple path* is a walk that visits each vertex at most once. We use the term *path* to refer to a simple path.

In a de Bruijn graph, a given multiedge x was generated by a $(k+1)$-mer starting at some position j of some string s_i. We denote $gen(x) = i$ and $pos(x) = j$. For a multiedge x, its successor, denoted by $next(x)$, is a multiedge y such that $gen(x) = gen(y)$ and $pos(y) = pos(x)+1$. Note that a successor does not always exist. A walk $p = (x_1, \ldots, x_{|p|})$ is *genomic* if $next(x_i) = x_{i+1}$ 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 $q = (y_1, \ldots, y_{|q|})$ such that $y_1 = next(end(p))$ 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. The concatenation of two walks x and y is a walk (if it exists) xy consisting of multiedges of x followed by multiedges of y.

3.2 Problem formulation

In this section, we will define the collinear block reconstruction problem in de Bruijn graphs. A *collinear block* is a set of multiedge-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 multiedge-disjoint and have the largest score.

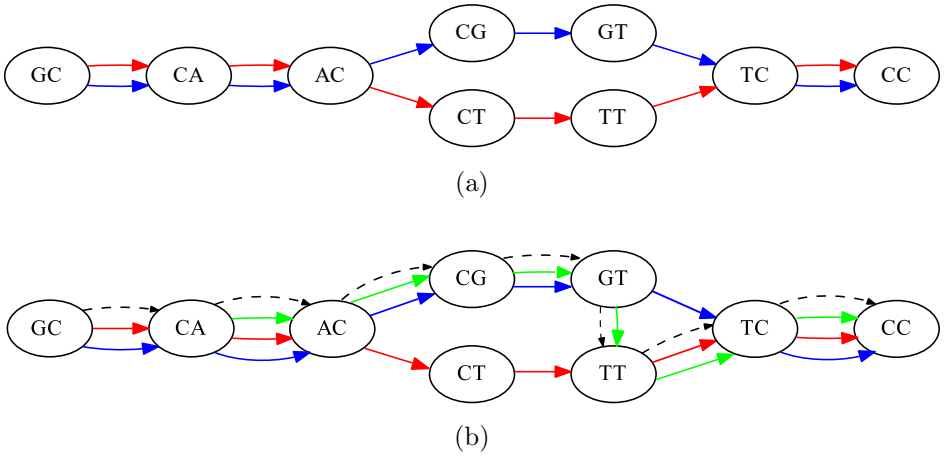

Figure 5: The de Bruijn graph and an example of a collinear block. (a) The graph built from strings "GCACGTCC" and "GCACTTCC", with $k = 2$. The two strings are reflected by the blue and red walks, respectively. This is an example of a collinear block from two walks. There are four bubbles. The bubble formed by vertices "AC" and "TC" describes a substitution within the block, while three other bubbles are formed by parallel edges. The blue and red walks form a chain of four consecutive bubbles. (b) An example of more complex block, where we have added a third sequence "CACGTTCC" (green) 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 corresponds to a hypothetical common ancestor of the three collinear walks. The ancestral path shares many vertices with the three extant walks, and each walk forms its own chain with the ancestral path. The task of finding good collinear blocks can then be framed as finding ancestral paths that form good chains with the extant walks.

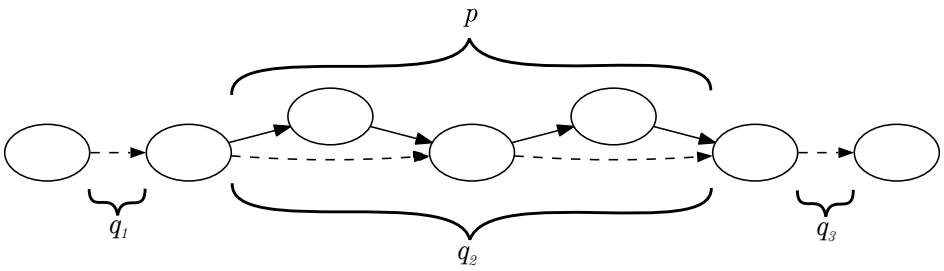

Figure 6: An example of computing the score of a walk p (solid) relative to an ancestral path $p_a = q_1q_2q_3$ (dashed). The path p forms a chain with the subpath q_2 of p, while subpaths q_1 and q_3 form "hanging ends". We count the length of p and subtract lengths of the hanging ends. Thus, the score $f(p_a, p) = 4 - 1 - 1 = 2$.

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; (3) $|x| \leq b$ and $|y| \leq b$ where b is a parameter. A *chain* $c = ((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$ is a sequence of bubbles such that $x = x_1 x_2 \ldots x_n$ and $y = y_1 y_2 \ldots y_n$ 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. For an example of a bubble and a chain, see Fig. 5.

The subgraph resulting from more than two collinear walks can be complex, and there are several ways of capturing it. 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, there should be some ancestral path $p_a = G(s_a, k)$ through the graph forming a long chain with each walk p in the collinear block. We require the chains to be longer than a predefined threshold to avoid confusing spuriously similar sequences with true homologs. At the same time, a collinear walk may only partially form a chain with the ancestral path, leaving "hanging ends" at the ends of the ancestral path, which is undesirable since it implies that these graphs originate from dissimilar sequences. We formalize this intuition by introducing a scoring function quantifying how well an ancestral path describes a collection of the collinear walks. The function rewards long chains formed by the ancestral path and a collinear walk and penalizes the hanging ends. Given an ancestral 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 $p_a = q_1 q_2 q_3$. Recall that m is the parameter denoting the minimum length of a collinear block, and b is the maximum bubble size (in practice, we typically set $m = b$). We define the score $f(p_a, p)$ as:

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

Note that the longer the part of p_a forming a chain with p, the higher the score is. At the same time, the score is reduced 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 showed better performance compared to other alternatives. Walks where the hanging ends are too long are forbidden under this score function (given a score of $-\infty$), and walks that weave through p_a but are too short are ignored (given a score of 0). Fig. 6 shows an example of computing the score.

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

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

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 a multiedge. 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 an ancestral path capturing it, refer to Fig 5b.

3.3 The collinear blocks reconstruction algorithm

Algorithm 1 *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: $\mathcal{P} \leftarrow \emptyset$ ▷ Collinear blocks
2: $G \leftarrow G(S,k)$ ▷ Construct the multigraph
3: **for** all distinct pairs $(u,v) \in E(G)$ **do** ▷ Check possible seeds
4: Initialize the current ancestral path p_a with (u,v)
5: $P \leftarrow \emptyset$ ▷ Sorted set of collinear walks forming chains with p_a
6: $P_{\text{best}} \leftarrow \emptyset$ ▷ Highest-scoring collinear block induced by p_a
7: **for** multiedges $x \in E(G)$ 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** ▷ Extend the ancestral path as far as possible
10: $Q \leftarrow \{q \mid q$ is the b-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 $w_0, w_1, \ldots, w_{|r|}$, with $w_{|r|} = t$
15: **for** $i \leftarrow 1$ **to** $|r|$ **do**
16: Append (w_{i-1}, w_i) to the ancestral 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 multiedges visited by walks of P_{best} as used
23: **return** $\mathcal{P}$

Our algorithm's main pseudocode is shown in Algorithm 1 and its helper function in Algorithm 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 multiedge in the graph, and tries to extend it into an ancestral path that induces a collinear block with the highest possible collinearity score P_{best}. If the block has a positive score, then P_{best} is added to our collection of collinear blocks $\mathcal{P}$. The algorithm then repeats, attempting to build a collinear block from a different multiedge seed. New collinear blocks cannot use multidges belonging to previously discovered collinear blocks. This process continues until all possible multiedges are considered as seeds. The algorithm is greedy in a 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 multiedge parallel to the seed (including the seed) (lines 7 to 8). These parallel multiedges represent the different occurences 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 to 19). During each phase, the ancestral path p_a is extended using a walk r of length at most b (lines 10 to 14). Next, we try to extend each of the collinear walks in a way that forms chains with the extended p_a (lines 15 to 19). The extension of a seed into a collinear block is also

Algorithm 2 *Update-collinear-walks*

Input: A sorted set of collinear walks P, a vertex w

Output: Updated set P

1: **for** multiedges $x \in E(G)$ 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 $pos(end(p))$ is maximized ▷ Find a walk extendable with x
3: **if** such p exists **then**
4: Truncate q so that $end(q) = x$
5: Append p with q ▷ Lengthen the chain that p forms with p_a
6: **else**
7: Add a new walk consisting of the multiedge x to P
8: **return** P

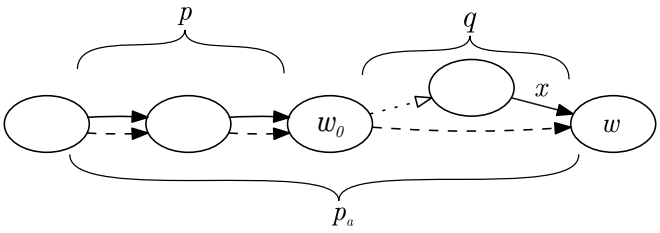

Figure 7: Illustration for Algorithm 2. A collinear walk p (solid) requires an update after the ancestral path p_a is extended with the dashed edge (w_0, w). The path p_a now ends at the vertex w, which has another incoming edge x. Since x is a part of b-extension of p (denoted by q), p can be appended with q to form a longer chain and boost the collinearity score.

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 to 22).

To pick the walk r by which to extend p_a, we use a greedy heuristic (lines 10 to 14). First, we pick the vertex t which we want to extension to reach (lines 10 to 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 comparing 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 $p_a r$. This is done by the function *Update-collinear-blocks* (Algorithm 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 a multiedge 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. Fig. 7 shows an example of updating a collinear walk and Fig. 8 shows a full run of the algorithm for a single seed.

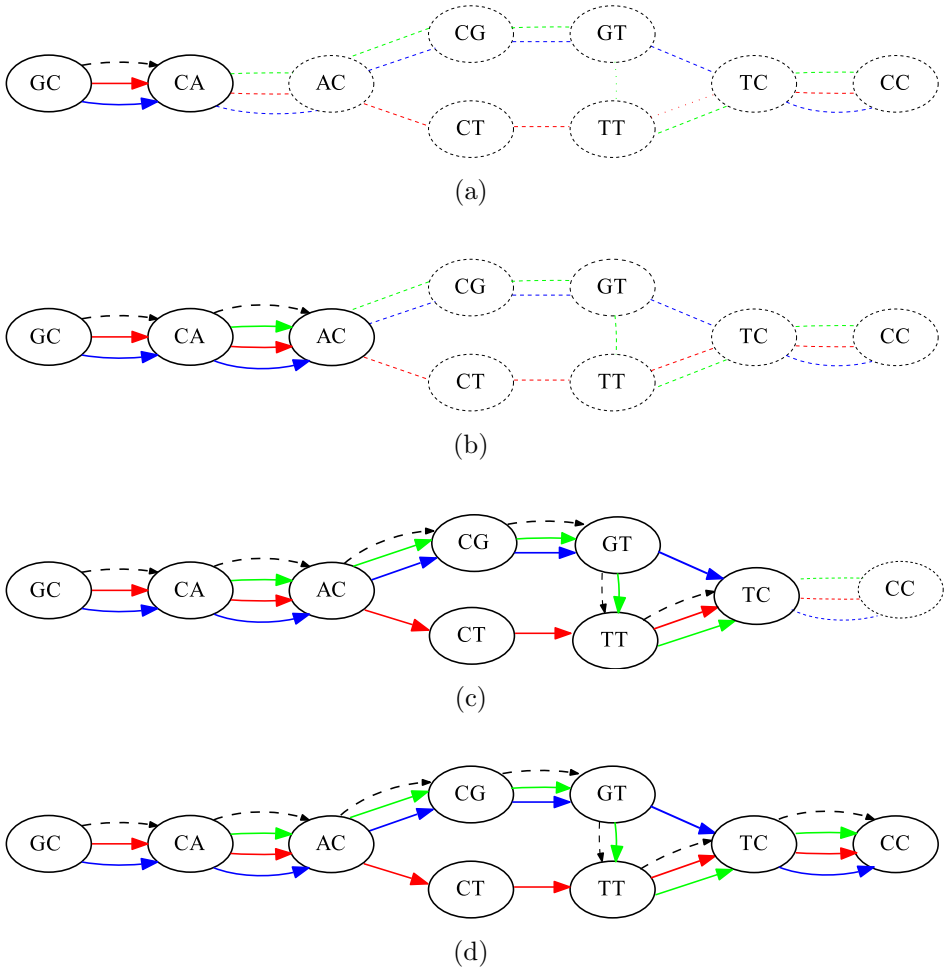

Figure 8: An example of running Algorithm 1 on the graph from Fig. 5b, starting from edge GC $\rightarrow$ CC as the seed. Each subfigure shows the content of the collinear block P and the ancestral path. The collinear walks are solid, the ancestral path is dashed, and the rest of the graph is dotted. Subfigure (a) shows the state of these variables after the initialization; subfigures (b-d) show the state after the completion of each phase.

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 in-coming edges. The case is symmetric and we therefore omit the details.

3.4 Other considerations

For simplicity of presentation, we have described the algorithm in terms of the ordinary de Bruijn graph; however, it is crucial for 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 consist 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 $\ell = v_1, \ldots, v_n$ 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 multiedges $\{v_1 \rightarrow v_2, v_2 \rightarrow v_3, \ldots, v_{n-1} \rightarrow v_n\}$. We efficiently construct the

compacted graph using our previously published algorithm TwoPaCo (Minkin *et al.*, 2017).

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 binary search in line 2 of Algorithm 2.

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 the natural way by using the comprehensive de Bruijn graph, which is defined as $G_{\text{comp}}(s,k) = G(s,k) \cup G(\bar{s},k)$ (Minkin *et al.*, 2017). 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 while loop at line 9 of Algorithm 1. This loop is not embarrassingly parallelizable, since two threads can start exploring two seeds belonging to the same ancestral path. In such a case, there will be a collision on the data structure used to store used marks. To account for this situation, we create locks on multiedges so that when a collision is detected, one thread truncates its block accordingly. We utilized the Intel Threading Building Blocks (TBB) library for implementing parallelism.

4 Discussion

In this paper, we presented a novel 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 the ancestral copy of each collinear block and identifies the induced collinear walks. We assume that the collinear walks share many vertices with this ancestral path and form so-called chains of bubbles. Each ancestral path and the induced block is found greedily, using a scoring function that measures how close this ancestral path is to all the sequences in the block.

The main strength of our approach is speed — we achieve drastic speedups on a small dataset

compared to the state-of-the-art Progressive Cactus aligner (Paten *et al.*, 2011). On 16 mice genomes, SibeliaZ ran in under 15 hours, while Progressive Cactus is not able to complete for even 8 mice genomes, within seven days. Although Progressive Cactus showed better recall on the simulated data, especially for highly divergent genomes, SibeliaZ was able to recall more position pairs belonging to highly-similar paralogous genes on the real dataset. While Cactus remains the best option for smaller or highly divergent datasets, SibeliaZ is the only tool that can scale to many large, closely-related genomes using reasonable computing resources.

We have demonstrated how SibeliaZ-LCB can be incorporated into a whole-genome alignment pipeline, but it can be also used in any application which requires the construction of collinear blocks without the alignment. Such applications mostly stem from studies of genome rearrangements, which can be applied to study breakpoint reuse (Pevzner and Tesler, 2003b), ancestral genome reconstruction (Kim *et al.*, 2017) and phylogenetic studies (Luo *et al.*, 2012). Locally collinear blocks are also a required input for scaffolding tools using multiple reference genomes (Kim *et al.*, 2013; Kolmogorov *et al.*, 2014; Chen *et al.*, 2016; Aganezov and Alekseyev, 2016). For such applications output of SibeliaZ-LCB can be used either directly or after postprocessing by a synteny block generator (Pham and Pevzner, 2010; Proost *et al.*, 2011).

There are several important research directions. Improving the accuracy to match that of Sibelia and Cactus would be beneficial to downstream analyses. A formal analysis of SibeliaZ-LCB's runtime would also be interesting, but doing it in a useful way is a challenge. The worst-case runtime does not reflect the actual running time; moreover, we observed that the run-time depends on the 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 ancestral path using exhaustive enumeration, but the search space even for a small realistic example is too big. We suspect that a polynomical time optimal solution is not possible, but the computational complexity of our problem is open.

SibeliaZ is the first multiple whole-genome aligner that can run in 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, tens of thousands species will have a reference genome available. Furthermore, the sequencing and assembly of various sub-species and strains will be the next logical step for many comparative genomics studies. SibeliaZ makes a significant leap towards the analysis of such datasets.

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; and the Ensembl support team for helping us with retrieving the gene annotations.

Funding

This work has been supported in part by NSF awards DBI-1356529, CCF-1439057, IIS-1453527, and IIS-1421908 to PM. Research reported in this publication was supported by the National Institute Of General Medical Sciences of the National Institutes of Health under Award Number R01GM130691. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

References

Aganezov, S. and Alekseyev, M. A. (2016). Multi-genome scaffold co-assembly based on the analysis of gene orders and genomic repeats. In *International Symposium on Bioinformatics Research and Applications*, pages 237–249. Springer.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, **215**(3), 403–410.

Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, **25**(17), 3389–3402.

Angiuoli, S. V. and Salzberg, S. L. (2011). Mugsy: fast multiple alignment of closely related whole genomes. *Bioinformatics*, **27**(3), 334–342.

Baier, U., Beller, T., and Ohlebusch, E. (2016). Graphical pan-genome analysis with compressed suffix trees and the burrows-wheeler transform. *Bioinformatics*, **32**(4), 497–504.

Beller, T. and Ohlebusch, E. (2016). A representation of a compressed de bruijn graph for pan-genome analysis that enables search. *Algorithms for Molecular Biology*, **11**(1), 20.

Benson, D. A., Cavanaugh, M., Clark, K., Karsch-Mizrachi, I., Ostell, J., Pruitt, K. D., and Sayers, E. W. (2017). Genbank. *Nucleic acids research*.

Blanchette, M., Kent, W. J., Riemer, C., Elnitski, L., Smit, A. F., Roskin, K. M., Baertsch, R., Rosenbloom, K., Clawson, H., Green, E. D., *et al.* (2004). Aligning multiple genomic sequences with the threaded blockset aligner. *Genome research*, **14**(4), 708–715.

Brudno, M., Do, C. B., Cooper, G. M., Kim, M. F., Davydov, E., Green, E. D., Sidow, A., Batzoglou, S., Program, N. C. S., *et al.* (2003). Lagan and multi-lagan: efficient tools for large-scale multiple alignment of genomic dna. *Genome research*, **13**(4), 721–731.

Chen, K.-T., Chen, C.-J., Shen, H.-T., Liu, C.-L., Huang, S.-H., and Lu, C. L. (2016). Multi-car: a tool of contig scaffolding using multiple references. *BMC bioinformatics*, **17**(17), 469.

Chikhi, R., Limasset, A., and Medvedev, P. (2016). Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, **32**(12), i201–i208.

Cleary, A., Kahanda, I., Mumey, B., Mudge, J., and Ramaraj, T. (2017). Exploring frequented regions in pan-genomic graphs. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 89–97. ACM.

Dalquen, D. A., Anisimova, M., Gonnet, G. H., and Dessimoz, C. (2011). Alfa simulation framework for genome evolution. *Molecular biology and evolution*, **29**(4), 1115–1123.

Darling, A. C., Mau, B., Blattner, F. R., and Perna, N. T. (2004). Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome research*, **14**(7), 1394–1403.

Darling, A. E., Mau, B., and Perna, N. T. (2010). progressivemauve: multiple genome alignment with gene gain, loss and rearrangement. *PloS one*, **5**(6), e11147.

Dewey, C. N. (2007). Aligning multiple whole genomes with mercator and mavid. In *Comparative genomics*, pages 221–235. Springer.

Dewey, C. N. and Pachter, L. (2006). Evolution at the nucleotide level: the problem of multiple whole-genome alignment. *Human Molecular Genetics*, **15**(suppl_1), R51–R56.

Doerr, D. and Moret, B. M. (2018). Sequence-based synteny analysis of multiple large genomes. In *Comparative Genomics*, pages 317–329. Springer.

Dubchak, I., Poliakov, A., Kislyuk, A., and Brudno, M. (2009a). Multiple whole-genome alignments without a reference organism. *Genome research*, **19**(4), 682–689.

Dubchak, I., Poliakov, A., Kislyuk, A., and Brudno, M. (2009b). Multiple whole genome alignments without a reference organism. *Genome research*, pages gr–081778.

Earl, D., Nguyen, N., Hickey, G., Harris, R. S., Fitzgerald, S., Beal, K., Seledtsov, I., Molodtsov, V., Raney, B. J., Clawson, H., Kim, J., Kemena, C., Chang, J.-M., Erb, I., Poliakov, A., Hou, M., Herrero, J., Kent, W. J., Solovyev, V., Darling, A. E., Ma, J., Notredame, C., Brudno, M., Dubchak, I., Haussler, D., and Paten, B. (2014). Alignathon: a competitive assessment of whole-genome alignment methods. *Genome Research*, **24**(12), 2077–2089.

Ernst, C. and Rahmann, S. (2013). Pancake: a data structure for pangenomes. In *OASIcs-OpenAccess Series in Informatics*, volume 34. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Harris, R. S. (2007). *Improved pairwise alignment of genomic DNA*. The Pennsylvania State University.

Holley, G., Wittler, R., and Stoye, J. (2016). Bloom filter trie: an alignment-free and reference-free data structure for pan-genome storage. *Algorithms for Molecular Biology*, **11**(1), 3.

Kent, W. J. (2002). Blatthe blast-like alignment tool. *Genome research*, **12**(4), 656–664.

Kim, J. and Ma, J. (2011). Psar: measuring multiple sequence alignment reliability by probabilistic sampling. *Nucleic Acids Research*, **39**(15), 6359–6368.

Kim, J., Larkin, D. M., Cai, Q., Zhang, Y., Ge, R.-L., Auvil, L., Capitanu, B., Zhang, G., Lewin, H. A., Ma, J., *et al.* (2013). Reference-assisted chromosome assembly. *Proceedings of the National Academy of Sciences*, **110**(5), 1785–1790.

Kim, J., Farré, M., Auvil, L., Capitanu, B., Larkin, D. M., Ma, J., and Lewin, H. A. (2017). Reconstruction and evolutionary history of eutherian chromosomes. *Proceedings of the National Academy of Sciences*, **114**(27), E5379–E5388.

Kolmogorov, M., Raney, B., Paten, B., and Pham, S. (2014). Ragouta reference-assisted assembly tool for bacterial genomes. *Bioinformatics*, **30**(12), i302–i309.

Laing, C., Buchanan, C., Taboada, E. N., Zhang, Y., Kropinski, A., Villegas, A., Thomas, J. E., and Gannon, V. P. (2010). Pan-genome sequence analysis using panseq: an online tool for the rapid analysis of core and accessory genomic regions. *BMC bioinformatics*, **11**(1), 461.

Lee, C., Grasso, C., and Sharlow, M. F. (2002). Multiple sequence alignment using partial order graphs. *Bioinformatics*, **18**(3), 452–464.

Lilue, J., Doran, A. G., Fiddes, I. T., Abrudan, M., Armstrong, J., Bennett, R., Chow, W., Collins, J., Collins, S., Czechanski, A., *et al.* (2018). Sixteen diverse laboratory mouse reference genomes define strain-specific haplotypes and novel functional loci. *Nature genetics*, **50**(11), 1574.

Luo, H., Arndt, W., Zhang, Y., Shi, G., Alekseyev, M. A., Tang, J., Hughes, A. L., and Friedman, R. (2012). Phylogenetic analysis of genome rearrangements among five mammalian orders. *Molecular phylogenetics and evolution*, **65**(3), 871–882.

Marcus, S., Lee, H., and Schatz, M. C. (2014). Splitmem: a graphical algorithm for pan-genome analysis with suffix skips. *Bioinformatics*, **30**(24), 3476–3483.

Marschall, T., Marz, M., Abeel, T., Dijkstra, L., Dutilh, B. E., Ghaffaari, A., Kersey, P., Kloosterman, W., Makinen, V., Novak, A., *et al.* (2018). Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, **19**(1), 118–135.

Minkin, I., Pham, H., Starostina, E., Vyahhi, N., and Pham, S. (2013a). C-sibelia: an easy-to-use and highly accurate tool for bacterial genome comparison. *F1000Research*, **2**.

Minkin, I., Patel, A., Kolmogorov, M., Vyahhi, N., and Pham, S. (2013b). *Sibelia: A Scalable and Comprehensive Synteny Block Generation Tool for Closely Related Microbial Genomes*, pages 215–229. Springer Berlin Heidelberg, Berlin, Heidelberg.

Minkin, I., Pham, S., and Medvedev, P. (2017). Twopaco: an efficient algorithm to build the compacted de bruijn graph from many complete genomes. *Bioinformatics*, **33**(24), 4024–4032.

Ng, M.-P., Vergara, I. A., Frech, C., Chen, Q., Zeng, X., Pei, J., and Chen, N. (2009). Orthoclusterdb: an online platform for synteny blocks. *BMC bioinformatics*, **10**(1), 192.

Onodera, T., Sadakane, K., and Shibuya, T. (2013). Detecting superbubbles in assembly graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 338–348. Springer.

Paten, B., Herrero, J., Beal, K., Fitzgerald, S., and Birney, E. (2008). Enredo and pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome research*, **18**(11), 1814–1828.

Paten, B., Earl, D., Nguyen, N., Diekhans, M., Zerbino, D., and Haussler, D. (2011). Cactus: Algorithms for genome multiple sequence alignment. *Genome Research*, **21**(9), 1512–1528.

Paten, B., Novak, A. M., Garrison, E., and Hickey, G. (2017). Superbubbles, ultrabubbles and cacti. In S. C. Sahinalp, editor, *Research in Computational Molecular Biology*, pages 173–189, Cham. Springer International Publishing.

Perry, E. (2018). Personal communication.

Pevzner, P. and Tesler, G. (2003a). Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. *Genome research*, **13**(1), 37–45.

Pevzner, P. and Tesler, G. (2003b). Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. *Proceedings of the National Academy of Sciences*, **100**(13), 7672–7677.

Pham, S. and Pevzner, P. (2010). Drimm-synteny: decomposing genomes into evolutionary conserved segments. *Bioinformatics*, **26**(20), 2509–2516.

Prakash, A. and Tompa, M. (2007). Measuring the accuracy of genome-size multiple alignments. *Genome biology*, **8**(6), R124.

Proost, S., Fostier, J., De Witte, D., Dhoedt, B., Demeester, P., Van de Peer, Y., and Vandepoele, K. (2011). i-adhore 3.0fast and sensitive detection of genomic homology in extremely large data sets. *Nucleic acids research*, **40**(2), e11–e11.

Raphael, B., Zhi, D., Tang, H., and Pevzner, P. (2004). A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*, **14**(11), 2336–2346.

Sakharkar, M. K., Perumal, B. S., Sakharkar, K. R., and Kangueane, P. (2005). An analysis on gene architecture in human and mouse genomes. *In silico biology*, **5**(4), 347–365.

Schwartz, S., Kent, W. J., Smit, A., Zhang, Z., Baertsch, R., Hardison, R. C., Haussler, D., and Miller, W. (2003). Human–mouse alignments with blastz. *Genome research*, **13**(1), 103–107.

Sela, I., Ashkenazy, H., Katoh, K., and Pupko, T. (2015). Guidance2: accurate detection of unreliable alignment regions accounting for the uncertainty of multiple parameters. *Nucleic Acids Research*, **43**(W1), W7–W14.

Sheikhizadeh, S., Schranz, M. E., Akdel, M., de Ridder, D., and Smit, S. (2016). Pantools: representation, storage and exploration of pan-genomic data. *Bioinformatics*, **32**(17), i487–i493.

Tettelin, H., Masignani, V., Cieslewicz, M. J., Donati, C., Medini, D., Ward, N. L., Angiuoli, S. V., Crabtree, J., Jones, A. L., Durkin, A. S., *et al.* (2005). Genome analysis of multiple pathogenic isolates of streptococcus agalactiae: implications for the microbial pan-genome. *Proceedings of the National Academy of Sciences of the United States of America*, **102**(39), 13950–13955.

Vernikos, G., Medini, D., Riley, D. R., and Tettelin, H. (2015). Ten years of pan-genome analyses. *Current opinion in microbiology*, **23**, 148–154.

Zekic, T., Holley, G., and Stoye, J. (2018). Pan-genome storage and analysis techniques. In *Comparative Genomics*, pages 29–53. Springer.

Supplementary Note 1 Other related work

A closely related problem to multiple whole-genome alignment is synteny reconstruction. In this setting, genomes are decomposed into large blocks such that the gene order within each block is preserved. This is similar to locally collinear blocks, but collinear blocks are usually smaller blocks representing single genes or exons (or non-coding DNA). Collinear blocks can be viewed as high resolution synteny blocks and, in general, the distinction between the two concepts can be blurry. For a discussion on representation of synteny blocks at multiple scales, see Minkin *et al.* (2013b). Synteny blocks are often reconstructed from anchors such as genes (Pevzner and Tesler, 2003a; Ng *et al.*, 2009; Pham and Pevzner, 2010; Proost *et al.*, 2011) and, less commonly, from the nucleotide sequences directly (Minkin *et al.*, 2013b; Doerr and Moret, 2018).

A related active research area is data structures for representing pan-genomes (Tettelin *et al.*, 2005). A pan-genome as a collection of related genomes that are to be analyzed jointly. For a review on computational pan-genomics, see (Vernikos *et al.*, 2015; Marschall *et al.*, 2018; Zekic *et al.*, 2018). Constructing a data structure for the efficient storage and querying of a pan-genome is related but tangential to the problem of identifying collinear blocks, which we consider in this paper. Pan-genome data structures are concerned with efficiently representing the homology within the pan-genome, while we focus on fast algorithms for obtaining such homologies. There is naturally some overlap between the two areas, e.g. some pan-genome tools include a multiple whole-genome alignment component (Ernst and Rahmann, 2013; Laing *et al.*, 2010). Others use the de Bruijn graph for representing the pan-genome (Marcus *et al.*, 2014; Holley *et al.*, 2016; Beller and Ohlebusch, 2016; Sheikhizadeh *et al.*, 2016). Our approach also relies on a de Bruijn graph, though we use it as a technique to find collinear blocks rather than to represent them.

De Bruijn graphs are also used in genome assembly. Sequencing errors in reads often leave patterns in the graph (e.g. bubbles) that are similar to those observed when de Bruijn graphs are used for genome comparison. There has been some effort to characterize such patterns more formally using superbubbles (Onodera *et al.*, 2013), and the concepts were then generalized to snarls and ultrabubbles in the context of representing variants in the genome graphs (Paten *et al.*, 2017).

Supplementary Note 2 Parameter details and command lines

We tried to find the optimal parameters for all tools. For Sibelia, which could only run on simulated data, we used the parameter set designed to yield the highest sensitivity (called the "far" set in Sibelia). Progressive Cactus requires a phylogenetic tree in addition to the input genomes. For the simulated datasets, we used the real tree generated by the simulator; for the mice genomes, we used the guide tree from (Lilue *et al.*, 2018). Multiz+TBA were run with default parameters. We could not compile the version of MultiZ+TBA publicly available for download and used a slightly modified version provided by Robert S. Harris. For TwoPaCo and spoa, we set the parameters following the guidelines provided with the respective software.

SibeliaZ's parameter settings for k is described in the main text. For the abundance pruning parameter, we recommend setting it as high as the compute resources allow. In our case, we used $a = 150$. For the maximum length of a bubble's branch (b), we observed that SibeliaZ-LCB is robust for different values. We used $b = 200$ as the default as it led to high accuracy across all tested ranges of k on our simulated data. For the minimum length of a collinear block, we set $m = 50$ as a default, since this is smaller than 93.1% of the known mice exons (Sakharkar *et al.*, 2005) and, more generally, we do not expect most applications to be interested in blocks shorter than 50nt.

We performed all experiments on a machine running Ubuntu 16.04.3 LTS with 512 GB of RAM and a 64 core CPU Intel Xeon CPU E5-2683 v4. We were limited to using at most 32 threads at any given time. Progressive Cactus was run with 32 threads, since the authors recommended to use as many threads as possible for the best performance. MultiZ+TBA and Sibelia are both single-threaded. (There were several submissions to Alignathon which used an extensively parallelized MultiZ or TBA; unfortunately, the software packages used for those submissions are not available publicly for download.) TwoPaCo and SibeliaZ-LCB were run with at most 16 threads. We note that spoa is run on each block, and our software includes a wrapper to automate this.

Here are the exact command lines for the tools we ran.

TwoPaCo:

```
twopaco -k <k_value> -f <bloom_filter_size> -t 16 -o <dbg_graph> <genomes_file>
```

SibeliaZ-LCB:

```
SibeliaZ-LCB --fasta <genomes_file> --graph <dbg_graph> -o <output_directory>
-k <k_value> -b 200 -m 50 -a 150 -t 4
```

spoa:

```
spoa <input_fasta_file> -l 1 -r 1
```

Sibelia:

```
Sibelia <genomes_file> -o <output_directory> -s far --lastk 50 -m 50 --nopostprocess
```

MultiZ:

```
all_bz <guide_tree>
tba <guide_tree> *.*.maf <outputMafFile>
```

Progressive Cactus:

```
runProgressiveCactus.sh --maxThreads 32 <seqFile> <workDir> <outputHalFile>
source ./environment && hal2mafMP.py <outputHalFile> <outputMafFile>
```

All running times and memory usage numbers were obtained using the GNU time utility.

Tables and Figures

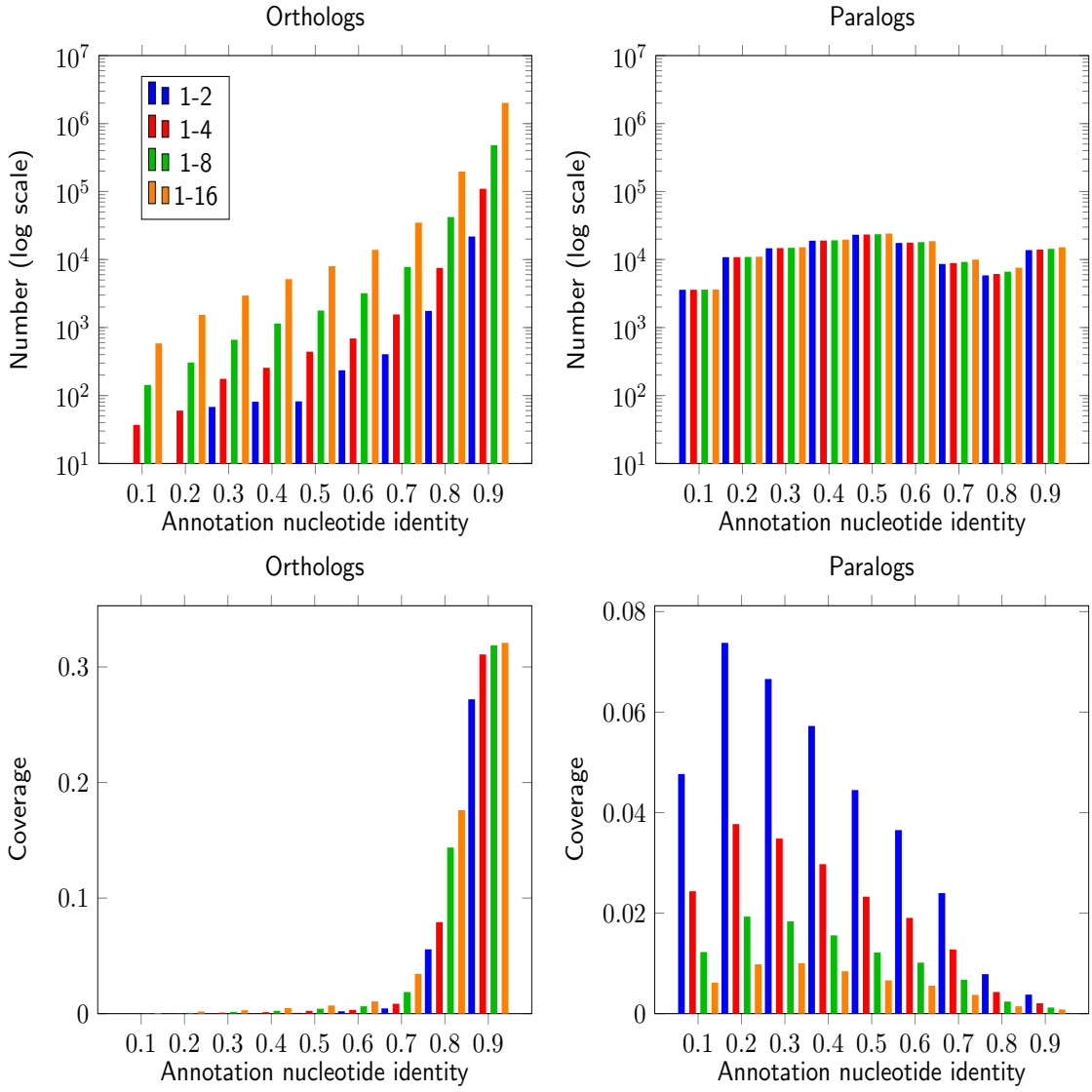

Figure S1: Properties of the pairwise alignments constructed from pairs of homologous protein-coding genes in the various mice datasets.

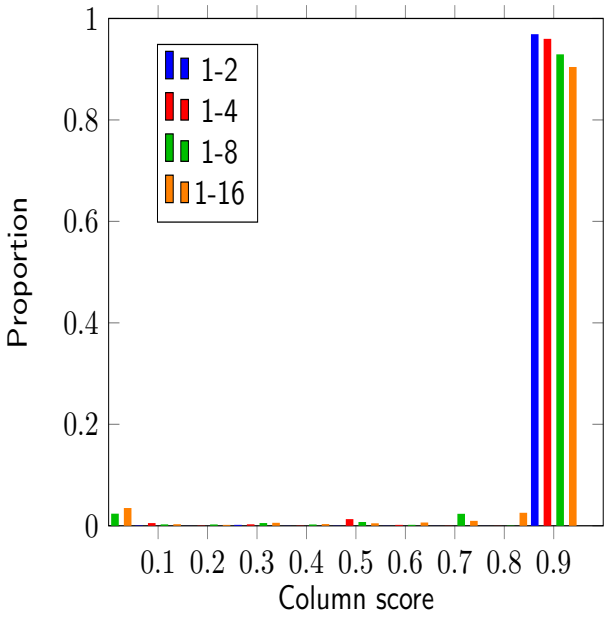

Figure S2: Histogram of the column scores $f(C)$ of the SibeliaZ alignment, for the various mice datasets.

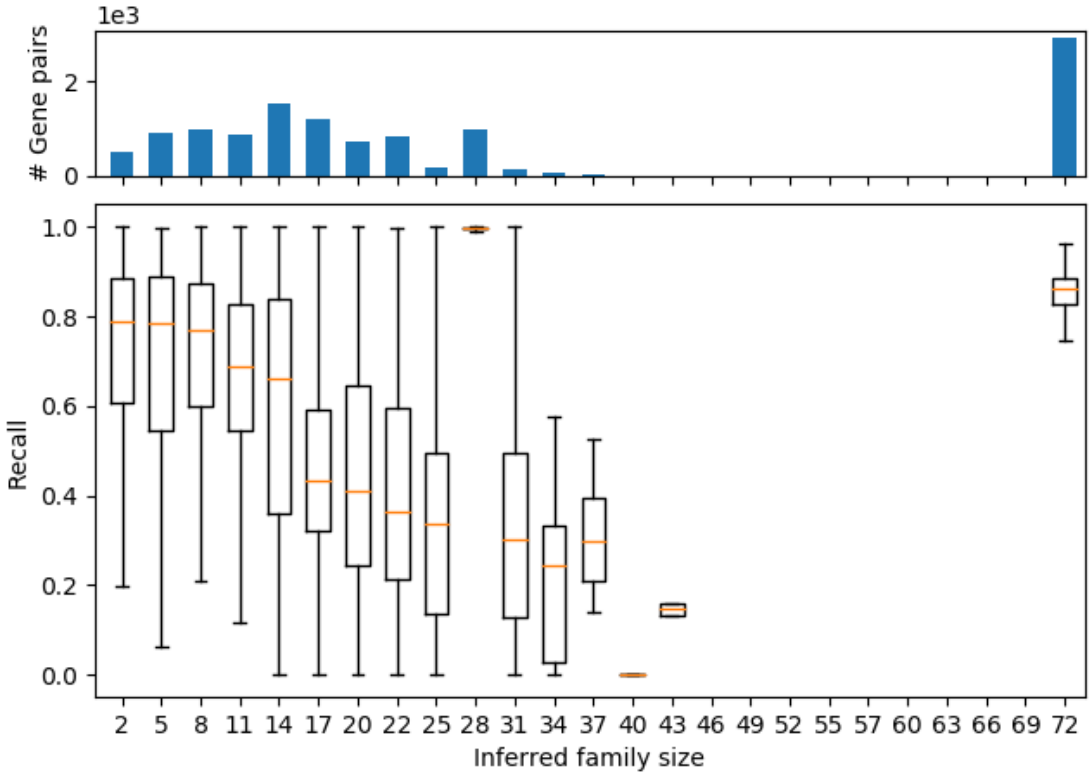

Figure S3: The recall as a function of inferred family size, using the two-mice dataset. The family size is binned into 25 equally sized bins. The top histogram shows the number of gene pairs in each bin.

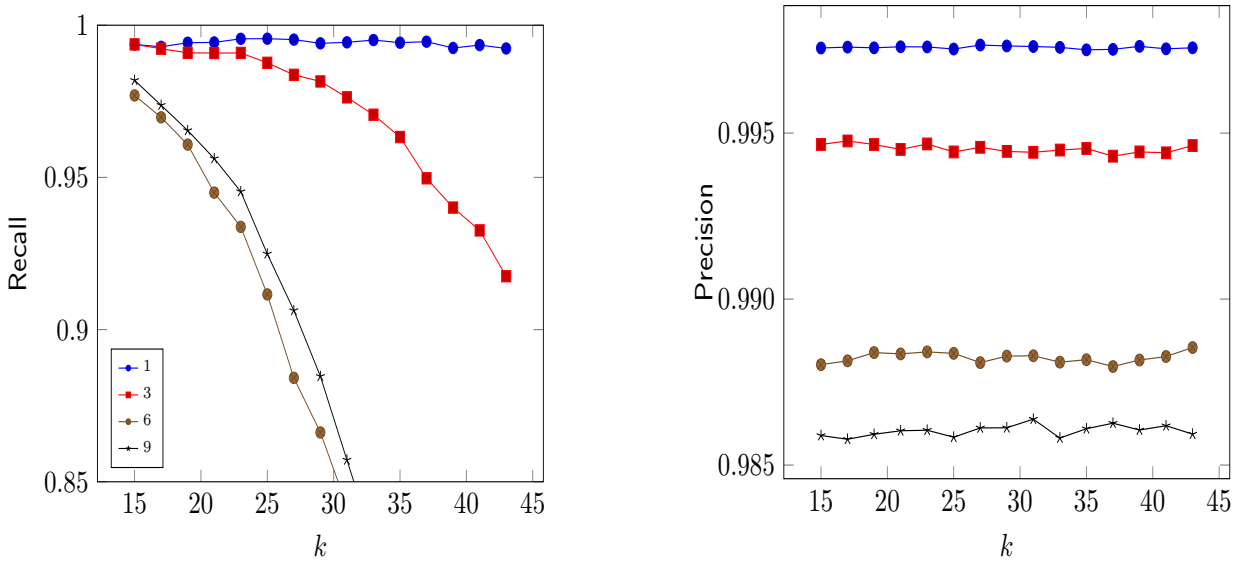

Figure S4: Effects of the parameter k on recall and precision, on the simulated bacterial datasets. Each line corresponds to a dataset with the specified root-to-leaf divergence in PAM units. We show curves only for datasests with divergence less or equal to 9 PAM as they corresponds to the target range of SibeliaZ.

ID	Strain	Size (Mb)	N. Scaffolds	Accession number
1	C57BL/6J	2,819	336	GCA_000001635.8
2	129S1/SvImJ	2,733	7,154	GCA_001624185.1
3	A/J	2,630	4,688	GCA_001624215.1
4	AKR/J	2,713	5,953	GCA_001624295.1
5	CAST/EiJ	2,654	2,977	GCA_001624445.1
6	CBA/J	2,922	5,466	GCA_001624475.1
7	DBA/2J	2,606	4,105	GCA_001624505.1
8	FVB/NJ	2,589	5,013	GCA_001624535.1
9	NOD/ShiLtJ	2,982	5,544	GCA_001624675.1
10	NZO/HiLtJ	2,699	7,022	GCA_001624745.1
11	PWK/PhJ	2,560	3,140	GCA_001624775.1
12	WSB/EiJ	2,690	2,239	GCA_001624835.1
13	BALB/cJ	2,627	3,825	GCA_001632525.1
14	C57BL/6NJ	2,807	3,894	GCA_001632555.1
15	C3H/HeJ	2,701	4,069	GCA_001632575.1
16	LP/J	2,731	3,499	GCA_001632615.1

Table S1: Properties of the assembled mice genomes available at GenBank.

Dataset	SibeliaZ				Cactus
	TwoPaCo	SibeliaZ-LCB	spoa	Total	
1-2	16 (35)	113 (23)	93 (127)	222 (127)	2,279 (38)
1-4	28 (35)	122 (47)	148 (127)	298 (127)	6,105 (90)
1-8	48 (35)	134 (80)	293 (129)	475 (129)	-
1-16	90 (35)	174 (153)	435 (134)	699 (153)	-

Table S2: Running time (minutes) and memory usage (gigabytes, in parenthesis) of SibeliaZ and Cactus on the mice datasets. A dash in a column indicates that the program did not complete within in a week.

Data set	SibeliaZ				Sibelia-based			Cactus	Multiz + TBA
	TwoPaCo	SibeliaZ-LCB	spoa	Total	Sibelia	spoa	Total		
1	5	1	58	64	107	126	233	1,285	688
3	6	2	85	93	155	103	258	1,214	662
6	6	2	12	20	253	21	274	1,134	652
9	6	2	21	29	253	20	273	1,254	661
12	6	3	10	19	271	12	283	1,282	690
15	7	3	10	20	279	12	291	1,150	680
18	7	3	14	24	304	8	312	1,148	731
21	6	2	9	17	261	10	271	1,275	704
24	6	3	11	20	302	9	311	1,164	707
P	110	173	918	1,201	-	-	-	21,234	-

Table S3: Running time (in seconds) on the simulated datasets. A dash in a column indicates that the program did not complete within in a week.

Data set	SibeliaZ				Sibelia-based			Cactus	Multiz + TBA
	TwoPaCo	SibeliaZ-LCB	spoa	Total	Sibelia	spoa	Total		
1	1,168	53	7,092	7,092	427	18,458	18,458	1,326	163
3	1,247	81	8,495	8,495	500	15,815	15,815	770	164
6	1,233	123	993	1,233	508	1,612	1,612	339	153
9	1,312	129	2,336	2,336	517	2,749	2,749	340	164
12	1,203	134	396	1,203	533	530	533	323	133
15	1,191	134	535	1,191	533	530	533	323	133
18	1,233	141	313	1,233	560	505	560	277	146
21	1,193	120	421	1,193	507	530	530	1,295	163
24	1,262	138	252	1,262	546	490	546	683	164
P	1,326	4,748	12,055	12,055	-	-	-	18,422	-

Table S4: Memory consumption (in megabytes) on the simulated datasets. A dash in a column indicates that the program did not complete within in a week.