On Deriving Synteny Blocks by Compacting Elements

Leonard Bohnenkämper*[1][0000−0003−4508−0078], Luca Parmigiani*[1][0000−0002−2139−3259], Cedric Chauve[2][0000−0001−9837−1878], and Jens Stoye[1][0000−0002−4656−7155]
* co-first authors

[1] Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Bielefeld, Germany
{leonard.bohnenkaemper,luca.parmigiani,jens.stoye}@uni-bielefeld.de
[2] Department of Mathematics, Simon Fraser University, Burnaby (BC), Canada
cedric.chauve@sfu.ca

Abstract. Genomic rearrangements are major drivers of evolution and genetic disease. However, studying rearrangements requires segmenting the genomes of interest into conserved regions, called synteny blocks, that highlight structural differences between genomes. Synteny blocks are typically defined from annotated genes or derived as a by-product of whole-genome alignments. As these procedures are heuristic and do not explicitly model rearrangements, they can obscure real variation, create false similarities, and affect phylogenetic inference.
The importance of synteny block definition has long been recognized, as shown for example by discussions on breakpoint reuse, where different definitions of synteny blocks led to different estimates of rearrangement complexity in mammalian genomes.
We present a formal framework for deriving synteny blocks directly from sequence data by partitioning genomic elements into blocks that do not contain breakpoints. A breakpoint is defined between a pair of genomes as an adjacency of shared elements that occurs in one genome but not in the other. Synteny blocks are therefore not allowed to span such boundaries, ensuring that rearrangements are not obscured. The framework is fully agnostic to the type of genomic element and applies to any genome representation expressed as sequences of elements, such as non-overlapping alignments, exact matches (MUMs/MEMs), k-mers, unitigs or minimizers.
We formalize two optimization problems: minimizing the total genome length after replacement by synteny blocks (the Minimum-Length Synteny Block Problem) and minimizing the number of distinct blocks (the Minimum-Size Synteny Block Problem). We show that both problems are NP-hard in general. However, when blocks are required to be collinear and to contain a shared element, we provide a linear-time algorithm with respect to the number of input elements that simultaneously minimizes both objectives. The resulting method is simple, efficient, and produces large synteny blocks without obscuring rearrangements.

Keywords: Comparative genomics · Rearrangements · Synteny blocks.

1 Introduction

Comparative genomics aims to detect and explain similarities and differences between groups of evolutionarily related genomes; examples of such analyses include genome rearrangement studies or phylogeny reconstruction, to cite only two examples. With the availability of thousands of fully assembled genomes, from small viral genomes to large and complex genomes of eukaryotic species, it is now common for comparative studies to consider tens to hundreds of genomes at once. The size and complexity of most genomes prevents that whole-genome comparative studies can be done at the nucleotide level, and a preliminary step is to abstract the considered genomes into more compact representations over an "alphabet" of conserved genome segments called *synteny blocks*, a term coined by Pevzner and Tesler in [1].

Intuitively, synteny blocks partition genomes into non-overlapping segments, each segment belonging to a unique synteny block, such that both differences between segments from the same synteny block and similarities between segments from different synteny blocks can be ignored for the comparative analysis that is done. For example, when comparing genomes to understand their evolution in terms of large-scale genomic rearrangements, synteny blocks aim to be large homologous genome segments that are likely to have evolved from a single ancestral genome segment through nucleotide-level mutations, small-scale duplications and losses, and micro-rearrangements [1, 2, 3, 4]. In practice, synteny blocks are determined from small, highly-conserved genome segments called *elements* (e.g., orthologous genes, unitigs, local pairwise or multiple sequence alignments), that are clustered into synteny blocks according to criteria that are typically parameterized by conservation of content and order; for example, locally collinear synteny blocks would require that all genome segments in a synteny block contain the set of elements in the same order and orientation.

Since the initial comparison between the human and mouse genomes, there has been a large corpus of methods for the computation of synteny blocks, using a wide variety of approaches [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]. Most methods are inherently heuristic, in that they do not rely on a formal definition of what a synteny block is; we refer the reader to the recent papers [20, 21, 23] for an overview of used approaches.

As a consequence of the many ways synteny blocks can be defined, the design of algorithms to identify synteny blocks is both challenging and prone to controversy. This is well illustrated by the debate on the rate of breakpoint reuse by genomic rearrangements which stemmed from initial comparisons of the human and mouse genomes and focused on the choice of synteny blocks [24, 25, 26, 27, 28]. Additionally, in [29], which focused on rearrangement-based phylogenetic reconstruction, it was shown that "the tree inference method had less impact than the block determination method", where impact was measured by the agreement of rearrangement-based trees with a SNP-based reference tree. In a different comparative study of synteny blocks methods [30], Ghiurcuta and Moret argued for "the need for a well founded, systematic approach to the decomposition of genomes into syntenic blocks."

In this work, we first focus in Section 2 on introducing a formal definition of synteny blocks and on formalizing the computation of synteny blocks as two related optimization problems. We define synteny blocks as groups of genomic elements that do not contain breakpoints, where a breakpoint is defined between a pair of genomes as an adjacency of shared elements that occurs in one genome but not in the other. This definition ensures that genome rearrangements are not obscured. Our framework is fully agnostic to the type of elements and applies to any genome representation expressed as sequences of elements, such as genes, exact matches (MUMs/MEMs), k-mers, or unitigs. We then formalize the computation of synteny blocks as two optimization problems. The first one aims to find a set of synteny blocks minimizing the total length of the genomes after they are abstracted as (signed) sequences of synteny blocks (the Minimum-Length Synteny Block Problem), while the second one aims to find a set of synteny blocks minimizing the number of blocks (the Minimum-Size Synteny Block Problem). In Section 3, we then show that both problems are NP-hard in general, but that when blocks are required to be collinear and to contain a conserved element, i.e., an anchor present in all genome segments forming a synteny block, both problems can be solved simultaneously in linear time with respect to the input size, measured as the sum of the elements spelling all genomes. The resulting approach is conceptually simple, runs efficiently, and preserves rearrangements while producing large synteny blocks. In Section 4 we describe experiments that provide a poof-of-principle of our definition of synteny blocks and of the usefulness of our algorithms. In Section 5 we discuss the results and future research directions.

2 Problem Definition

We formally define *synteny blocks* and the related algorithmic problems. For the sake of exposition, we consider unichromosomal linear genomes, although our definitions naturally extend to general genomic architectures.

Genomes and strings. Let E be a set of *elements*, where each element typically represents a family of genomic regions (e.g., k-mers, unitigs, orthologous genes).

A *string* $s = s[1] \dots s[n]$ of length n over E is a sequence of *signed* elements, each of the form $s[i] = oe$, where $e \in E$ and $o \in \{+, -\}$; we write $\mathrm{sgn}(s[i]) = o$ and $\mathrm{elm}(s[i]) = e$. We use $-x$ to indicate a reversal of a signed element x, i.e., $-+e = -e$ and $--e = +e$. The set of unsigned elements in a string s is denoted by $\mathrm{elm}(s) = \{\mathrm{elm}(s[i]) \mid i = 1, \dots, n\}$. We denote the *substring* of string s between positions i and j by $s[i : j] = s[i] \dots s[j]$.

For the moment we consider only strings in which each element occurs once, i.e., we assume that repeats have been masked. Such strings are known as *singular strings*. Our results can be generalized for arbitrary strings, with repeated elements, as outlined at the end of Section 3.2 ("Extension to Duplicates").

Adjacencies and Breakpoints. Rearrangements are usually characterized by their *breakpoints*. Because we aim to define blocks that reflect rearrangements and therefore breakpoints accurately, the definition of breakpoints is central to our notion of synteny blocks.

Each signed element oe is associated with two *extremities*: a *tail* e^t and a *head* e^h. The order of these extremities depends on the sign: if $o = +$, the extremities are ordered (e^t, e^h); if instead $o = -$, the order of the extremities is reversed, (e^h, e^t).

The *adjacency set* of s, $\mathrm{Adj}(s)$, is the set of unordered pairs of extremities connecting consecutive elements:

$$\mathrm{Adj}(s) = \{\, \{\mathrm{elm}(s[i])^x, \mathrm{elm}(s[i+1])^y\} \mid 1 \leq i < n \,\} \cup \{\{\$, \mathrm{elm}(s[0])^y\}, \{\mathrm{elm}(s[n])^x, \$\}\},$$

where, $x = h$ when $\mathrm{sgn}(s[i]) = +$ and $x = t$ otherwise, and $y = t$ when $\mathrm{sgn}(s[i+1]) = +$ and $y = h$ otherwise. We call \$ the *telomere*.

To compare two strings s and t with unequal content, we project them onto their common elements as follows: the *projection* of s onto t, denoted $s|_t$, is the string obtained from s by removing every $s[i]$ for which $\mathrm{elm}(s[i])$ is not contained in $\mathrm{elm}(t)$. Rearrangements can then be characterized by their *breakpoints*. The *breakpoint set* between two strings s and t is the symmetric difference between the two adjacency sets of the projections of s and t onto each other, $\mathrm{Adj}(s|_t)$ and $\mathrm{Adj}(t|_s)$:

$$\mathrm{Bp}(s, t) = \mathrm{Adj}(s|_t) \cup \mathrm{Adj}(t|_s) \setminus (\mathrm{Adj}(s|_t) \cap \mathrm{Adj}(t|_s)).$$

Parsings and Partitions. We aim to define synteny blocks as a partition of the set of elements E. Recall that a partition $P = \{P_1, \dots, P_k\}$ is a set of non-empty sets P_i, called *parts*, such that each element is contained in exactly one part. We call the partition $P = \{\{e\} \mid e \in E\}$, the *finest partition*.

Given a partition P, the maximal substrings in s consisting of elements from the same part P_i form a parsing of s. More formally, given a string s, a *parsing* of s is a decomposition into substrings $s[1 : r_1], s[r_1+1 : r_2], \dots, s[r_{l-1} + 1 : r_l]$. Each $s[r_{i-1} + 1 : r_i]$ of a parsing is called a *phrase*. The parsing *induced* by partition P is the parsing, such that $s[i], s[i+1]$ are in the same phrase if and only if $\mathrm{elm}(s[i])$ and $\mathrm{elm}(s[i+1])$ are in the same part of P. An example is given in Fig. 1.

In the following we define synteny blocks as partitions of E such that (i) phrases of distinct parts do not interleave, (ii) synteny blocks do not contain internal rearrangements (are free of breakpoints), and (iii) phrases can be assigned an orientation. The partition in Fig. 1 fulfills these properties.

Synteny blocks. First, since our input strings are presumed to be singular, we expect synteny blocks to occur only once per string as well, uninterrupted by other blocks. An example where this is not the case is given in Fig. 2 (i). More formally: Given a part P_i and a signed string s, the *index set* of P_i in s is $\{j \mid \mathrm{elm}(s[j]) \in P_i\}$; P is *absent* from s if its index set is empty, *present* otherwise. We call P_i *contiguous* in s if it is absent from s or its index set defines a single substring $s[i : j]$ of s. We then write $s^{P_i} = s[i : j]$ and call s^{P_i} the phrase *induced* by P_i. For a set of strings S, we write $S^{P_i} = \{s^{P_i} \mid P_i \text{ is present in } s \in S\}$.

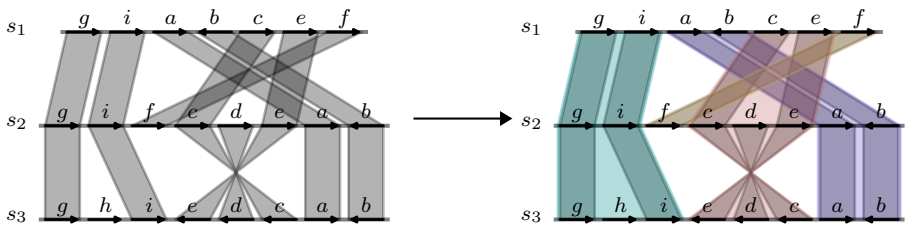

Fig. 1. Illustration of the parsing induced by partition $P = \{P_1 = \{a, b\}, P_2 = \{c, d, e\}, P_3 = \{f\}, P_4 = \{g, h, i\}\}$ for three strings; the corresponding phrases are indicated by four different colors. The parsed strings are $+g+i \mid +a-b \mid +c+e \mid +f$, $+g+i \mid +f \mid +c+d+e \mid +a-b$, and $+g+h+i \mid -e-d-c \mid +a-b$. One possible way of orienting the phrases is to view each as forward except $-e-d-c$ in s_3 which one considers backward.

Second, since we want synteny blocks to reflect the rearrangements in the original strings, we want blocks not to contain breakpoints like shown in Fig. 2 (ii). A contiguous part P_i is called *breakpoint free* on string set S if for any two strings $s, t \in S$, the induced phrases s^{P_i} and t^{P_i} have no breakpoints with respect to each other.

Third, to reflect rearrangements in the original strings, the phrases induced by synteny blocks need to be assigned an orientation. However, this is not always possible, even when the corresponding part is breakpoint free, as shown in Fig. 2 (iii). A part P_i is called *orientable* on string set S if for each of its induced phrases in S^{P_i}, we can assign an orientation $f : S^{P_i} \to \{+, -\}$, such that applying this orientation to $t \in S^{P_i}$, i.e., inverting it if $f(t) = -$, each element occurs only with one orientation. More formally, if $+e \in \{f(t)t[j] \mid t \in S^{P_i}, 1 \leq j \leq |t|\}$, then $-e \notin \{f(t)s[j] \mid t \in S^{P_i}, 1 \leq j \leq |t|\}$ and conversely. We call f an *orienting function* of P_i. For example $\{a, b\}$ in Figure 1 is orientable. Note that negating f, one can always obtain another orienting function g of P_i, i.e., $g(t) = -f(t)$.

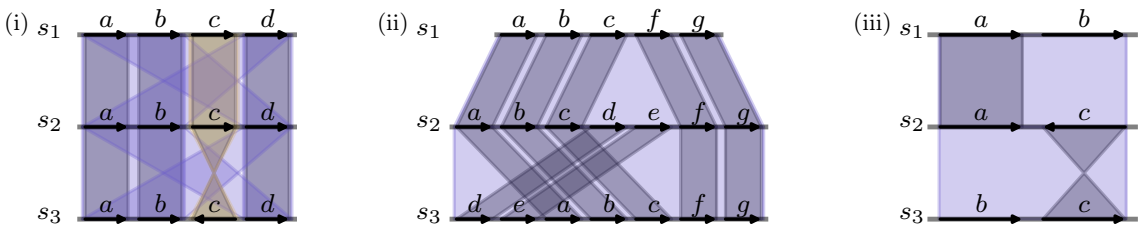

Fig. 2. Critically undesirable properties in synteny blocks. (i) The partition $\{\{a, b, d\}, \{c\}\}$ contains the part $\{a, b, d\}$ which is not contiguous. (ii) This block contains a transposition of elements d and e between s_2 and s_3. (iii) The block $\{a, b, c\}$ is not orientable. Trying to give this block an orientation leads to a contradiction: If it is forward in s_1, then it is forward in s_2 because of a, but also forward in s_3 because of b. However, since it is forward in s_2, it must also be backward in s_3 because of c, a contradiction.

We now formally define synteny blocks and the problem(s) of finding optimal sets of synteny blocks.

Definition 1 (Synteny block). *Let S be a set of strings over the set of elements E and $P = \{P_1, \ldots, P_k\}$ a partition of E. Part P_i is a* synteny block *if all of the following hold: (i) for all $s \in S$, P_i is contiguous in s, (ii) P_i is breakpoint free on S, and (iii) P_i is orientable on S. If each $P_i \in P$ is a synteny block, we call P a* synteny block partition *for S.*

For example, the partition illustrated on the right side of Fig. 1 is a synteny block partition.

As noted above, the phrases induced by blocks of P in each $s \in S$ define a unique parsing which we denote by $\rho_P(s)$. We denote by $\rho_P(S)$ the parsings of all strings in S induced by P. The parsing of S induced by P naturally defines an encoding of the strings in S as strings over the alphabet $\{1, \ldots, k\}$ (where k is the number of blocks of P) where, for every string $s \in S$, its encoding $\mathrm{enc}_P(s)$ is obtained from s by replacing, for every part P_i, the phrase induced by P_i (if P_i is present in s) with $+i$ (resp. $-i$) if P_i is forward (resp.

reversed) in s according to an orienting function f_{P_i}; i is then called the *block identifier*. For example, the encoding of the three strings in Fig. 1 is $\mathrm{enc}_P(s_1) = 4\ 1\ 2\ 3$, $\mathrm{enc}_P(s_2) = 4\ 3\ 2\ 1$ and $\mathrm{enc}_P(s_3) = 4\ -2\ 1$.

Note that the finest partition, i.e., the one where each part contains exactly only one element, is trivially a synteny block partition. However, this is also the largest partition, both in number of parts as well as regarding the resulting encoding. Since downstream rearrangement analyses may be computationally demanding (for example, many rearrangement problems are NP-hard [31]), it makes sense to search for a smaller partition. Moreover, conceptually, a representation that requires fewer blocks is also a more parsimonious explanation for the observed rearrangements than one that requires more blocks. In each case, we are looking for the "smallest" partition. We consider two different criteria for what constitutes a partition to be the "smallest".

Problem 1 (Minimum-Length Synteny Blocks Problem (MLSBP)). Given a collection of strings $S = \{s_1, \ldots, s_m\}$ over an element set E, find a synteny block partition P for S such that the total encoded length $\sum_{i=1}^{m} |\mathrm{enc}_P(s_i)|$ is minimized.

Problem 2 (Minimum-Size Synteny Blocks Problem (MSSBP)). Given a collection of strings $S = \{s_1, \ldots, s_m\}$ over an element set E, find a synteny block partition P for S such that the number of parts $|P|$ is minimized.

Problems 1 and 2 are already meaningful in their original form, but in practice it is often desirable for the resulting synteny block partition to satisfy additional properties. We discuss these next.

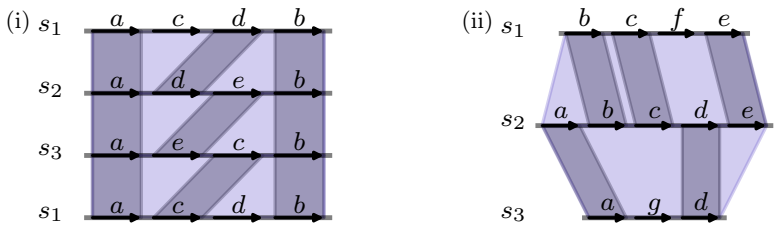

Fig. 3. Potentially undesirable properties of synteny blocks. (i) Despite being breakpoint free, this block does not have a (partial) ordering between elements c, d and e; it is thus not collinear. (ii) The phrases induced by this block in s_1 and s_3 do not share any elements.

Desirable Properties of Synteny Blocks. While enforcing breakpoint-freeness and orientability is sufficient to ensure that pairwise rearrangements are not hidden within synteny blocks, it is not sufficient to do so in a global comparison on strings with unequal content. Consider Fig. 3 (i): While none of the three strings s_1, s_2, s_3 has a breakpoint with respect to any other one, there is clearly disagreement about what the order of $+c$, $+d$ and $+e$ should be, indicating that there must have been a rearrangement at some point during the emergence of these "chromosomes". Additionally, there is no multiple alignment of the phrases induced by this synteny block that respects all element matches. Below, we introduce the concept of *collinearity*, which ensures that such a situation does not occur by enforcing a partial ordering consistent with every phrase.

Definition 2 (Collinear). *Let S be a set of strings. A synteny block P_i with orienting function f is* collinear *in S if there exists a partial order $\prec$ on the signed elements in P_i such that, for every $s \in S$:*

$$\textit{for all } j, 1 \leq j < \left|s^{P_i}\right|, \textit{ if } f(s^{P_i}) = + \textit{ then } s^{P_i}[j] \prec s^{P_i}[j+1], \textit{ otherwise } s^{P_i}[j+1] \prec s^{P_i}[j].$$

Collinearity implies breakpoint-freeness, as stated in the following lemma, proved in Appendix A.

Lemma 1. *Let S be a set of strings over element set E, and P_i a contiguous, orientable part of a partition on E. If P_i is collinear in S, then P_i is breakpoint-free on S.*

Additionally, one may wish to avoid cases where completely unrelated phrases are induced by the same block, such as in Fig. 3 (ii). This can create false signals of homology or rearrangement. Other problems, such as inconsistent orientation (Fig. 2 (iii)) can also arise when phrases induced by a block share no common elements. To prevent this, we require an *anchor* element.

Definition 3 (Anchored). *Let S be a set of strings and $P = \{P_1, \ldots, P_k\}$ be a set of synteny blocks for S. Recall that $S^{P_i} = \{s^{P_i} \mid P_i$ is present in $s \in S\}$. We say that P_i is* anchored *if $\bigcap_{t \in S^{P_i}} \mathrm{elm}(t)$ is not empty. We call any element in $\bigcap_{t \in S^{P_i}} \mathrm{elm}(t)$ an* anchor *of P_i. We say that P is anchored if all synteny blocks P_i are anchored.*

For readers familiar with the concept of "core elements" in pangenomics, we clarify that the concept of an anchor is distinct from the concept of a core element. An anchor does not need to appear in each input string, but only in those input strings where the respective block is present. In principle, even an element that occurs only in a single genome of a larger set of genomes, may be an anchor, whereas such an element is typically not considered part of the core.

Requiring anchored synteny blocks has additional benefits, some of which will be discussed later. Importantly for subsequent analyses, it enables a bijection between breakpoints in the input strings and in the encoded strings which implies preservation of genome rearrangement distances.

Theorem 1. *Let P be an anchored synteny block partition for a set S of singular strings. For each pair of strings $s, t \in S$ there is a bijection $\beta_{s,t}$ between extremities in $\mathrm{Bp}(s,t)$ and in $\mathrm{Bp}(\mathrm{enc}_P(s), \mathrm{enc}_P(t))$, such that*

- *for each breakpoint $\{a, b\} \in \mathrm{Bp}(\mathrm{enc}_P(s), \mathrm{enc}_P(t))$, $\{\beta_{s,t}(a), \beta_{s,t}(b)\}$ is a breakpoint of s, t,*
- *for each breakpoint $\{c, d\} \in \mathrm{Bp}(s,t)$, $\{\beta^{-1}_{s,t}(c), \beta^{-1}_{s,t}(d)\}$ is a breakpoint of $\mathrm{enc}_P(s), \mathrm{enc}_P(t)$.*

We prove this theorem in Appendix B. The bijection between breakpoints is a powerful property, as many rearrangement models – such as the Reversal [32], Hannenhalli-Pevzner [33], Single-Cut-or-Join [34] and Double-Cut-and-Join [35] models – consider only breakpoints, not conserved adjacencies. This means that distances under such models are preserved for an anchored synteny block partition.

Finally, we propose the following variants of Problems 1 and 2, restricted to synteny blocks with these desirable properties, which play a central role in our results.

Problem 3 (Collinear anchored-MSSBP). Given a collection of strings $S = \{s_1, \ldots, s_m\}$ over an element set E, find a synteny block partition P for S in which all blocks are collinear and anchored, and which is optimal with respect to the partition size, as defined in Problem 1.

Problem 4 (Collinear anchored-MLSBP). Given a collection of strings $S = \{s_1, \ldots, s_m\}$ over an element set E, find a synteny block partition P for S in which all blocks are collinear and anchored, and which is optimal with respect to the encoded length, as defined in Problem 2.

3 Methods

With these definitions in place, we now consider the computational complexity of the problem and propose an optimal algorithm for the restricted version. First, we prove that both Problem 1 (MLSBP) and Problem 2 (MSSBP) are NP-hard in general. Then we consider Problems 3 and 4 where blocks must be collinear and anchored. Under these constraints, both problems have the same optimal solution, and this solution can be computed in linear time by a simple greedy merging procedure.

3.1 Finding a Smallest Synteny Block Partition is NP-hard

The optimization Problems 1 and 2 admit no polynomial time solution unless P = NP. We establish this via two polynomial-time reductions: MLSBP is shown NP-hard by a reduction from Vertex Cover, and MSSBP is shown NP-hard by a reduction from SAT. The detailed proofs appear as Theorems 5 and 6 in Appendices C and D, respectively, from which the following result follows.

Theorem 2. *Problems 1 and 2 are NP-hard.*

3.2 Finding a Smallest Collinear, Anchored Synteny Block Partition is Polynomial

Although the unrestricted problems are hard, when adding the constraints that blocks should be collinear and have (at least) one anchor, the problem is much easier to solve. In fact, as we show in this section, the solution for both problems is the same.

We first formalize the notion that when every occurrence of an element forms an adjacency with the same element, they should belong to the same synteny block.

Definition 4 (Unique neighbor). *We say that extremity a^x has a* unique neighbor *extremity b^y on string set S, for some elements a, b and some $x, y \in \{h, t\}$, if $\{a^x, b^y\}$ is the only adjacency involving a^x on S. We write $a^x \to b^y$. When the concrete extremities x, y are irrelevant, we simply write $a \to b$.*

For two parts P_i, P_j of partition P, we write $P_i \to P_j$ if in any encoding $\mathrm{enc}_P(S)$, $i \to j$.

Note that the unique neighbor relationship is directed, i.e., for $a \to b$, whenever a occurs, b occurs adjacent to it, but the converse is not necessarily true. This is in contrast to the concept of compaction employed in compacted de Bruijn graphs, where two k-mers a, b are placed in the same unitig if a always occurs adjacent to b, but also b always occurs adjacent to a. In our notation, this would be $a \to b$ *and* $b \to a$. We also note that if $i^x \to j^y$ for some $x, y \in \{h, t\}$ in an encoding of a partition P with $P_i, P_j \in P$, we can always choose the orienting functions of P_i, P_j, such that $i^h \to j^t$.

We now show that one can create a set of collinear anchored synteny blocks by iteratively merging smaller (also collinear anchored) synteny blocks, where one of them is a unique neighbor of the other.

Lemma 2. *Let P_i, P_j be collinear synteny blocks of partition P with anchors c_i, c_j, such that $P_i \to P_j$. Then $P_i \cup P_j$ is a collinear synteny block with anchor c_j.*

Proof. In the encoding $\mathrm{enc}_P(s)$, P_i, P_j are represented by i, j. W.l.o.g. let $i^h \to j^t$. Given that P_i, P_j are synteny blocks, the only strings relevant to whether $P_i \cup P_j$ is a synteny block are those where both P_i and P_j are present. Since $i^h \to j^t$, in all strings where P_i and P_j are present, they occur in the same orientation and their induced phrases are adjacent. Thus $P_i \cup P_j$ is a synteny block and concatenating the partial orders of P_i and P_j, such that $e \prec d$ for all $e \in P_i, d \in P_j$ is a partial order that satisfies Definition 2 for each phrase. Moreover, in each string where P_i is present, P_j is also present, therefore anchor c_j is also present. Thus, $P_i \cup P_j$ has anchor c_j. □

This strategy can then be used to construct any collinear anchored synteny block partition.

Theorem 3. *Any collinear anchored synteny block partition $P = \{P_1, \ldots, P_k\}$ on element set $E = \{e_1, \ldots e_l\}$ can be constructed by iteratively merging pairs of parts $Q \cup R$, where $Q \to R$, starting from the finest partition $\{\{e_1\}, \ldots, \{e_l\}\}$.*

We provide here a high-level sketch of a constructive proof, with full details in Appendix E. Recall that an anchored collinear block must have an anchor c and a partial order $\prec$ which describes the order of elements in each phrase. Starting from the anchor c, regard the closest element e to c in the partial order, such that no other element d exists with $c \prec d \prec e$. Then in each string where e occurs, c must also occur (since c is an anchor) and moreover, because there can't be any element d between c and e, c is the unique neighbor of e. This argument can be iterated using a topological sorting of the block elements. One can thus construct any collinear anchored synteny block by merging elements with their unique neighbors.

To characterize optimal sets of synteny blocks, it is helpful to keep track of only one anchor in case there are multiple anchors per block.

Definition 5 (Canonical anchor). *For a collinear anchored partition $P = \{P_1, \ldots, P_k\}$ on strings S on element set E, we call the smallest anchor $c_i \in P_i$ according to an (arbitrary) total order on E the* canonical anchor *of P_i.*

The canonical anchors identify the parts that they belong to, since each phrase must contain them. It is therefore possible to find an encoding of the input string set by simply removing all elements that are not canonical anchors.

Observation 1 *Let $P = \{P_1, \dots, P_k\}$ be a collinear anchored synteny block partition of element set E with canonical anchors $c_1, \dots c_k$ on string set S. Then the string set S' where each element but $c_1, \dots, c_k$ is removed and then each $+c_i$ $(-c_i)$ is replaced by $+i$ $(-i)$ is an encoding for P.*

We can then construct a partition P where no further parts can be merged by iteratively unifying synteny blocks. The corresponding encoding S^* can then be created by removing elements that are not canonical anchors. To that end, we denote by $S \setminus e$ for a string set S the operation that removes each occurrence of element e in S. Pseudocode is given in Algorithm 1 and a visual representation in Appendix G, Fig. 8.

Algorithm 1 Generate a partition P of collinear, anchored synteny blocks and its encoding S^*.

Require: A set of singular signed strings S on element set $E = \{e_1, \dots, e_l\}$
  $P \leftarrow \{\{e_1\}, \dots, \{e_l\}\}$
  while $P_i \to P_j$ for some $P_i, P_j \in P$ with canonical anchors c_i, c_j :
    $P \leftarrow P \setminus \{P_i, P_j\} \cup \{P_i \cup P_j\}$
    $r \leftarrow c_i$
    if $P_j \to P_i$: ▷ If they are each other's unique neighbor, remove the larger anchor
      $r \leftarrow \max(c_i, c_j)$
    $S \leftarrow S \setminus r$
  Obtain S^* by replacing each canonical anchor c_i with i, for each remaining part P_i
  return P, S^*

The way elements are merged here is similar to other compaction algorithms, such as the compaction step in maf2synteny [17]. However, while these algorithms are heuristics, Algorithm 1 is optimal for collinear, anchored MLSBP and MSSBP, as we show in the following. We begin by showing that, in the first iteration of the while loop, the removed element r from S cannot be a canonical anchor.

Lemma 3. *Consider two elements e, d with $e \to d$. If $d \not\to e$ or $d < e$ and e is a canonical anchor in an anchored, collinear partition $P = \{P_1, \dots, P_k\}$, then there is a shorter partition P' with a shorter encoding than P, such that P' is also anchored and collinear.*

Proof. (By contradiction) Assume that e is a canonical anchor of part P_i. Note that because $e \to d$ and $d < e$ or $d \not\to e$, e cannot be the canonical anchor of a block which contains d, i.e. $d \notin P_i$. Let $d \in P_j$ with canonical anchor c. Then $P_i \to P_j$ (because $e \to d$), that is $P_i \cup P_j$ is a collinear synteny block and has anchor c according to Lemma 2. Observe that $P' = P \setminus \{P_i, P_j\} \cup \{P_i \cup P_j\}$ has both fewer parts and a shorter encoding than P. □

Note that removing an element also does not change whether the phrases induced by a block are contiguous or collinear, and as long as one anchor remains, the block also remains anchored. Thus, any optimal solution P on S still has an equivalent solution on $S \setminus r$.

Observation 2 *Given a set of strings $S = \{s_1, \dots, s_m\}$ and a collinear anchored partition $P = \{P_1, \dots, P_k\}$, let $P \setminus e$ be the set of parts and $S \setminus e$ be the set of strings obtained by removing an element e that is not a canonical anchor. Then $P \setminus e$ is a collinear anchored synteny block partition on $S \setminus e$.*

Let S_q be S in the q-th iteration of Algorithm 1. It follows from Observation 2 and Lemma 3 that any optimal solution on S_q still has a (possibly suboptimal) equivalent solution on S_{q+1} and the element r removed in S_{q+1} never is a canonical anchor of an optimal solution on S_q. Then any optimal solution on S^* is at most as large as any optimal solution on S.

Corollary 1. *Let S^* be the string set produced by Algorithm 1.*

Let P^*_{Size}, P_{Size} be the collinear, anchored partitions on S^* and on S respectively, such that $|P^*_{Size}|$ and $|P_{Size}|$ are minimized. Then $|P^*_{Size}| \leq |P_{Size}|$.

Let P^*_{Length}, P_{Length} be collinear, anchored partitions on S^* and on S respectively, such that the respective encoding lengths $p^* := \sum_{s^* \in S^*} |\operatorname{enc}_{P^*_{Length}}(s^*)|$ and $p := \sum_{s \in S} |\operatorname{enc}_{P_{Length}}(s)|$ are minimized. Then $p^* \leq p$.

Note that on S^* there are no further elements with unique neighbors. On such a string set, Theorem 3 implies that the only collinear anchored synteny block partition on S^* is the finest partition. Since $|P|$ is the number of elements in S^* and $\sum_{s \in S} |\operatorname{enc}_P(s)| = \sum_{s^* \in S^*} |s^*|$, the partition constructed by the algorithm is optimal for both MLSBP and MSSBP.

Theorem 4. *Algorithm 1 constructs a partition P that is a solution to Problems 3 and 4.*

This shows that in the restricted case Problems 3 and 4 are in fact equivalent. However, given the differences in the hardness proofs, we conjecture that in general Problems 1 and 2 are distinct.

With the correctness of the greedy algorithm established, we analyze its computational complexity. Let L denote the total number of extremity occurrences across all genomes, counted with multiplicity.

Lemma 4. *Algorithm 1 admits an implementation which is linear in time and space with respect to the input genome size L.*

A detailed proof is provided in the Appendix F; in the following we give a brief summary. Algorithm 1 repeatedly merges elements that have a unique neighbor until no further merges are possible. Initially, all extremities with a unique neighbor are added to a queue. While the queue is not empty, the algorithm removes one extremity a^x, determines its unique neighbor b^y (with $a \to b$), and merges the corresponding element a into b. All extremities c^z that were adjacent to the opposite extremity of a^x are reconnected to b^y. If this update causes any of these neighboring extremities to have a unique neighbor, they are added to the queue. The algorithm terminates when the queue is empty. The complexity analysis has three main components: the number of iterations of the main while loop, the cost of updating adjacencies following a merge, and the cost of detecting new unique neighbors. We show that the while loop considers each adjacency at most twice, each corresponding to a possible merge between two elements. Merging an element a into b affects only the occurrences of a, and since merged elements are never reinserted, the total update cost remains linear in L. Finally, checking for new unique neighbors can be done in amortized linear time with respect to L, as each adjacency list is scanned only until one or two non-merged elements are found.

Extension to Duplicates. We propose two extensions to handle duplicates. First, we introduce an approach that marks duplicated elements and prevents them from merging with any other element. In this way, breakpoints are fully preserved (by Theorem 1). We refer to this variant as the "breakpoint (BP) bijection" mode. Second, to allow for smaller block representations, we permit merging with duplicated elements. The theoretical implications are explored in Appendix G. However, it is not possible to define breakpoints between duplicate elements without disambiguating copies of duplicates, which in turn depends on which rearrangement model is used and is typically NP-hard [36, 37]. We therefore require in the duplicate theory presented in Appendix G that phrases induced by blocks contain each element at most once, i.e., each phrase is a singular string of elements. This allows to define a block as breakpoint-free if none of its phrases have breakpoints with respect to each other. However, this does not guarantee that global breakpoints are preserved, because Theorem 1 cannot be defined for duplicates without a concrete rearrangement model. And even singular elements with a breakpoint between them can appear in the same block if they are merged with duplicates as we show in Appendix Fig. 7. Nonetheless, this theory guarantees that the phrases induced by a block are breakpoint free with respect to each other, which implies in particular that any non-duplicate block does not obscure any rearrangements. We refer to this mode in the following as the "duplicates" mode.

These two approaches mirror the concepts of *global collinearity* and *local collinearity*, found in the existing literature on synteny blocks [20]; the "breakpoint (BP) bijection" mode preserves breakpoints globally, while the "duplicates" mode preserves breakpoints locally.

4 Results

We implemented the linear version of Algorithm 1 and made it available at `github.com/gi-bielefeld/mice`. We refer to this implementation as *Markers Inferred by Compacting Elements (MICE)*. We tested three distinct modes of the algorithm. Since the theory presented here is based on singular elements, the default mode ("MICE") ignores duplicates as elements, but may still create blocks that span positions with duplicate elements. As described in the previous section, we offer two modes with different guarantees for duplicates.

"MICE duplicates" filters out elements which occur more than 10 times per genome while allowing merges with duplicates. "MICE BP bijection" on the other hand does not permit merging with or over any duplicate element.

For our proof-of-concept experiments, we conceptually use 31-mers as elements, since k-mers are a well-known annotation-free and frequently used type of object in genomics and $k = 31$ is among the most frequent settings of this parameter. Since our compaction is a generalized version of unitigs, we used unitigs directly as input, not individual k-mers. We generated unitigs using ggcat [38]. This value for k is slightly higher than for SibeliaZ, which we describe next. However, for MICE the value of k primarily defines the threshold above which rearrangements are considered, not sensitivity for similarity. We benchmarked MICE against state-of-the-art synteny block detection methods, described in the following.

SibeliaZ [20] is a heuristic algorithm that also works on elements (usually k-mers), selecting a block if there is a strong enough "backbone" of k-mers. It is currently one of the fastest tools to generate synteny blocks. In each case, we used the recommended k-value ($k = 15$ for bacteria, $k = 25$ otherwise) and two different k-mer frequency thresholds a, namely $a_{\text{low}} = 2 \times m$ and $a_{\text{high}} = 2 \times 10 \times m$. According to its documentation, SibeliaZ with a_{low} should ignore duplicates, thus behaving like the default mode of MICE. SibeliaZ with a_{high} should be able to recover duplicates that occur up to 10 times and therefore behave more similarly to MICE duplicates or MICE BP bijection.

Minigraph-Cactus [39] is a workflow that builds on minigraph [40] using the Cactus framework [41, 42], which is an alignment-based, optimization-free way to construct pangenome graphs and synteny blocks.

Datasets. We used five pangenomic datasets to assess the performance of our method, namely a selection of 56 *Y. pestis* assemblies, 48 *E. coli* assemblies, all 29 currently available *S. cerevisiae* assemblies with NCBI assembly level "complete", all 5 currently available *A. thaliana* assemblies with NCBI assembly level "complete" as well as the 16 *M. musculus* genomes studied in the initial publication of SibeliaZ [20]. We give details to these datasets in Appendix H, Table 1.

4.1 Benchmark

Runtime. We benchmarked MICE against SibeliaZ. For both methods we only evaluated the synteny block generation step, that is, we measured the time it takes to convert elements as input to blocks as output. To that end, we provided MICE with a GFF-format file representing unitigs in the ggcat graph and we provided SibeliaZ with the binary TwoPaCo graph and skipped SibeliaZ's alignment step. We did not include Minigraph-Cactus in this evaluation as the costly alignment step is central to the Minigraph-Cactus workflow. The runtimes for Minigraph-Cactus in each case were slower (see also Fig. 9 in Appendix H), but are not directly comparable to the measurements reported here.

We ran all algorithms on a single thread on a cluster machine. The results are visualized in Fig. 4. We see that despite being an exact algorithm with strict theoretical guarantees, MICE achieves similar performance to SibeliaZ, even outperforming the version with a high abundance filter. However, for both tools we observed that a significant portion of the runtime was spent on loading the input data from disk. We therefore see major potential for performance improvement by integrating ggcat directly with MICE. Because not all SibeliaZ runs finished within 24 hours on the *M. musculus* dataset, in the following experiments we only used the first 4 of the 16 assemblies of this dataset.

Block contiguity. We proceeded to evaluate the coverage and contiguity of SibeliaZ, Minigraph-Cactus and MICE on the benchmark datasets. By default, SibeliaZ restricts its output to blocks of 50 basepairs or longer. To ensure a fair comparison, we therefore included only blocks of at least 50 basepairs for all tools in this analysis. We examined how many blocks it takes to cover a certain fraction of the input genomes. As a summary, we give the coverage and encoding sizes in Table 2 in Appendix H and the respective N50, N75 and N90 values in Fig. 5. Here Nx is defined as the length threshold with which one can cover x% of the total genome lengths using only sequences of length Nx or higher. We show details in Fig. 10 of Appendix H. We see that for all datasets, the default version of MICE covers the same or more positions with fewer and larger blocks than SibeliaZ and Minigraph-Cactus. Running in duplicate and BP bijection mode produces slightly shorter blocks on low duplicate datasets, but has problems covering genomes with large numbers of duplicates.

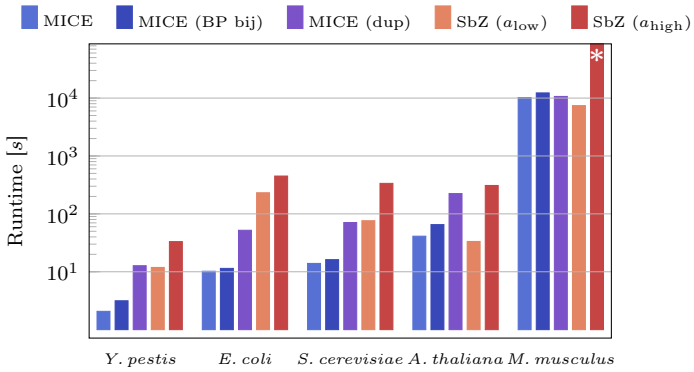

Fig. 4. Runtimes of three MICE variants (MICE, breakpoint-bijection mode "MICE (BP bij)", and duplicate mode "MICE (dup)") and two SibeliaZ variants (SbZ with a_{low} and a_{high}) on five benchmark datasets, measured with `/usr/bin/time` wall-clock time. The asterisk in SbZ (a_{high}) for *M. musculus* indicates that it exceeded the 24-hour limit.

Nonetheless, the coverage improvement of the default mode in these genomes is even more noticeable. Additionally, examining the proportion of shared blocks between pairs of genomes (Appendix H, Fig. 11) shows that the high coverage of MICE does not come at the expense of significantly reducing sensitivity in finding shared regions.

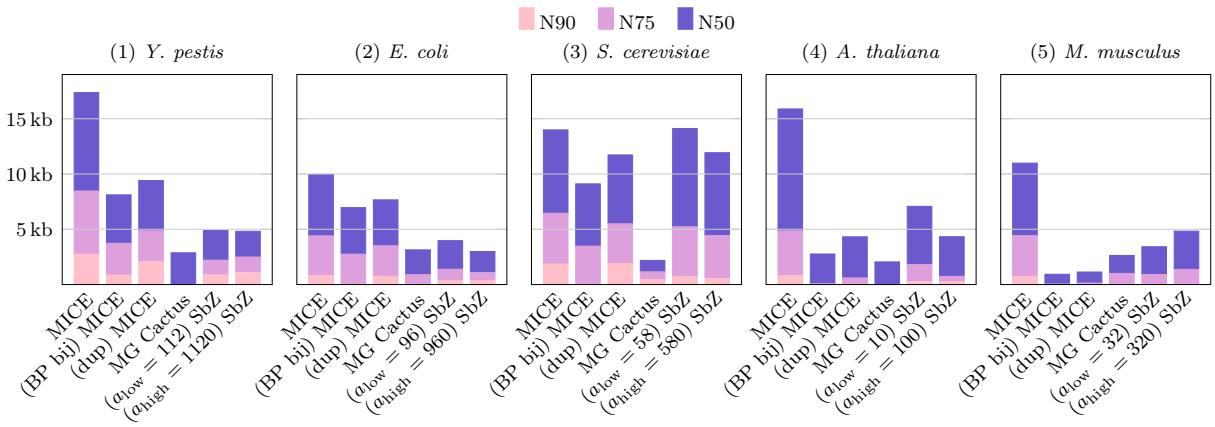

Fig. 5. N50, N75 and N90 values for blocks generated by the tools (on the x-axis): MICE, MICE in breakpoint bijection mode ("(BP bij) MICE"), MICE with duplicates ("(dup) MICE"), Minigraph-Cactus (MG Cactus) and SibeliaZ (SbZ), evaluated on five benchmark datasets.

4.2 Rearrangement Detection

While it is desirable for synteny blocks to be large and have high coverage, they must also avoid obscuring true rearrangements or create new ones. To assess this, we measured precision and recall for each method and for four of the five datasets. We excluded the *M. musculus* dataset because determining breakpoints on the original elements in this dataset was too computationally expensive.

Breakpoints involving duplicated elements are not well-defined without assuming a specific rearrangement and matching model. Therefore, for this analysis, we discarded any element that appeared as a duplicate in at least one genome of the dataset. For each dataset, we removed duplicated elements, compared all pairs of

genome sequences, projected their sequences onto their shared 31-mers, and computed the set of breakpoints. We refer to this set as the element breakpoints.

We define a breakpoint as unrecoverable if an element breakpoint falls entirely within a block, making it undetectable without inspecting the elements inside the block. For each block produced by a tool, we identified the elements whose coordinates lie within the block's coordinate range. For every such block, we examined all pairs of contained elements (a, b). If (a, b) constitutes an element breakpoint but the block merges them, this breakpoint is counted as a false negative. In Appendix I, we compare this definition to a more relaxed variant in which all elements assigned the same block identifier are unified; the resulting numbers of unrecoverable breakpoints are very similar.

The set of true positives consists of all element breakpoints that are not obscured by any block. Conversely, false positives are defined as breakpoints between two blocks x and y that do not correspond to a breakpoint in the element set. To identify these, we again compare each pair of genomes and project their sequences onto the shared blocks. A breakpoint between x and y is counted as a false positive if there is no supporting element breakpoint between any element in block x and any element in block y.

Using these definitions, recall is computed as the fraction of element breakpoints that remain detectable after block construction, i.e., the proportion of true positives among all element breakpoints. Precision is computed as the fraction of true positives breakpoints among all the one recalled, i.e., the proportion of true positives among all predicted breakpoints. Figure 6 reports precision and recall for all evaluated methods.

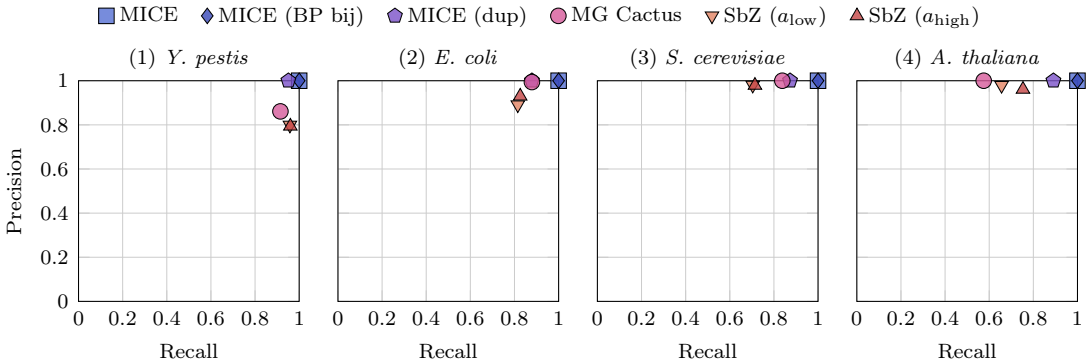

Fig. 6. Precision and recall for all methods across four datasets. A false negative is a pair of elements that forms a breakpoint in a pair of genomes but is placed within the same block, thereby obscuring the breakpoint. True positives are element breakpoints that remain detectable after block construction. False positives are breakpoints implied by block pairs that have no supporting element breakpoint between the corresponding elements.

By construction, MICE and MICE (BP bij) achieve 100% precision and 100% recall (Theorem 1). MICE (dup) also attains 100% precision by construction. Its recall is the highest among the remaining methods on all datasets except *Y. pestis*, where it is comparable to the other tools, though still below the recall of MICE and MICE (BP bij). MG Cactus achieves the highest precision among the non-MICE methods, reaching 100% on *S. cerevisiae* and *A. thaliana*. Its recall ranges between 84% and 92% on all datasets except *A. thaliana*, where it decreases to 58%. SibeliaZ (a_{low}) and SibeliaZ (a_{high}) show similar performance. Both achieve precision between 80% and 98% and recall between 70% and 96%. As with MG Cactus, recall drops on *A. thaliana*, where it decreases to 66%. Overall, SibeliaZ (a_{low}) has marginally higher precision and marginally lower recall than SibeliaZ (a_{high}).

Additionally, to illustrate how breakpoints can become unrecoverable, we selected three *E. coli* genomes for a small-scale visualization. We applied all five methods and chose a representative locus to compare their behavior. This analysis is presented in Appendix J.

5 Discussion

We introduced a formal framework for deriving synteny blocks by compacting genomic elements in a way that preserves rearrangements. To our knowledge, this is the first such framework. We defined the MSSBP and MLSBP problems and showed that both are NP-hard in general. Yet, when blocks are required to be collinear and contain an anchor element, both objectives can be optimized simultaneously by a linear-time algorithm. Our experiments indicate that, using 31-mers as elements, this algorithm is competitive with state-of-the-art synteny block construction methods while preserving all rearrangements of unique elements present in the input data. Nonetheless, which elements to use as input for different research objectives, is an open question.

We did not assess how well the sequences within the generated blocks align. Methods that explicitly model alignment, such as Minigraph-Cactus, likely produce blocks that are better suited for whole-genome alignment. Nonetheless, future work may combine rearrangement-based and alignment-based synteny block theory. In practice, it may be possible to use the larger blocks generated by MICE as a pre-segmentation step before applying slower alignment-based methods or even seeding aligners with anchors computed by MICE. As of now, both of these possibilities depend on further theoretical work as neither anchors nor partitions generated by MICE are necessarily unique. Additionally, while our results on breakpoints focus on representing rearrangements, we did not evaluate duplication breakpoints due to the challenges of defining this kind of breakpoint (see "Extension to Duplicates" in Section 3.2). This limitation may affect all tools, except for "MICE BP bijection" where we preserve breakpoints by avoiding merging with duplicated elements.

Our proposed methods for duplicated elements come with a tradeoff of either more fragmented blocks ("MICE BP bijection") or losing some theoretical guarantees ("MICE duplicates"). This suggests that a different generalization may yield a more effective handling of duplicates while ensuring stricter guarantees regarding breakpoints. For example, "MICE duplicates" so far guarantees only what is known in the literature as *local collinearity* (collinearity per phrase induced by a block), not *global collinearity* (collinearity w.r.t. the entire genome).

Another research direction is to explore relaxed variants of the theoretical properties presented here. For example, one could require that each pair of genomes shares at least one element in a block, without requiring it to be the same element across all pairs. However, this may come at the risk of a computationally harder problem or too dissimilar blocks. Nonetheless, we observe that even when the optimization problem of finding the smallest synteny block partition may be NP-hard, it could still be practically valuable to find a synteny block partition that retains all biologically motivated constraints (such as breakpoint-freeness) while not guaranteeing to be the smallest. Heuristics in this direction may thus be one way to tackle the NP-hard problems we discussed here. Finally, one could consider the frequency of breakpoints and allow merges across breakpoints that occur only rarely, which may enable stronger compression in closely related genomes, such as pangenome datasets.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement number 872539 (PANGAIA). Most of the work was conceived and developed during a research stay of LB and LP at Simon Fraser University. We thank Kamil Hepak for helpful comments on readability, visualization and language.

References

[1] P. A. Pevzner and G. Tesler. "Genome rearrangements in mammalian evolution: lessons from human and mouse genomes". In: *Genome Research* 13.1 (2002), pp. 37–45. DOI: 10.1101/gr.757503.

[2] G. Bourque, P. A. Pevzner, and G. Tesler. "Reconstructing the genomic architecture of ancestral mammals: lessons from human, mouse, and rat genomes". In: *Genome Research* 14.4 (2004), pp. 507–516. DOI: 10.1101/gr.1975204.

[3] J. Ma et al. "Reconstructing contiguous regions of an ancestral genome". In: *Genome Research* 16.12 (2006), pp. 1557–1565. DOI: 10.1101/gr.5383506.

[4] C. Chauve and É. Tannier. "A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genomes". In: *PLoS Computational Biology* 4.11 (2008), e1000234. DOI: 10.1371/journal.pcbi.1000234.

[5] Y. Wang et al. "Mcscanx: a toolkit for detection and evolutionary analysis of gene synteny and collinearity". In: *Nucleic Acids Research* 40 (7 2012), e49–e49. DOI: 10.1093/nar/gkr1293.

[6] A. Sinha and J. Meller. "Cinteny: flexible analysis and visualization of synteny and genome rearrangements in multiple organisms". In: *BMC Bioinformatics* 8.1 (2007). DOI: 10.1186/1471-2105-8-82.

[7] B. Paten et al. "Enredo and pecan: genome-wide mammalian consistency-based multiple alignment with paralogs". In: *Genome Research* 18.11 (2008), pp. 1814–1828. DOI: 10.1101/gr.076554.108.

[8] S. Pham and P. A. Pevzner. "Drimm-synteny: decomposing genomes into evolutionary conserved segments". In: *Bioinformatics* 26.20 (2010), pp. 2509–2516. DOI: 10.1093/bioinformatics/btq465.

[9] M. Grabherr et al. "Genome-wide synteny through highly sensitive sequence alignment: satsuma". In: *Bioinformatics* 26 (9 2010), pp. 1145–1151. DOI: 10.1093/bioinformatics/btq102.

[10] S. Proost et al. "I-adhore 3.0—fast and sensitive detection of genomic homology in extremely large data sets". In: *Nucleic Acids Research* 40.2 (2011), e11–e11. DOI: 10.1093/nar/gkr955.

[11] B. Paten et al. "Cactus: algorithms for genome multiple sequence alignment". In: *Genome Research* 21 (9 2011), pp. 1512–1528. DOI: 10.1101/gr.123356.111.

[12] B. Brejová, M. Burger, and T. Vinař. "Automated Segmentation of DNA Sequences with Complex Evolutionary Histories". In: *Algorithms in Bioinformatics - 11th International Workshop, WABI 2011, Saarbrücken, Germany, September 5-7, 2011. Proceedings.* Vol. 6833. Lecture Notes in Computer Science. 2011, pp. 1–13. DOI: 10.1007/978-3-642-23038-7_1.

[13] I. Minkin et al. "Sibelia: a scalable and comprehensive synteny block generation tool for closely related microbial genomes". In: *Algorithms in Bioinformatics - 13th International Workshop, WABI 2013, Sophia Antipolis, France, September 2-4, 2013. Proceedings.* Vol. 8126. Lecture Notes in Computer Science. 2013, pp. 215–229. DOI: 10.1007/978-3-642-40453-5_17.

[14] G. Drillon, A. Carbone, and G. Fischer. "Synchro: a fast and easy tool to reconstruct and visualize synteny blocks along eukaryotic chromosomes". In: *PLoS ONE* 9.3 (2014), e92621. DOI: 10.1371/journal.pone.0092621.

[15] J. Lee et al. "Synteny portal: a web-based application portal for synteny block analysis". In: *Nucleic Acids Research* 44 (W1 2016), W35–W40. DOI: 10.1093/nar/gkw310.

[16] J. M. Lucas and H. Roest Crollius. "High precision detection of conserved segments from synteny blocks". In: *Plos One* 12.7 (2017), e0180198. DOI: 10.1371/journal.pone.0180198.

[17] M. Kolmogorov et al. "Chromosome assembly of large and complex genomes using multiple references". In: *Genome Research* 28.11 (2018), pp. 1720–1732. DOI: 10.1101/gr.236273.118.

[18] E. Polevikov and M. Kolmogorov. "Synteny paths for assembly graphs comparison". In: *19th International Workshop on Algorithms in Bioinformatics, WABI 2019, September 8-10, 2019, Niagara Falls, NY, USA.* Vol. 143. LIPIcs. 2019, 24:1–24:14. DOI: 10.4230/lipics.wabi.2019.24.

[19] M. Goel et al. "Syri: finding genomic rearrangements and local sequence differences from whole-genome assemblies". In: *Genome Biology* 20.1 (2019). DOI: 10.1186/s13059-019-1911-0.

[20] I. Minkin and P. Medvedev. "Scalable multiple whole-genome alignment and locally collinear block construction with sibeliaz". In: *Nature Communications* 11.1 (2020). DOI: 10.1038/s41467-020-19777-8.

[21] K. Krasheninnikova et al. "Halsynteny: a fast, easy-to-use conserved synteny block construction method for multiple whole-genome alignments". In: *GigaScience* 9.6 (2020). DOI: 10.1093/gigascience/giaa047.

[22] J. Armstrong et al. "Progressive cactus is a multiple-genome aligner for the thousand-genome era". In: *Nature* 587 (7833 2020), pp. 246–251. DOI: 10.1038/s41586-020-2871-y.

[23] L. Coombe et al. "Multi-genome synteny detection using minimizer graph mappings". In: (2024). DOI: 10.1101/2024.02.07.579356.

[24] P. A. Pevzner and G. Tesler. "Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution". In: *Proceedings of the National Academy of Sciences* 100.13 (2003), pp. 7672–7677. DOI: 10.1073/pnas.1330369100.

[25] D. Sankoff and P. Trinh. "Chromosomal breakpoint reuse in genome sequence rearrangement". In: *Journal of Computational Biology* 12.6 (2005), pp. 812–821. DOI: 10.1089/cmb.2005.12.812.

[26] Q. Peng, P. A. Pevzner, and G. Tesler. "The fragile breakage versus random breakage models of chromosome evolution". In: *PLoS Computational Biology* 2.2 (2006), e14. DOI: 10.1371/journal.pcbi.0020014.

[27] D. Sankoff. "The Signal in the Genomes". In: *PLOS Computational Biology* 2.4 (2006), e35. DOI: 10.1371/journal.pcbi.0020035.

[28] A. Bergeron, J. Mixtacki, and J. Stoye. "On Computing the Breakpoint Reuse Rate in Rearrangement Scenarios". In: *RECOMB International Workshop on Comparative Genomics (RECOMB-CG) 2008*. Vol. 5267. LNBI. Springer, 2008, pp. 226–240. DOI: 10.1007/978-3-540-87989-3_17.

[29] A. Elghraoui et al. "Evaluating impacts of syntenic block detection strategies on rearrangement phylogeny using *mycobacterium tuberculosis* isolates". In: *Bioinformatics* 39.1 (2023). DOI: 10.1093/bioinformatics/btad024.

[30] C. G. Ghiurcuta and B. M. E. Moret. "Evaluating synteny for improved comparative studies". In: *Bioinformatics* 30.12 (2014), pp. i9–i18. DOI: 10.1093/bioinformatics/btu259.

[31] Eric Tannier, Chunfang Zheng, and David Sankoff. "Multichromosomal median and halving problems under different genomic distances". In: *BMC Bioinformatics* 10.1 (2009), p. 120. ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-120. URL: https://doi.org/10.1186/1471-2105-10-120.

[32] S. Hannenhalli and P. A. Pevzner. "Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals". In: *Journal of the ACM* 46.1 (1999), pp. 1–27. DOI: 10.1145/300515.300516.

[33] S. Hannenhalli and P. A. Pevzner. "Transforming Men into Mice (Polynomial Algorithm for Genomic Distance Problem)". In: *36th Annual Symposium on Foundations of Computer Science, FOCS 1995, Milwaukee, Wisconsin, USA, 23-25 October 1995*. IEEE Computer Society, 1995, pp. 581–592. DOI: 10.1109/SFCS.1995.492588.

[34] P. Feijão and J. Meidanis. "SCJ: A Breakpoint-Like Distance that Simplifies Several Rearrangement Problems". In: *IEEE ACM Transactions in Computational Biology and Bioinformatics* 8.5 (2011), pp. 1318–1329. DOI: 10.1109/TCBB.2011.34. URL: https://doi.org/10.1109/TCBB.2011.34.

[35] A. Bergeron, J. Mixtacki, and J. Stoye. "A Unifying View of Genome Rearrangements". In: *Algorithms in Bioinformatics, 6th International Workshop, WABI 2006, Zurich, Switzerland, September 11-13, 2006, Proceedings*. Vol. 4175. Lecture Notes in Computer Science. Springer, 2006, pp. 163–173. DOI: 10.1007/11851561_16.

[36] Mingfu Shao, Yu Lin, and Bernard M.E. Moret. "An Exact Algorithm to Compute the Double-Cut-and-Join Distance for Genomes with Duplicate Genes". In: *Journal of Computational Biology* 22.5 (2015). PMID: 25517208, pp. 425–435. DOI: 10.1089/cmb.2014.0096. eprint: https://doi.org/10.1089/cmb.2014.0096. URL: https://doi.org/10.1089/cmb.2014.0096.

[37] Leonard Bohnenkämper et al. "Computing the Rearrangement Distance of Natural Genomes". In: *Journal of Computational Biology* 28.4 (2021). PMID: 33393848, pp. 410–431. DOI: 10.1089/cmb.2020.0434. eprint: https://doi.org/10.1089/cmb.2020.0434. URL: https://doi.org/10.1089/cmb.2020.0434.

[38] Andrea Cracco and Alexandru I Tomescu. "Extremely fast construction and querying of compacted and colored de Bruijn graphs with GGCAT". In: *Genome Research* 33.7 (2023), pp. 1198–1207.

[39] G. Hickey et al. "Pangenome graph construction from genome alignments with Minigraph-Cactus". In: *Nature Biotechnology* 42.4 (2024), pp. 663–673. ISSN: 1546-1696. DOI: 10.1038/s41587-023-01793-w.

[40] H. Li, X. Feng, and C. Chu. "The design and construction of reference pangenome graphs with minigraph". In: *Genome Biology* 21.1 (2020), p. 265. ISSN: 1474-760X. DOI: 10.1186/s13059-020-02168-z.

[41] B. Paten et al. "Cactus Graphs for Genome Comparisons". In: *Journal of Computational Biology* 18.3 (2011), p. 469. DOI: 10.1089/cmb.2010.0252.

[42] B. Paten et al. "Superbubbles, Ultrabubbles, and Cacti". In: *Journal of Computational Biology* 25.7 (2018), pp. 649–663. DOI: 10.1089/cmb.2017.0251.