UNIVERSITY OF CALIFORNIA SAN DIEGO

Human centromeres: from initial assemblies to structural and evolutionary analysis

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Bioinformatics and Systems Biology

by

Andrey V. Bzikadze

Committee in charge:

Pavel A. Pevzner, Chair
Melissa Gymrek, Co-Chair
Vineet Bafna
Vikas Bansal
Karen H. Miga
Siavash Mirarab

2022

The Dissertation of Andrey V. Bzikadze is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2022

DEDICATION

To my precious family that made it all possible. I owe you everything.
My dear grandma, you left us way too soon. I love you. Rest in peace.

EPIGRAPH

How I wish, how I wish you were here
We're just two lost souls swimming in a fishbowl year after year
Running over the same old ground, what have we found?
The same old fears, wish you were here

Roger Waters

In my defence what is there to say
We destroy the love — it's our way
We never listen enough never face the truth
Then like a passing song
Love is here and then it's gone

Freddie Mercury

And I can tell you why people go insane
I can show you how you could do the same

Chris Cornell

Oh, I got a message for you
Up and away, it's what I got to do
Forgive what you have
For what you might lose

Axl Rose

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am indebted to my scientific advisor Pavel A. Pevzner. His supervision and guidance played an immense role in my growth as a scientist. Pavel convinced me to apply to graduate schools in the U.S. It was my privilege to learn from his vast expertise in conducting research, writing manuscripts, and teaching younger generations of researchers.

I am grateful to my committee members for their support and crucial feedback. I thank my colleagues at University of California, San Diego: Anton Bankevich, Mikhail Kolmogorov; at the Center of Algorithmic Biotechnology, St Petersburg University, Russia: Ivan Alexandrov, Dmitry Antipov, Tatiana Dvorkina, Alexey Gurevich, Alla Mikheenko, Olga Kunyavskaya. I was fortunate to be a member of the Telomere-to-Telomere Consortium and I am thankful to its leaders Karen H. Miga and Adam M. Phillippy.

It was my pleasure to start my path in Bioinformatics under supervision of my colleague and friend Yana Safonova. I thank my mentors Vladimir V. Nekrutkin, Nina E. Golyandina, and Anton Korobeynikov for their passion and patience in teaching me Mathematics. Those studies built the foundation for what is presented in this manuscript.

This study would certainly be impossible without invaluable support from my family. My brother, Alexander Bzikadze, helped to shape ideas that lie in foundation of the presented research. I remain grateful to my grandmother, Valentina Dolgosheva. Although she sadly never got to read these lines, she walked with me as far as she could. I thank Cynthia Wu, my significant other and best friend, for always being there for me through thick and thin.

Chapter 1, in full, is a reprint of the material as it appears in Bzikadze, A. V., & Pevzner, P. A. (2020). Automated assembly of centromeres from ultra-long error-prone reads. *Nature Biotechnology, 38*(11), 1309-1316. The dissertation author is the primary developer of the centroFlye algorithm and the first author of this paper.

Chapter 2, in full, is a reprint of the material as it appears in Mikheenko, A., Bzikadze, A. V., Gurevich, A., Miga, K. H., & Pevzner, P. A. (2020). TandemTools: mapping long reads and assessing/improving assembly quality in extra-long tandem repeats. *Bioinformatics,*

36(Supplement_1), i75-i83. The dissertation author is one of the two lead developers of the TandemTools algorithm and one of the two lead authors of this paper.

Chapter 3, in full, has been submitted for a journal publication and may appear as Bzikadze, A. V., Mikheenko, A., & Pevzner, P. A. (2022). Fast and accurate mapping of long reads to complete genome assemblies with VerityMap. The dissertation author is the primary developer of the VerityMap algorithm and the first author of this paper.

Chapter 4, in full, is a reprint of the material as it appears in Kunyavskaya, O., Dvorkina, T., Bzikadze, A. V., Alexandrov, I. A., & Pevzner, P. A. (2022). Automated annotation of human centromeres with HORmon. *Genome Research.* The dissertation author is one of the three lead developers of the HORmon algorithm and one of the three lead authors of this paper.

Chapter 5, in full, is a reprint of the material as it appears in Bankevich, A., Bzikadze, A. V., Kolmogorov, M., Antipov, D., & Pevzner, P. A. (2022). Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature biotechnology*, 1-7. The dissertation author is one of the two lead developers of the LJA algorithm and one of the two lead authors of this paper.

Chapter 6, in full, has been submitted for a journal publication and appears as a *bioRxiv* pre-print: Bzikadze, A. V., & Pevzner, P. A. (2022). TandemAligner: a new parameter-free framework for fast sequence alignment. *bioRxiv*. The dissertation author is the primary developer of the TandemAligner algorithm and the first author of this paper.

VITA

2015	Bachelor of Science, St Petersburg University, St Petersburg, Russia
2017	Master of Science, St Petersburg University, St Petersburg, Russia
2016–2017	Research Scientist, St Petersburg University, St Petersburg, Russia
2022	Doctor of Philosophy in Bioinformatics and Systems Biology, University of California San Diego

PUBLICATIONS

Bzikadze, A. V., & Pevzner, P. A. (2022). TandemAligner: a new parameter-free framework for fast sequence alignment. *bioRxiv*. https://www.biorxiv.org/content/10.1101/2022.09.15.507041.

Bzikadze, A. V., & Mikheenko, A., & Pevzner, P. A. (2022). Fast and accurate mapping of long reads to complete genome assemblies with VeriyMap. *(submitted)*.

Bankevich, A., Bzikadze, A. V., Kolmogorov, M., Antipov, D., & Pevzner, P. A. (2022). Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature biotechnology*, 1-7.

Mc Cartney, A. M., Shafin, K., Alonge, M., Bzikadze, A. V., Formenti, G., Fungtammasan, A., ... & Rhie, A. (2022). Chasing perfection: validation and polishing strategies for telomere-to-telomere genome assemblies. *Nature Methods*, 1-9.

Altemose, N., Logsdon, G. A., Bzikadze, A. V., Sidhwani, P., Langley, S. A., Caldas, G. V., ... & Miga, K. H. (2022). Complete genomic and epigenetic maps of human centromeres. *Science*, 376(6588), eabl4178.

Nurk, S., Koren, S., Rhie, A., Rautiainen, M., Bzikadze, A. V., Mikheenko, A., ... & Phillippy, A. M. (2022). The complete sequence of a human genome. *Science*, 376(6588), 44-53.

Kunyavskaya, O., Dvorkina, T., Bzikadze, A. V., Alexandrov, I. A., & Pevzner, P. A. (2022). Automated annotation of human centromeres with HORmon. *Genome Research.* doi: https://doi.org/10.1101/gr.276362.121.

Dvorkina, T., Kunyavskaya, O., Bzikadze, A. V., Alexandrov, I., & Pevzner, P. A. (2021). CentromereArchitect: inference and analysis of the architecture of centromeres. *Bioinformatics, 37*(Supplement_1), i196-i204.

Logsdon, G. A., Vollger, M. R., Hsieh, P., Mao, Y., Liskovykh, M. A., Koren, S., ... & Eichler, E. E. (2021). The structure, function and evolution of a complete human chromosome 8. *Nature, 593*(7857), 101-107.

Mikheenko, A., Bzikadze, A. V., Gurevich, A., Miga, K. H., & Pevzner, P. A. (2020). TandemTools: mapping long reads and assessing/improving assembly quality in extra-long tandem repeats. *Bioinformatics, 36*(Supplement_1), i75-i83.

Dvorkina, T., Bzikadze, A. V., & Pevzner, P. A. (2020). The string decomposition problem and its applications to centromere analysis and assembly. *Bioinformatics, 36* (Supplement_1), i93-i101.

Miga, K. H., Koren, S., Rhie, A., Vollger, M. R., Gershman, A., Bzikadze, A., ... & Phillippy, A. M. (2020). Telomere-to-telomere assembly of a complete human X chromosome. *Nature, 585*(7823), 79-84.

Bzikadze, A. V., & Pevzner, P. A. (2020). Automated assembly of centromeres from ultra-long error-prone reads. *Nature Biotechnology, 38*(11), 1309-1316.

Zhang, C., Bzikadze, A. V., Safonova, Y., & Mirarab, S. (2020). Scalable Models of Antibody Evolution and Benchmarking of Clonal Tree Reconstruction Methods. *bioRxiv*. doi: https://doi.org/10.1101/2020.09.17.302505.

Shlemov, A., Bankevich, S., Bzikadze, A., Turchaninova, M. A., Safonova, Y., & Pevzner, P. A. (2017). Reconstructing antibody repertoires from error-prone immunosequencing reads. *The Journal of Immunology, 199*(9), 3369-3380.

Bzikadze, A. V., & Nekrutkin, V. V. (2016). On some statistical properties of the "Book Stack" transformation. *Vestnik St. Petersburg University: Mathematics, 49*(4), 305-312.

ABSTRACT OF THE DISSERTATION

Human centromeres: from initial assemblies to structural and evolutionary analysis

by

Andrey V. Bzikadze

Doctor of Philosophy in Bioinformatics and Systems Biology

University of California San Diego, 2022

Pavel A. Pevzner, Chair
Melissa Gymrek, Co-Chair

Recent advances in long-read sequencing technologies allowed generation of the first complete assembly of a human genome. They revealed previously inaccessible sequences of human centromeres and allowed analysis of their structure and evolution. We introduce centroFlye — the first algorithm for automated assembly of centromeres from error-prone long reads. We then describe TandemTools and VerityMap algorithms for quality assessment of the newly assembled regions. Afterwards, we present HORmon algorithm for structural and evolutionary analysis of human centromeres. We introduce LJA — the first de Bruijn-based genome assembler for accurate long reads. Finally, we describe TandemAligner —– the first parameter-free sequence alignment algorithm that introduces a sequence-dependent scoring that automatically changes for any pair of compared sequences.

Chapter 1

Automated assembly of centromeres from ultra-long error-prone reads

1.1 Abstract

Centromeric variation has been linked to cancer and infertility, but centromere sequences contain multiple tandem repeats and can only be assembled manually from long error-prone reads. Here we describe the centroFlye algorithm for centromere assembly using long error-prone reads, and apply it to assemble human centromeres on chromosomes 6 and X. Our analyses reveal putative breakpoints in the manual reconstruction of the human X centromere, demonstrate that human X chromosome is partitioned into repeat subfamilies and provide initial insights into centromere evolution. We anticipate that centroFlye could be applied to automatically close remaining multimegabase gaps in the reference human genome.

1.2 Introduction

Long-read technologies (such as Pacific Biosciences and Oxford Nanopore) have greatly increased the contiguity of genome assemblies as compared to short-read technologies. However, the existing long-read assemblers, such as Falcon [1], Miniasm [2], Flye [3], HINGE [4], Canu [5], Marvel [6] and wtdbg2 [7] typically fail to resolve long segmental duplications [8] and long tandem repeats [3]. This paper focuses on the latter challenge of assembling long tandem repeats,

and specifically on centromere assembly, a problem that was viewed as intractable until recently. As shown in [3], the existing long-read assemblers are inaccurate, even in the case of relatively short bridged tandem repeats that are spanned by long reads. Although Flye improved on other tools in assembling such repeats [3], assembly of unbridged tandem repeats remains an open problem.

Centromeric satellite repeats are among the longest tandem repeats in the human genome and the biggest gaps in the reference human genome assembly (for brevity, we refer to such regions simply as centromeres). As a result, studies of associations between sequence variations and genetic diseases currently ignore roughly 3% of the human genome. This is unfortunate since centromeres play crucial roles in chromosome segregation and a large component of genetic disease result from aneuploidies arising during meiosis [9]. In addition, variations in centromeres are linked to cancer and infertility [10], [11], [12], [13], [14], [15], [16], [17], [18]. Centromere sequencing is also important for addressing open problems about centromere evolution [19], [20], [21], [22], [23]. These studies revealed fast evolution of centromeres; indeed, complex higher-order chromosome-specific centromeric repeats are unique to the hominid lineage and are not chromosome-specific even in closely related species such as gibbons [24]. Recent discovery of large archaic blocks of Neanderthal DNA spanning human centromeres reveals the potential of centromeres for studies of human population history [25].

Human centromeres comprise long tandem repeats (also known as satellite DNA) that are often repeated thousands of times with extensive variations in copy numbers in the human population. Although long-read technologies facilitated analysis of centromere on Y chromosome [26], there is no software tool for centromere reconstruction and it remains unclear how accurate semimanual centromere reconstructions are.

We report the centroFlye algorithm for centromere assembly and apply our algorithm to enable automatic reconstruction of the centromeres on chromosomes 6 and X (referred to as cen6 and cenX).

1.3 Results

1.3.1 Centromere structure

The alpha satellite repeat family forms roughly 3% of the human genome [27]. Each alpha satellite (monomer) is around 171 nucleotides in length. Blocks of multiple consecutive monomers form higher-order repeat (HOR) units that can be repeated thousands of times. Individual monomers within a HOR show low (50–90%) sequence identity to each other while HORs within a single centromere show high (95–100%) sequence identity. Organization and nucleotide sequence of HORs is specific for a particular chromosome. HORs typically occupy multimegabase-sized segments that may include rearrangements and transposon insertions [28]. Since centromere assembly of a diploid genome is particularly challenging, studies of the centromeres on X and Y chromosomes in the male genome represent a simpler (albeit still very complex) problem [19], [29], [30], [31].

Human centromeres differ widely with respect to their architecture. For example, the centromere on chromosome X (cenX) is built from a single 12-monomer HOR with very few abnormal HORs. In contrast, the centromere on chromosome 6 (cen6) has a highly irregular architecture formed by three HORs (consisting of 15, 16 and 18 monomers) and many abnormal HORs. These two examples (highly regular and highly irregular centromere architectures) exemplify two extremes and result in two different algorithmic challenges. centroFlye thus has two modes: the HOR mode (designed for centromeres with a single HOR such as cenX) and the monomer mode (designed for centromeres with multiple HORs and irregular architecture such as cen6).

1.3.2 CentroFlye pipeline

centroFlye in the HOR mode (centroFlye$_{\text{HOR}}$) takes a read set from the entire genome and a consensus HOR (characterizing a specific centromere) as an input. centroFlye$_{\text{HOR}}$ modifies the approach to resolving unbridged repeats used in the Flye assembler [3] for the case of tandem

repeats. Flye finds divergent positions within an unbridged repeat (positions where repeat copies differ from each other) and uses them as stepping stones for resolving these repeats. However, since mapping of reads within cenX is unknown, it is unclear how to infer the divergent positions between various copies of a HOR. centroFlye$_{\text{HOR}}$ instead defines a set of rare k-mers (k-mers that appear in a single or a few copies of a HOR) and uses them for reconstructing centromeres. Fig. 1.7 reveals frequencies of rare k-mers in the assembly.

The centroFlye pipeline in the HOR mode consists of the following steps (Fig. 1.1): (1) recruiting centromeric reads, (2) partitioning centromeric reads into units, where each unit represents a HOR copy, (3) classifying centromeric reads into prefix reads (that start before the centromere and 'enter' it), internal reads (located entirely within the centromere) and suffix reads (that start in the centromere and 'leave' it), (4) identifying rare centromeric k-mers, (5) constructing the distance graph to filter out false positives among rare centromeric k-mers, (6) reconstructing the centromere, and (7) polishing the reconstructed centromere sequence. The Methods section describes each of these steps and illustrates them using the cenX assembly.

In difference from centroFlye in the HOR mode (that under-uses abnormal units in assembly), centroFlye in the monomer mode (referred to as centroFlye$_{\text{mono}}$) uses abnormal units as markers for assembly. centroFlyemono takes a read-set from the entire genome and the set of monomers for a specific chromosome as an input (Fig. 1.2). It includes the following steps (Fig. 1.2): (1) recruiting centromeric reads, (2) transforming reads into monoreads; that is, translating each read from the nucleotide alphabet into the monomer alphabet, (3) error correction of monoreads, (4) constructing the iterative de Bruijn graph [32], [33] of monoreads, (5) assembling monocentromere by scaffolding in the iterative de Bruijn graph and (6) translating monocentromere back from the monomer to the nucleotide alphabet. The Methods section describes each of these steps and illustrates them using the cen6 assembly. Next we focus on the cenX assembly.

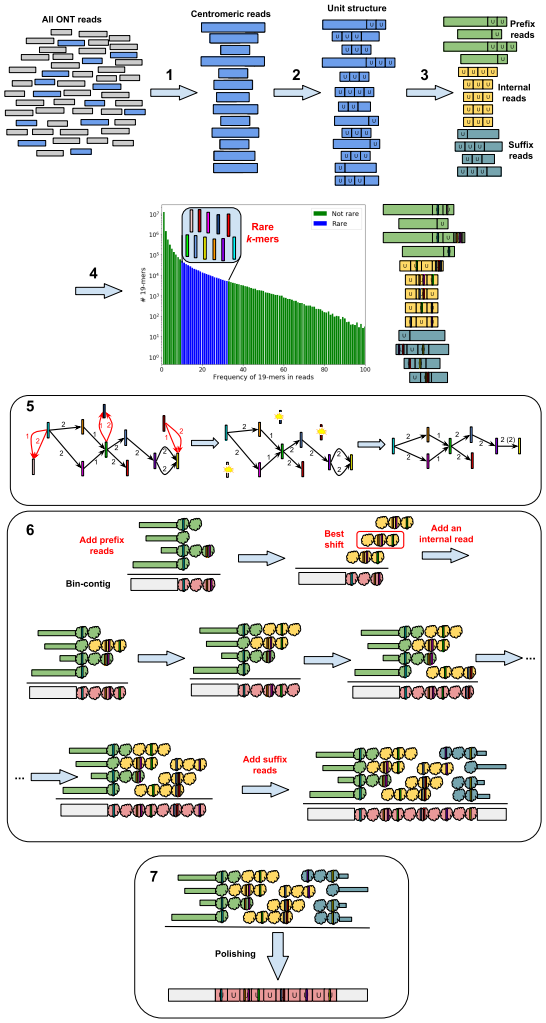


Figure 1.1. (1) Recruitment of centromeric reads from the entire read-set. (2) Partitioning each read into units, where each unit represents a HOR copy. (3) Classifying centromeric reads into prefix, internal and suffix reads. (4) The frequency histogram of all k-mers in centromeric reads reveals rare k-mers. Each colored vertical bar represents a rare k-mer. k-mers with lower frequencies than rare k-mers likely represent sequencing errors. (5) Construction of the distance graph. Rare k-mers represent vertices and each edge connects a pair of rare k-mers occurring in the same read. Edge labels represent distances (in units) between rare k-mers in a read. An edge is red if there are conflicting parallel edges connecting corresponding vertices, and black otherwise. (Left) The distance graph constructed on all rare k-mers. (Middle) The distance graph with conflicting parallel edges and isolated vertices removed. (Right) The final distance graph with collapsed multiedges. (6) Reconstruction of a centromere. Each unit in each read is represented as a bin (see Methods) with colored bars representing unique k-mers. After all prefix reads are added to a bin-contig, the best alignment (shift) of each read against the bin-contig is selected and the read with the highest-scoring alignment is added to the growing bin-contig. This procedure is repeated until the suffix reads are added to the bin-contig. (7) Polishing the reconstructed centromere sequence.

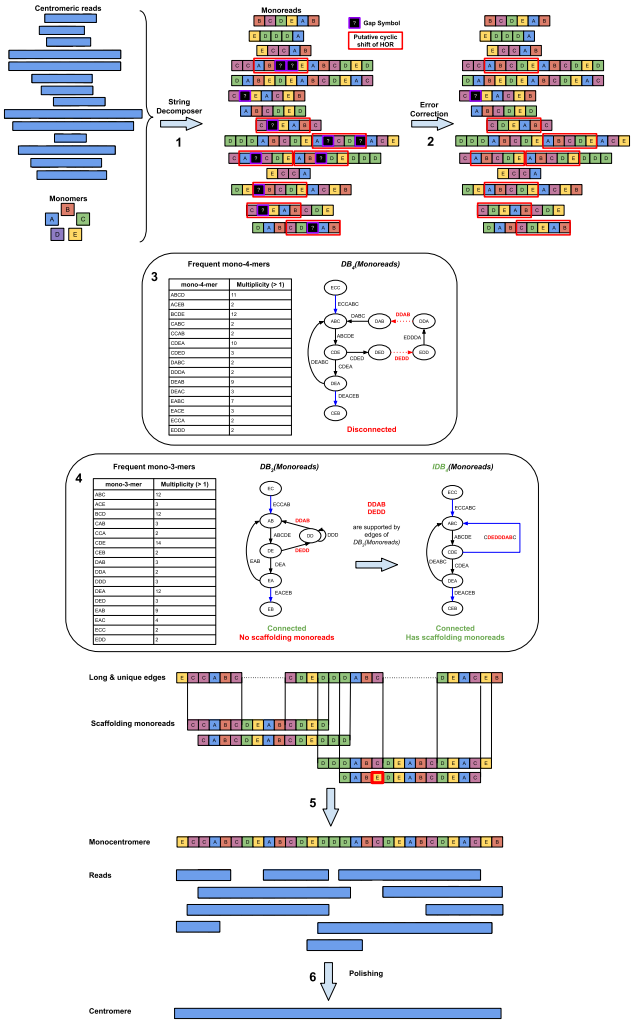


Figure 1.2. For the sake of illustration, we assume only five monomers forming a HOR ABCDE. (1) Transforming centromeric reads into monoreads. (2) Error correction of monoreads. (3) The de Bruijn graph DB_4(Monoreads) (with $k = 3$) is shown just to illustrate the limitations of standard de Bruijn graph for centromere assembly and to motivate our use of the iterative de Bruijn graphs instead. The graph DB_4(Monoreads) is constructed on frequent mono-4-mers (minMultiplicity, 2). Red dotted edges DEDD and DDAB are not present in this disconnected graph since they represent infrequent mono-4-mers. Long (MinLength, $k+2$) and unique edges are shown in blue. (4) Construction of the iterative de Bruijn graph IDB4(Monoreads) starts from constructing the standard de Bruijn graph DB_3(Monoreads). Although DB_3(Monoreads) is connected, it does not enable unique centromere assembly since there are no scaffolding reads that span two long and unique edges. However, both mono-4-mers DEDD and DDAB (that are missing in DB_4(Monoreads)) represent edges in DB_3(Monoreads) that are inherited by IDB_4(Monoreads). As a result, unlike DB_4(Monoreads), IDB_4(Monoreads) is connected. It contains three long and unique edges. (5) Assembling monocentromere by scaffolding in IDB_4(Monoreads). A mismatch in a monoread is highlighted with a red border. (6) Translating monocentromere back from the monomer to the nucleotide alphabet. Only monoreads that have unambiguous mapping to the monocentromere are used for polishing.

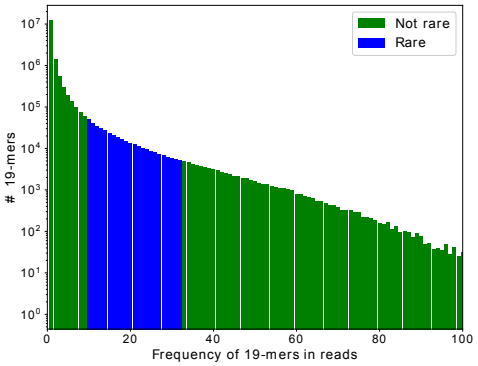
1.4 cenX assembly

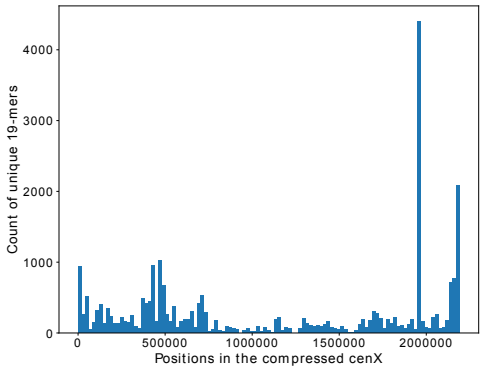
We analyzed the dataset of Oxford Nanopore Technologies (ONT) reads generated by the Telomere-to-Telomere consortium [34] and released on 2 March 2019. The dataset contains 11,069,717 reads (155 gigabases (Gb) total length, 50× coverage, N50 = 70 kilobases (kb)) generated from the CHM13hTERT female haploid cell line. 999,562 ultra-long reads (longer than 50 kb) have the biggest impact on the centromere assembly and result in around 32× coverage.

In addition to the centroFlye assembly, we analyzed the telomere-to-telomere consortium assemblies v.0.4 (referred to as the T2T4 assembly) and v.0.6 (referred to as the T2T6 assembly). Figure 1.3 presents information about cenX assemblies. Since ONT assemblies often have inflated lengths of homonucleotide runs, we compressed each homonucleotide run into a single nucleotide (in the read-set and assemblies) and recomputed the number of unique 19-mers. Figure 1.3 shows the distribution of frequencies of unique 19-mers in the compressed assemblies and illustrates that centroFlye and T2T6 assemblies have similar distributions of frequencies, while the T2T4 assembly has many low-frequency unique 19-mers that are likely erroneous.

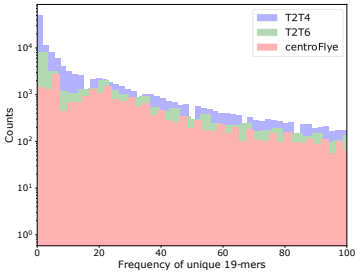
Given a centromere assembly, one can map each centromeric read to this assembly using its unique k-mers. Next we describe how to use this mapping for comparison of various assemblies. To illustrate the effect of an indel on various quality metrics we constructed an artificial centroFlye$_{\text{del}}$ assembly by introducing a deletion of length 50 kb (25 units) in the centroFlye assembly at position 600 kb (300 units).

For each pair of assemblies, we used their shared unique 19-mers to align centromeric reads to them. Figure 1.4 compares positions of read alignments to each pair of assemblies and reveals structural discrepancies between them. Comparison of centroFlye and centroFlye$_{\text{del}}$ assemblies reveals an expected discrepancy around unit 300. Both T2T4 and T2T6 assemblies differ from the centroFlye assembly around units 150–180 in the centroFlye assembly although the centroFlye and T2T6 are more complaint in this area. The size of deletion in T2T4 is





Assemblies	Length (kb)	Length (units)	# unique 19-mers	# (%) of unique & rare 19-mers
centroFlye	3,104	1,510	39,530 / 26,333	17,298 (44%)
T2T4	2,723	1,337	243,433 / 126,595	37,368 (15%)
T2T6	2,838	1,379	49,055 / 33,574	16,915 (35%)
centroFlye&T2T4			21,146 / 15,604	11,612 (55%)
centroFlye&T2T6			29,803 / 22,817	15,320 (51%)
T2T4&T2T6			22,427 / 16,242	11,827 (53%)



assembly 1	assembly 2	# centromeric reads recruited to both assemblies	# discordant reads		discordance(assembly1, assembly2)
			preference for assembly 1	preference for assembly 2	
centroFlye	T2T6	1,535	54	3	+5,609
centroFlye	T2T4	1,234	97	5	+7,129
T2T6	T2T4	1,670	77	8	+4,636
centroFlye	centroFlye$_{del}$	2,385	13	0	+5,503

Figure 1.3. a, Frequency histogram of 19-mers in the centromeric reads. Each bar represents the number of 19-mers in units of centromeric reads with a given frequency (log-scale). The bars corresponding to the rare 19-mers are shown in blue. Only 19-mers with frequencies that do not exceed 100 are shown. The 19-mers with lower frequencies than rare 19-mers likely represent sequencing errors. **b**, Distribution of unique 19-mers along the compressed cenX sequence. Each bar represents the number of unique 19-mers in a segment of length 20 kb (out of 26,724 unique 19-mers in the compressed cenX sequence). A large peak at positions 1,955,990–1,960,812 corresponds to a 4,822 nucleotide long (compressed) LINE insertion in cenX. **c**, Comparison of centroFlye, T2T4 and T2T6 assemblies. The column 'No. of unique 19-mers' shows the number of unique 19-mers before/after compression of homonucleotide runs. The column 'No. (percentage) of unique and rare 19-mers' refers to the number (percentage) of unique 19-mers in an assembly that are rare in reads. The T2T4 centromere has 57,811,689–60,534,892 coordinates and the T2T6 centromere has 57,827,622–60,665,308 coordinates on chromosome X. **d**, Distribution of frequencies (in logarithmic scale) of unique 19-mers in the compressed centroFlye, T2T4 and T2T6 cenX assemblies. **e**, Number of recruited reads, discordant reads and the discordance score for all pairs of assemblies.

around 32 units (around unit 135), and in T2T6—only 5 units (around unit 175). Four other discrepancies are shared between both versions of T2T and the centroFlye assembly (Fig. 1.4). Next, we argue that T2T assemblies have putative misassembles in the surrounding areas.

1.4.1 Quality assessment of the centromere assemblies

Although Supplementary Note 1 in [35] demonstrates that centroFlye accurately reconstructs simulated centromeres using reads simulated by the NanoSim simulator [36], it is unclear how to evaluate assemblies of real centromeres. Benchmarking of various genome assemblers would not be possible without the quality assessment tools such as QUAST [37]. However, since QUAST is not applicable for analyzing centromere assemblies, we developed some metrics for the reference-free quality assessment of centromere assemblies.

Errors in an assembly affect the coverage near the assembly breakpoints. Thus, in the case of a uniform coverage, regions with abnormal coverage may point to assembly errors. For example, a deletion inflates the coverage near the deletion breakpoint (doubles the coverage in the case of a long deletion) and an insertion reduces the coverage in the inserted segment. Figure 1.5 shows the coverage plots for all assemblies and reveals that a deletion in centroFlye$_{\text{del}}$ assembly inflates the coverage by roughly 60% at the deletion breakpoint (from ˜50× to ˜80×).

It turned out that the analyzed ONT dataset is characterized by a nonuniform read distribution, making it difficult to infer assembly errors from irregularities in the read coverage (Fig. 1.8). Even though T2T assemblies of cenX demonstrate higher coverage variations than the centroFlye assembly, these variations do not necessarily point to assembly errors, necessitating a need to introduce additional metrics for centromere assemblies. The T2T6 assembly coverage is more similar to the centroFlye assembly coverage (as compared to T2T4 assembly), but has a large spike at the end of the array, which is coordinated with a discrepancy in Fig. 1.4.

A k-mer is shared between an assembly and a read aligned to this assembly if it occurs in both the assembly and the read at the same position in their alignment. Given a set of k-mers Anchors, we define shared$_{\text{Anchors}}$(Read,Assembly) as the number of k-mers from Anchors that are

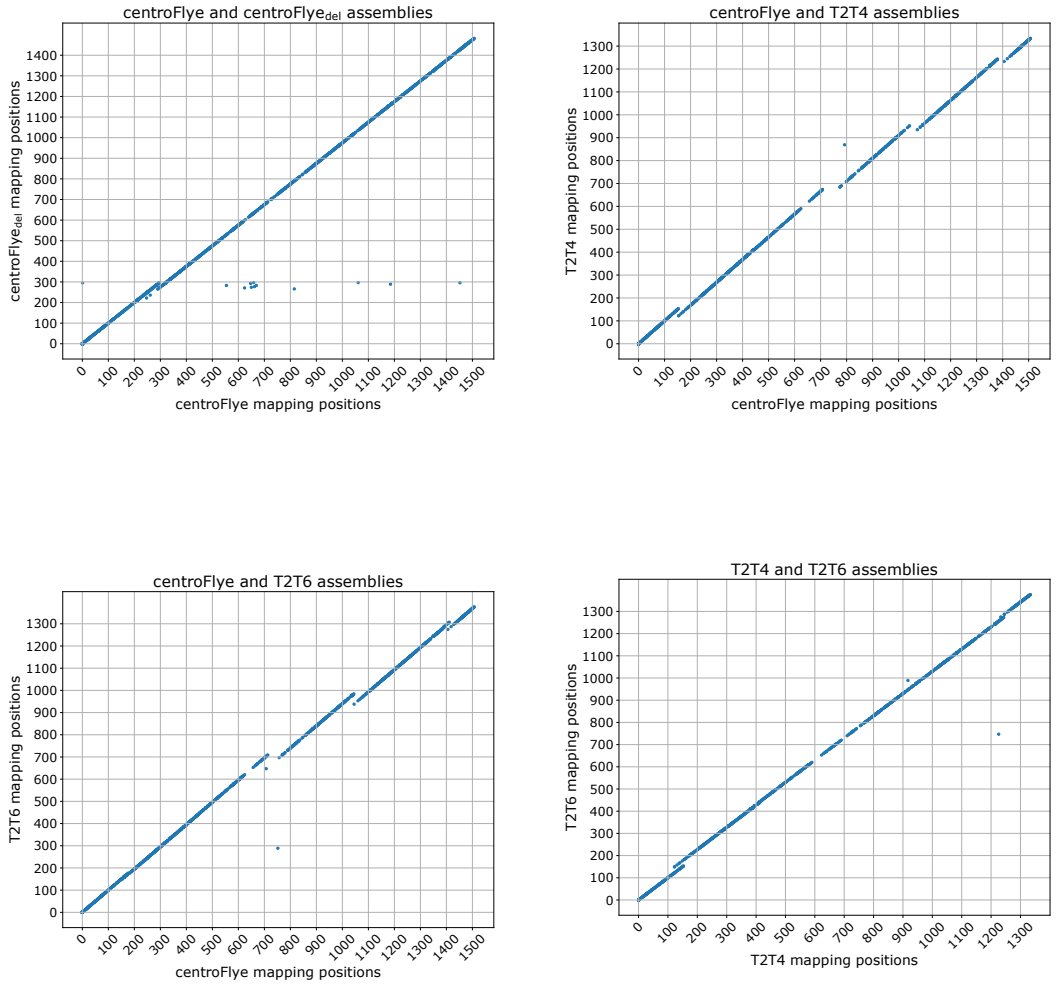


Figure 1.4. Each dot corresponds to a centromeric read. The x and y coordinates represent the starting unit position in the corresponding assemblies. **a**, Comparison of centroFlye and centroFlye$_{del}$ assemblies reveals a discrepancy around unit 300—25 units deletion in the centroFlye$_{del}$ assembly. **b**, Comparison of centroFlye and T2T4 assemblies reveals discrepancies around the following units in centroFlye (T2T4) assemblies: 150 (135)—32 units deletion in T2T4, 450 (410)—2 units deletion in T2T4, 750 (700)—56 units deletion in T2T4, 1,050 (950)—47 units deletion in T2T4 and 1,400 (1,250)—36 units deletion in the T2T4 in the centroFlye (T2T4) assembly. **c**, Comparison of centroFlye and T2T6 assemblies reveals discrepancies around the following units in centroFlye (T2T assemblies): 180 (175)—5 units deletion in T2T6, 450 (445)—1 unit deletion in T2T6, 750 (720)—56 units deletion in T2T6, 1,050 (975)—47 units deletion in T2T6, 1,400 (1,300)—24 units deletion in T2T6. **d**, Comparison of T2T4 and T2T6 assemblies reveals discrepancies around units 150 and 1,240 in both assemblies.

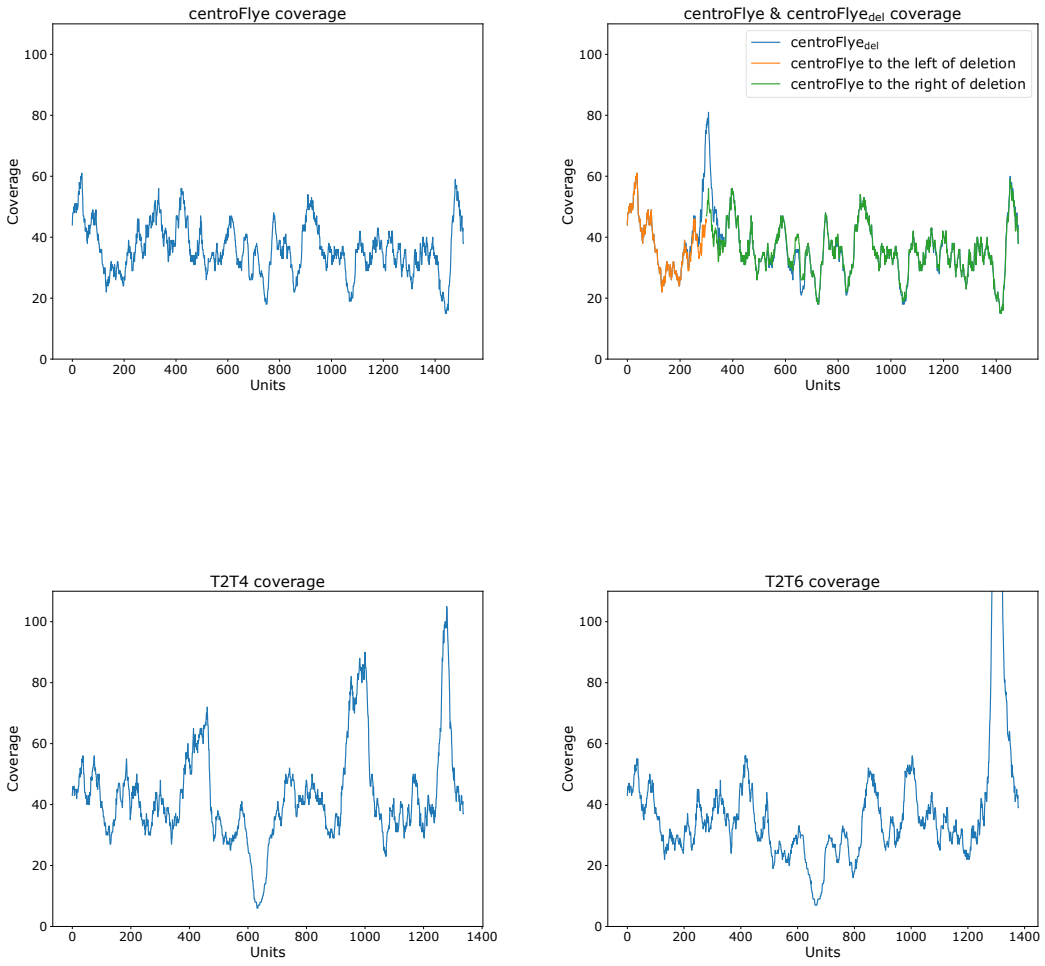


Figure 1.5. centroFlye$_{del}$ refers to the centroFlye assembly of cenX with an artificially introduced 50 kb (25 units) deletion at position 600 kb (unit 300). The peak around unit 1,300 in T2T6 gives coverage of almost 1,000 and was cut. The reduction in the number of mappable reads caused by the deletion in centroFlye$_{del}$ assembly affects the coverage near the deletion in this assembly: 1,696 and 1,676 reads were mapped to centroFlye and centroFlye$_{del}$ assemblies, respectively.

shared between Read and Assembly. The larger is $\text{shared}_{\text{Anchors}}$(Read,Assembly), the better the assembly 'explains' the read. Given a read-set Reads, we define $\text{shared}_{\text{Anchors}}$(Reads,Assembly) as the sum of $\text{shared}_{\text{Anchors}}$(Read,Assembly) over all reads in Reads.

To compare the assemblies Assembly and Assembly', we define Anchors as the set of shared unique k-mers between them (default $k = 19$) and compute the discordance between these assemblies as discordance(Assembly,Assembly') = $\text{shared}_{\text{Anchors}}$(Reads,Assembly) - $\text{shared}_{\text{Anchors}}$(Reads, Assembly'). For centroFlye and T2T6 assemblies, discordance(centroFlye, T2T6) = 5,609, suggesting that the centroFlye assembly is a better fit for the read set than the T2T6 assembly (Fig. 1.3).

We classify a read Read as discordant with respect to assemblies Assembly and Assembly' and a k-mer-set Anchors if there is a large difference (by at least k) between $\text{shared}_{\text{Anchors}}$(Read, Assembly) and $\text{shared}_{\text{Anchors}}$(Read, Assembly'), thus showing preference for one of the assemblies. A discordant read votes for Assembly (Assembly') if this difference is positive (negative). There are 54 (3) discordant reads voting for centroFlye (T2T6) assemblies. Figure 1.3 illustrates that the centroFlye assembly improves on other assemblies with respect to the discordance score.

A concentration of discordant reads at a certain region voting for Assembly over Assembly' suggests that Assembly' has a multi-unit deletion in this region. Figure 1.6 reveals three clusters of discordant reads voting for centroFlye over T2T6 assembly at the regions of around 200, 400–600 and 1,400 units in the centroFlye assembly (only two discordant reads vote for the T2T6 over the centroFlye assembly). These regions are coordinated with discrepancies shown in Fig. 1.4 and likely point to large deletions in the T2T6 assembly. Similar comparison between centroFlye and T2T4 assemblies reveals putative misassembles in T2T4 roughly at units 200, 400, 800, 1,150 and 1,400 in the centroFlye assembly. As expected, comparison of centroFlye and $\text{centroFlye}_{\text{del}}$ detects a single deletion at unit 300.

Supplementary Note 2 in [35] describes the hanging index test and the breakpoint test that provide additional support for the centroFlye assembly.

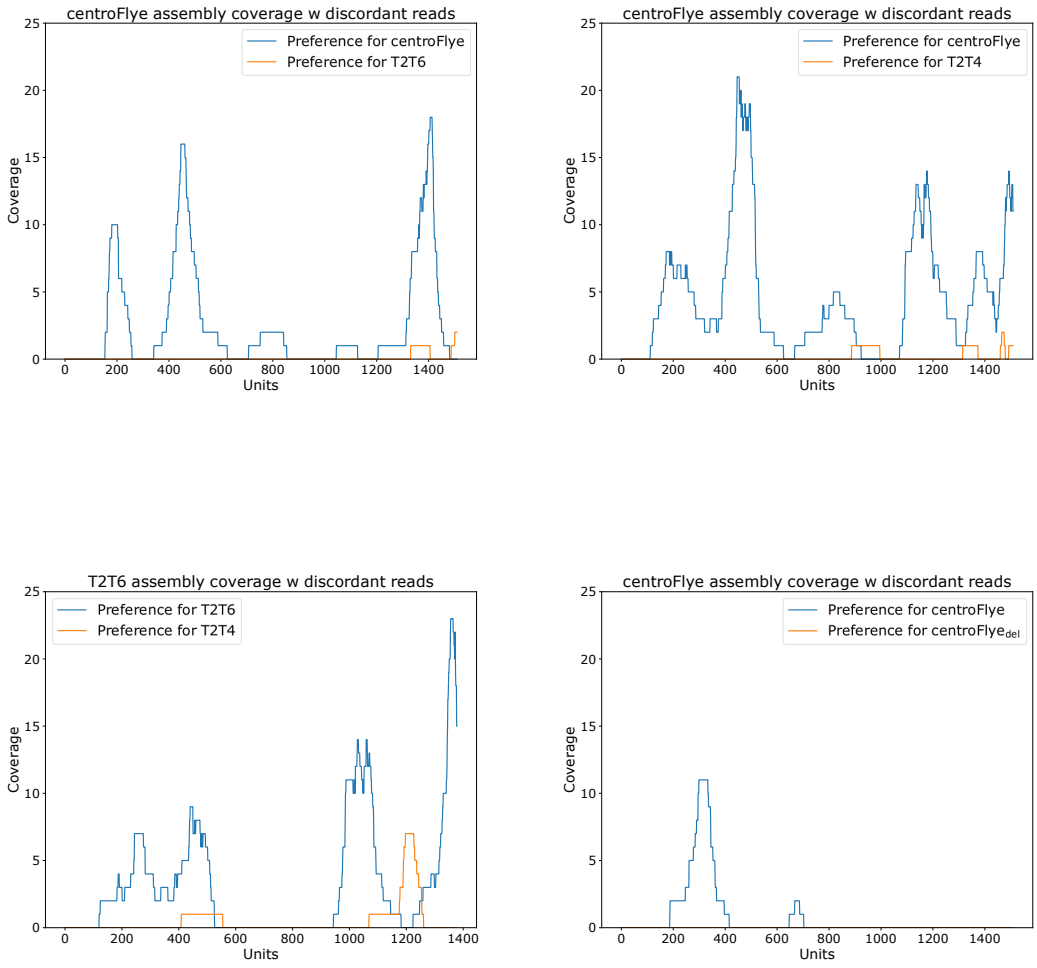


Figure 1.6. Three (five) clusters of discordant reads vote for the centroFlye assembly over the T2T6 (T2T4) assembly suggesting multi-indel deletions in the T2T assemblies. A single cluster of discordant reads votes for the centroFlye assembly over the centroFlye$_{del}$ assembly and detects the existing deletion at unit 300 in the centroFlye$_{del}$ assembly.

1.4.2 Variations in HORs provide insights into centromere evolution

Various copies of a repeat are usually partitioned into subfamilies that reflect evolutionary history of this repeat. For example, Alu repeats in the human genome are split into over 200 subfamilies that provide insights into evolutionary history of Alu repeats [38]. Some 'active' Alu copies continue to produce variation by 'jumping' to new genomic locations, thus posing the mutagenic threat to the human genome [39].

To reveal subfamilies of the cenX HOR, we align each canonical HOR in the homonucleotide compressed centroFlye cenX assembly (1,471 out of 1,510 units) to the compressed DXZ1* and use the resulting multiple alignment to construct the profile logo for every position of DXZ1*. Although most positions in units are highly conserved, some positions reveal substantial variations (Fig. 1.9). We select divergent positions such that the ratio of the secondary vote to the majority vote is greater than minRatio (default value minRatio = 0.3) and the secondary vote is not a deletion. This procedure reveals seven divergent positions. Each divergent position is characterized by its majority vote frequency and secondary vote frequency.

We consider pairs of divergent positions that are at least minDistance (default value minDistance = 100) positions apart and select the 'most correlated' pairs using the biprofile approach [40]. Afterward, we apply the χ^2 test of independence and select the pair of divergent positions with the smallest p-value. The divergent positions 580 and 695 with majority votes 71 and 72% (for nucleotides 'AA') and secondary votes 30 and 28% (for nucleotides 'GG') were selected. For these positions, the biprofile frequency for 'AA' and 'GG' is 69 and 27%, respectively, resulting in $p < 10^{-10}$. We split the set of all units into two clusters with respect to nucleotides at the selected pair of positions, resulting in a split of all canonical HOR into two clusters with 1,010 and 38 units, respectively. We repeat this process iteratively for each of the new clusters until no more clusters of size greater than minSize (default value minSize, 100) are generated. This process stops after two iterations and results in four subfamilies of cenX HOR. Fig. 1.10 reveals that units in different subfamilies are alternating along cenX, providing

an initial insight into how subfamilies 'colonize' cenX during its evolution.

Since we derived HOR subfamilies without using positions of the units, clustering of each subfamily into close positions along cenX suggests that the algorithm originally described for identifying Alu subfamilies [7] also works for HOR subfamilies. However, our results also reveal a new phenomenon of HOR recombination and suggest that analysis of centromere evolution may be more complex that analysis of Alu evolution.

1.5 Discussion

Although the role of centromeres (chromosome segregation) is conserved throughout evolution, centromere sequences vary among species. For example, the X chromosome is highly conserved across all mammals, but the mammalian X centromeres vary across mammalian species. Here we enable detailed study of centromeres by devising and validating centroFyle, an automatic tool for centromere assembly.

We compared centroFlye assembly with the semimanual T2T assemblies and identified several misassembled parts in the T2T assembly. Figure 1.4 reveals five large discrepancies (deletions) between centroFlye and T2T6 assemblies, Fig. 1.5 shows highly inflated coverage around unit 1,300 in T2T6 assembly, Fig. 1.6 describes discordant reads that reveal three problems in T2T6 assembly, Supplementary Note 2 in [35] describes high hanging index in five regions in T2T6 assembly and five potential breakpoints in the T2T6 assembly. The TandemTools software for analyzing centromere assemblies [41] confirmed our conclusions and demonstrated that the problems we detected in the T2T assemblies are not specific to the mapping method. The latest version of the T2T cenX assembly published on 24 October 2019 (T2T v.0.7) incorporates some elements of centroFlye assembly, for example, corrects the duplication errors in the previous assembly, and was further polished using a new marker-assisted read mapping strategy using both nanopore and PacBio CLR reads.

Although centroFlye could be used to fill the largest remaining gaps in the human genome

and study centromere evolution, further algorithmic developments are needed to assemble all human centromeres. For example, although centroFlye revealed 37 abnormal HORs in cenX assembly (Supplementary Note 3 in [35]), these HORs were not used to guide the centroFlye$_{\text{HOR}}$ assembly. We thus extended the functionality of the centroFlye algorithm by developing the centroFlye$_{\text{mono}}$ mode for analyzing centromeres with highly irregular HORs such as cen6. Although this approach successfully assembled cen6 it remains unclear how to recruit reads to centromeres with shared monomers. For example, some chromosomes share the same monomers or HORs with other chromosomes; for example, human chromosomes 1, 5 and 19 share the same HOR D1Z7/D5Z2/D19Z3 [42]. We hope to extend centroFlye to address the read recruitment challenge as well as the challenges of reconstructing centromeres in diploid genomes and identifying functional centromere sequences by coanalyzing centromere assemblies and chromatin immunoprecipitation-sequencing data [43].

1.6 Methods

1.6.1 Recruiting centromeric reads

centroFlye$_{\text{mono}}$ recruits centromeric reads for a specific chromosome by identifying all reads that align to HORs from this chromosome (see Fig. 1.1 step (1)). It uses the fitting alignment of HORs to all reads and recruits reads with sequence identity exceeding a threshold. centroFlye$_{\text{mono}}$ uses a sequence identity threshold (the default value is 83% because most human HORs differ from the HOR consensus by less than 5% and the error rate in reads is $\tilde{1}2\%$). In the case of cenX, centroFlye$_{\text{mono}}$ recruits 2,680 centromeric reads (total length $\tilde{1}33$ Mb) that align to DXZ1 or its reverse complement.

In the following, we assume that all centromeric reads have DXZ1 in the forward orientation and complement a read if it is not the case (Supplementary Note 4 in [35]). Of the centromeric reads, 150 have lengths varying from 2 to 5 kb, 1,382 reads are longer than 30 kb and 897 of them are ultra-long. The longest centromeric read is 527 kb.

1.6.2 Partitioning centromeric reads into units

DXZ1 was derived at the dawn of the sequencing era based on limited sequencing data [44] (see Fig. 1.1 step (2)). Supplementary Note 5 in [35] describes how to infer a more accurate consensus HOR DXZ1* for cenX.

We use the noise-cancelling repeat finder (NCRF) [45] to partition each centromeric read into units. Given a read and a consensus HOR, NCRF partitions a read into units, each unit representing a single copy of a HOR. Although NCRF was not designed to characterize the possible gaps between units (for example, transposon insertions or small rearrangements), this limitation of NCRF does not considerably affect the centroFlye results. If NCRF reports several alignments for a given read, the longest one is kept. Incomplete units appearing at prefix or suffix of a read are discarded.

centroFlye discards all centromeric reads with (the longest) alignment shorter than three units. So, 295 centromeric reads (including 42 ultra-long reads) of total length 7 Mb were discarded. NCRF identified 56,138 units in the remaining 2,385 centromeric reads, including 39,363 units in the remaining 855 ultra-long reads.

1.6.3 Classifying centromeric reads

A centromeric read is classified as a prefix (suffix) read if it has a prefix (suffix) of length at least prefixThreshold that does not match the HOR consensus (default threshold prefixThreshold = 50 kb) (Fig. 1.1 step (3)). Otherwise, a read is classified as internal. NCRF revealed 15 prefix, 2,357 internal and 13 suffix reads. This classification is important for 'moving inside the centromere' using an approach similar to the approach for reconstructing unbridged repeats in Flye [3].

1.6.4 Identifying rare centromeric *k*-mers

We define the frequency of a *k*-mer as the number of occurrences of this *k*-mer in units of reads from a centromeric read-set. centroFlye identifies rare centromeric *k*-mers by analyzing

k-mers with frequencies that fall into a predefined interval (centroFlye uses the default value k = 19) (Fig. 1.1 step (4)). Since a HOR may be repeated in a centromere thousands of times (with small variations), we expect most k-mer from a HOR to have high frequencies. Our goal is to identify rare k-mers that appear just once (unique k-mers) or a few times in a centromere and use them for centromere assembly. Note that a k-mer from a genome 'survives' without errors in a read with the survival rate that can be approximated as survivalRate(k) = $(1-p)^k$, where p is the probability of an error at a given position of a read. A more accurate estimate from the real data suggests that the survival rate of 19-mers in the ONT reads is 0.34. Thus, since the recruited set of ultra-long cenX reads has roughly 32× coverage, we expect that a unique k-mer from a given position in the genome survives in around 11 ultra-long centromeric reads.

We define the interval (bottom, top) and classify a k-mer as rare if its frequency is larger than bottom*survivalRate*coverage and smaller than top*survivalRate*coverage (the default values were bottom, 1 and top, 3). Although 391,361 k-mers from centromeric reads were classified as rare (Fig. 1.3, a), many of them represent erroneous versions of k-mers from a HOR copy rather than truly rare k-mers in cenX. Indeed, the number of reads containing a given k-mer b is affected by the number of genomic positions with k-mers similar to b since error-prone reads covering these similar k-mers may contain b. Since many HOR copies may contain k-mers similar to a unique k-mer, this observation explains the complications in inferring the set of unique/rare k-mers. For example, a single nucleotide insertion in a k-mer from a genome occurs in an ONT read with probability of 0.03 [46]. Thus, each such insertion in a k-mer from DXZ1* has a high chance of being classified as a rare k-mer. Next, we describe how to filter out such spurious rare k-mers using the distance graph.

1.6.5 Constructing the distance graph

The key observation to separate false positives from unique k-mers is that the distances between unique k-mers in reads are likely to be conserved but distances between false positive k-mers are not necessarily conserved (Fig. 1.1 step (5)). The distance graph reveals pairs of

unique k-mers that are separated by roughly the same distance in multiple reads.

Given a set of rare centromeric k-mers V, we define the weighted directed distance graph with the vertex-set V and the edge set defined as follows. Two vertices v and w are connected by a directed edge of length $\ell > 0$ if there is a centromeric read where w follows v at the distance ℓ units. If there are t different edges between v and w of the same length ℓ, we combine them into a single multiedge (v, w, ℓ) of multiplicity t. We further remove all multiedges with multiplicities below minCoverage = C*survivalRate*coverage (the default value $C = 0.4$ and, thus, minCoverage of four for our set of rare centromeric reads). Finally, we remove all conflicting parallel multiedges from the graph (multiedges connecting the same vertices but having different lengths) and all isolated vertices. The remaining vertices form a set of only 28,703 k-mers out of 391,361 initially constructed rare centromeric k-mers (next, we refer to the remaining k-mers as 'unique'). Even though some of the remaining k-mers turned out to be rare rather than unique (Fig. 1.7), they still appear to be valuable for assembly efforts.

1.6.6 Reconstructing the centromere

centroFlye$_{\text{HOR}}$ reconstructs the centromere sequence using an approach similar to the approach for resolving unbridged repeats in Flye [3] (Fig. 1.1 step (6)). Instead of using the divergent positions (as in Flye), it uses the unique centromeric k-mers to iteratively reconstruct the centromere.

Given a unit in a read, a bin of this unit is defined as the set of unique k-mers occurring in this unit. centroFlye represents each read as a sequence of bins (bin-sequence) that we refer to as readBin. Given two bins c and c', we define $shared(c, c')$ as the number of shared unique k-mers in these bins. Given two bin-sequences of the same length $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_n$, their score is computed as $\sum_{i=1}^{n} shared(x_i, y_i)$. Given bin-sequences x of length n and y of length m, their i-score ($1 \leq i \leq n$) is defined as the score between $x_i, \ldots, x_{\min(i+m,n)}$ and the prefix $y_1, \ldots, y_{\min(m,n-i+1)}$ of y. The maxScore between x and y is defined as the maximum of i-scores over all possible values of i and an alignment of x and y that achieves the maxScore value is

referred to as an optimal alignment.

An alignment of multiple reads (a contig) defines an alignment of their bin-sequences—a bin-sequence that we refer to as a bin-contig (for multiple aligned units in this alignment, their combined bin is defined as the union of all individual bins). centroFlye$_{\text{HOR}}$ supports an operation of optimally aligning a new read against a bin-contig and updating the bin-contig to include a newly added read.

Figure 1.1 step (6) illustrates the centroFlye$_{\text{HOR}}$ repeat resolution algorithm. First, all prefix reads are aligned based on their prefixes, represented as bin-sequences and combined into an initial bin-contig that starts at the unit position 0. Afterward, centroFlye$_{\text{HOR}}$ selects a still unaligned read with a highest-scoring optimal alignment against the bin-contig (in case of ties, the read with the rightmost starting position of the optimal alignment is selected). If the score of this alignment exceeds stopThreshold, it adds this read to the growing bin-contig, otherwise it stops the contig extension (the default value of stopThreshold is 10). In case centroFlye$_{\text{HOR}}$ incorporates nearly all reads in the growing contig-sequence (including suffix reads), we classify the centromere construction as successful and proceed to the polishing step (see Supplementary Note 4 in [35]). Otherwise, we apply the same centromere construction procedure but this time starting from suffix rather than prefix reads. It may happen that the prefix-based centromere construction stops before completing the centromere but the suffix-based construction generates the entire centromere. If both prefix-based and suffix-based centromere reconstructions stop, we generate the suffix and prefix contig-sequences that do not span the entire centromere.

Note that at this step we do not obtain the cenX sequence, but rather the bin-sequence of the centromere and the starting unit positions inside the cenX for all aligned centromere reads. We can compare centroFlye assembly to other assemblies by mapping read bin-sequences to bin-sequences for these assemblies (see Results).

1.6.7 Polishing the reconstructed centromere sequence

Using reported starting unit positions for all centromeric reads, centroFlye$_{\text{HOR}}$ separately polishes each HOR unit of the (yet unknown) centromere separately (Fig. 1.1 step (7)). In the case of cenX, it selects unit of the median length from corresponding reads, uses it as a template and applies four rounds of the polishing algorithm in Flye [46] (v.2.5). Polishing strategy implemented in TandemTools [41] can be used to further improve assembly quality.

Polishing results in a sequence of length 3,103,541 that includes 1,510 units and a single insertion of a LINE repeat. It contains 39,530 unique 19-mers; that is, 19-mers that appear only once in the assembly. Since ONT assemblies have high rates of homonucleotide indels, we further compressed all homonucleotide runs in the polished centromere, resulting in a compressed centromere that has only 26,333 unique 19-mers (Fig. 1.3, b).

1.6.8 Assembly of cen6

For cen6 assembly we used the Oxford Nanopore reads dataset generated by the T2T consortium (release 3) with 28,449,385 reads (367 Gb, 118× coverage) and N50 read length equal to 53 kb. This read-set includes 1,999,007 ultra-long reads (longer than 50 kb) that result in ˜62× coverage of the human genome. Next we describe various steps of centroFlyemono (Fig. 1.2).

1.6.9 Recruitment of centromeric reads from cen6

Using 18 cen6-specific monomers, we recruited 6,558 centromeric reads from cen6 (total length ˜268 Mb). Here 1,621 out of these 6,558 reads represent ultra-long reads that are most useful for centromere assembly (the longest read has a length of 530 kb).

1.6.10 Transforming reads into monoreads

NCRF [45] performs well in the case when most of a centromere is formed by canonical HORs (Fig. 1.2 step (1)). However, it generates a suboptimal decomposition into units when

a read contains abnormal HOR units. StringDecomposer [47] addresses this limitation of NCRF by partitioning reads into monomers rather than HORs and generating monoreads in the monomer alphabet. centroFlye$_{\text{mono}}$ uses StringDecomposer to transfrom the read-set Reads into a monoread-set Monoreads.

StringDecomposer fails to unambiguously translate some regions in a read into monomers due to locally high error rate in a read, a retrotransposon insertion or a still unknown monomer for a given chromosome. It represents each such region as a run of gap-symbols '?' repeated GapMultiplicity times, where GapMultiplicity is defined as the length of this region divided by the mean nucleotide length of the input monomers.

For cen6, the average (maximum) monoread length is 238 (3,195). The total length of all cen6 monoreads is 1,562,933. The total number of gap-symbols across all reads is 34,303 (2.2%) and the total number of gap-runs (contiguous sequences of gaps) is 7,375. This is a high error rate that makes the construction of the de Bruijn graph on mono-k-mers problematic (at least, for a large k) and necessitates the error correction step.

1.6.11 Error correction of monoreads

Supplementary Note 6 in [35] describes how centroFlye$_{\text{mono}}$ filters out poor-quality reads, trims the low quality ends of monoreads and splits monoreads that have a large fraction of gap-symbols (Fig. 1.2 step (2)). This step reduces the total number of gap-symbols across all monoreads to 5,989 (0.4%) and the total number of gap-runs to 3,193.

Supplementary Note 6 in [35] describes how centroFlye$_{\text{mono}}$ extracts HOR sequences from the remaining monoreads by constructing the de Bruijn graph on short and abundant mono-k-mers. On cen6 we extract two standard HORs: a mono-18-mer identical to D6Z1 (represented as ABCDEFGHIJKLMNOPQR in the monomer alphabet) and a mono-15-mer ABFGHIJKLMNOPQR that differs from D6Z1 by the deletion of a mono-3-mer CDE (Supplementary Note 6 in [35]). We say that a mono-t-mer fixes a run of t gap-symbols in a monoread if substituting this run with the mono-t-mer increases the number of standard HORs in the

monoread. For each run of t gap-symbols in a monoread, centroFlye$_{\text{mono}}$ attempts to find a mono-t-mer that fixes it and error corrects the monoread by substituting the gap-run with the found mono-t-mer. Such HOR-based error correction reduces the total number of gap-symbols across all monoreads to 2,505 (0.2%) and number of gap runs to 948.

1.6.12 Constructing iterative de Bruijn graph of monoreads

The choice of the k-mer size affects the construction of the de Bruijn graph (Fig. 1.2 step (4)). Smaller values of k collapse more repeats, making the graph more tangled. Larger values of k fail to detect overlaps between reads, making the graph more fragmented. Also, since monoreads have gaps, increasing k becomes problematic when one constructs the de Bruijn graph on mono-k-mers (Fig. 1.2 step (3)). 222,639 out of 249,119 (˜89%) mono-400-mers in reads are gap free after error correction.

The iterative de Bruijn graph [32], [33] incorporates information about mono-k-mers for multiple values of k into a single graph to reduce fragmentation in low-coverage regions and reduce repeat collapsing in high-coverage regions. While the de Bruijn graph DB$_k$(Reads) is constructed based on all k-mers in the read-set Reads, the iterative de Bruijn graph IDB$_k$(Reads) is recursively constructed based on a larger set of k-mers that extends all k-mers in Reads by adding all k-mers that are spelled by valid paths in the graph IDB$_{k-1}$(Reads). Next we define the concept of a valid path.

A vertex in a path is called an internal vertex if it is neither the initial nor the terminal vertex of this path. A vertex is called a 1-out (1-in) vertex if it has outdegree 1 (indegree 1). A path in a directed graph is called a 1-out (1-in) path if all its internal vertices are 1-out (1-in) vertices. For each edge e, there is a single longest 1-in path ending in e (referred to as $(e_{init}, \ldots, e)$) and a single longest 1-out path starting at e (referred to as $(e, \ldots, e_{term})$). We refer to the path $(e_{init}, \ldots, e, \ldots, e_{term})$ as the valid path for the edge e to reflect that each traversal visiting all edges of the graph contains each valid path as a subpath. We further refer to a string spelled by a valid path as a pseudoread and consider the set of all pseudoreads (one for each edge of the

graph). Even though there may be no reads containing a given pseudoread, one can safely add all pseudoreads to the set of real reads since each such pseudoread represents a substring of the genome [33].

Given a parameter k, the graph IDB_k(Monoreads) on mono-k-mers is recursively defined based on the graph IDB_{k-1}(Monoreads). A mono-k-mer is called frequent if it appears at least minMultiplicity times in Monoreads. We define minMultiplicity as Coverage * MonoKmerSurvivalRate * C with default $C = 0.2$. Specifically, we consider the set of all frequent mono-k-mers in Monoreads combining with all mono-k-mers in all pseudoreads and define IDB_k(Monoreads) as the de Bruijn graph constructed on these mono-k-mers. Each nonbranching path is compressed into a single edge. The length of this edge is defined as the length of the path and its coverage is defined as the median multiplicity of mono-k-mers (edges) in the path. To initialize the construction of the iterative de Bruijn graph, we define IDB_k(Monoreads) as the de Bruijn graph on all frequent mono-k-mers in Monoreads for a small value of k (the default $k = 100$) and iterate until a large value of k is denoted as K (the default $K = 400$). Supplementary Note 6 in [35] presents the graph IDB_{400}(Monoreads) for cen6.

1.6.13 Assembling monocentromere by scaffolding

An edge in the graph IDB_k(Monoreads) is called long if it spells a string of length at least MinLength (default value MinLength = 1,000) (Fig. 1.2 (5)). An edge is called unique if its coverage does not exceed the average coverage of the genome (Supplementary Note 7 in [35]). $\text{centroFlye}_{\text{mono}}$ attempts to scaffold long unique edges that are likely traversed just once by the genome.

$\text{centroFlye}_{\text{mono}}$ maintains an alignment of each centromeric read to the graph IDB_K(Monoreads) during its construction. A monoread connects a long unique edge e with a long unique edge e' if it starts in e and ends in e'. Each such monoread R represents a concatenate of $\text{prefix}_e(R)$ (the prefix of R mapping to e), $\text{middle}_{e,e'}(R)$ (an internal segment of R that does maps neither to e nor to e') and $\text{suffix}_{e'}(R)$ (the prefix of R mapping to e'). We define $\text{offset}_{e,e'}(R)$ as the length

of middle$_{e,e'}(R)$. We say that a long unique edge e precedes a long unique edge e' if there are at least MinConnection reads connecting e with e' and offsets of these reads are the same. We construct the scaffolding graph Scaffold$_K$(Monoreads) by maintaining all long unique edges in IDB$_K$(Monoreads), removing all other edges and adding scaffolding edges that connect long unique edges e and e' if e precedes e'. Each scaffolding edge is labeled by the consensus of strings middle$_{e,e'}(R)$ taken over all monoreads R that connect e and e'. Paths in the scaffolding graph are referred to as scaffolds. centroFlye$_{\text{mono}}$ further extends each scaffold from both sides by two reads that map to the prefix and suffix of each scaffold and extend this prefix and suffix as far as possible.

centroFlye$_{\text{mono}}$ generated two scaffolds of length 11,156 and 5,582 monomers. There are no centromeric reads that connect long and unique edges of these scaffolds. The region of cen6 in-between consists of a long recent segmental duplication that results in a complex traversal of IDB$_{400}$(Monoreads). However, the extensions of these long edges with reads (entering the duplication from both sides) are overlapping in a single vertex. We thus concluded that these two scaffolds are joined via this vertex, resulting in a single path that is traversed by cen6 (further validation is required to rule out a possibility that this vertex is not duplicated in this path).

1.6.14 Translating monocentromere to the nucleotide alphabet

centroFlye$_{\text{mono}}$ greedily partitions monocentromere into longest substrings such that no monomer is repeated more than once in each of substrings (we refer to these substrings as pseudo-units) (Fig. 1.2 step (6)). Alignments of each centromeric read to the graph IDB$_K$(Monoreads) indicate alignments of these reads to the monocentromere. centroFlye$_{\text{mono}}$ uses these alignments to separately polish each of 962 identified pseudo-units on cen6 separately. The assembly can be further polished using TandemMapper from the TandemTools package [41].

1.7 Supplementary Figures

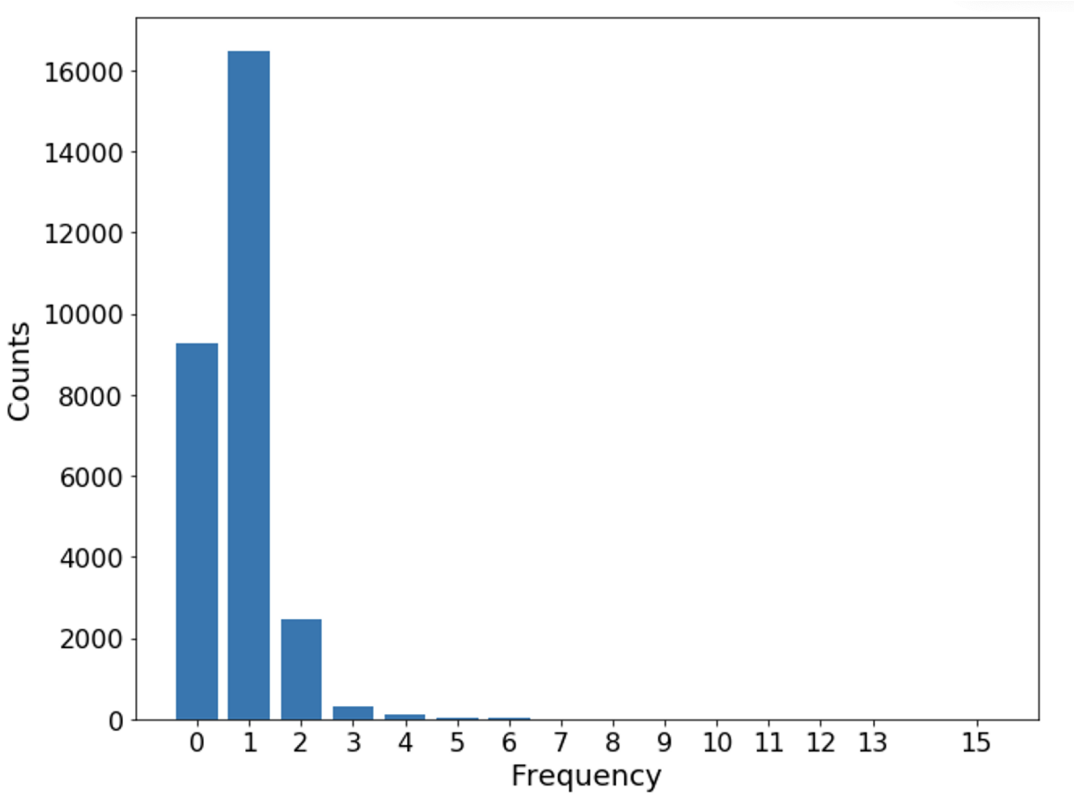


Figure 1.7. centroFlye cenX assembly contains 39,827 unique 19-mers. centroFlye recruited 28,703 of putative unique 19-mers from ONT reads. 16,488 (57.4%) of these 19-mers are also unique in the polished assembly. 9,267 (32.3%) are absent from the assembly and only 2,948 (10.3%) are repetitive in the assembly.

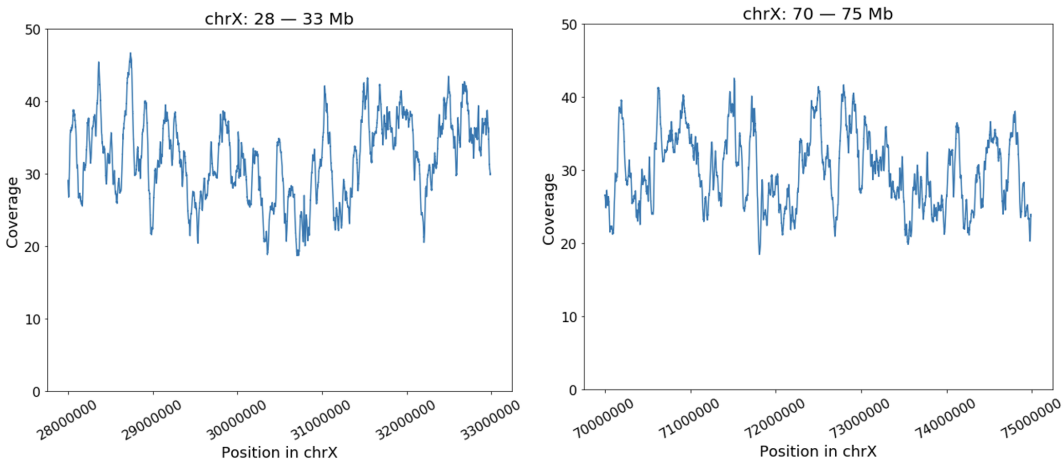


Figure 1.8. Range of coverage is [0, 50]. Moving average with window 10,000 in the region 28 - 33 Mb (**Left**) and the region 70 = 75 Mb (**Right**). Median coverage of the entire chrX in HG38 with ultra-long reads (longer than 50 kb) is 30x, 5-percentile is 19x and 95-percentile is 43x. The median length of regions with coverage higher than 43x or lower than 19x is 23,893 (mean is 10,237).

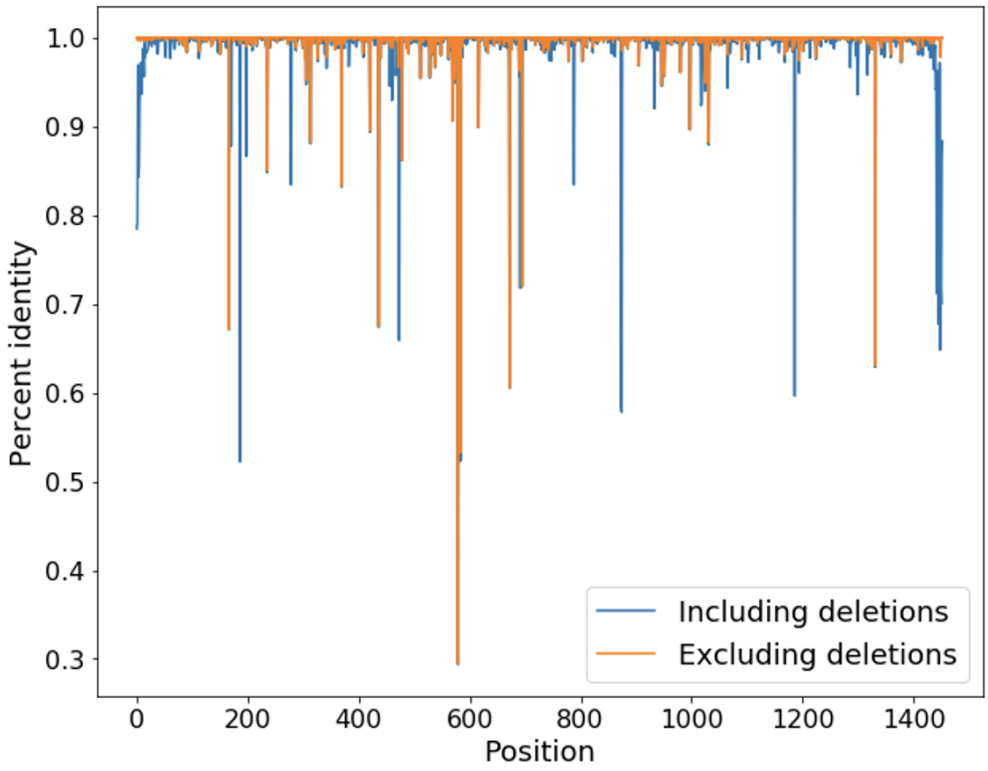


Figure 1.9. When considering deletions as mutations: mean percent identity = 98.8%, median = 99.8%. When ignoring deletions: mean percent identity = 99.5%, median = 100%.

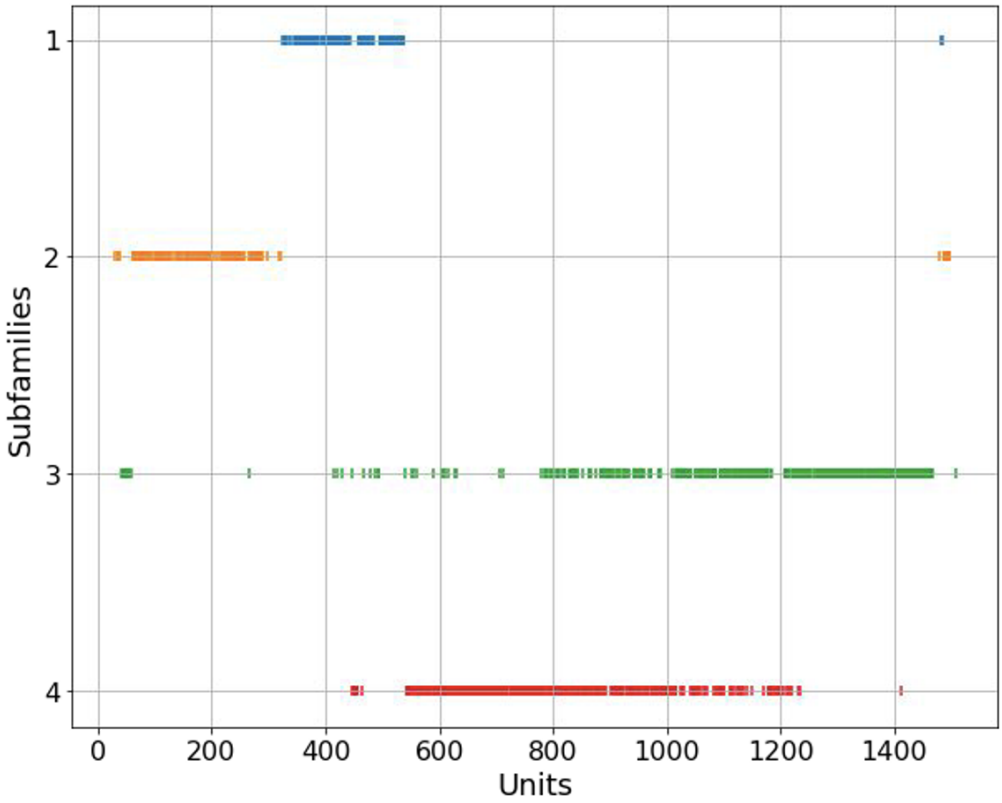


Figure 1.10. Red, green, orange, and blue subfamilies have 389, 429, 140, and 132 units. Red and green subfamilies are located in the middle and the second half of the assembly (units 500-1450), orange subfamily is concentrated in the beginning (units 1-300), and the blue subfamily is located between green and orange subfamilies (units 300-550).

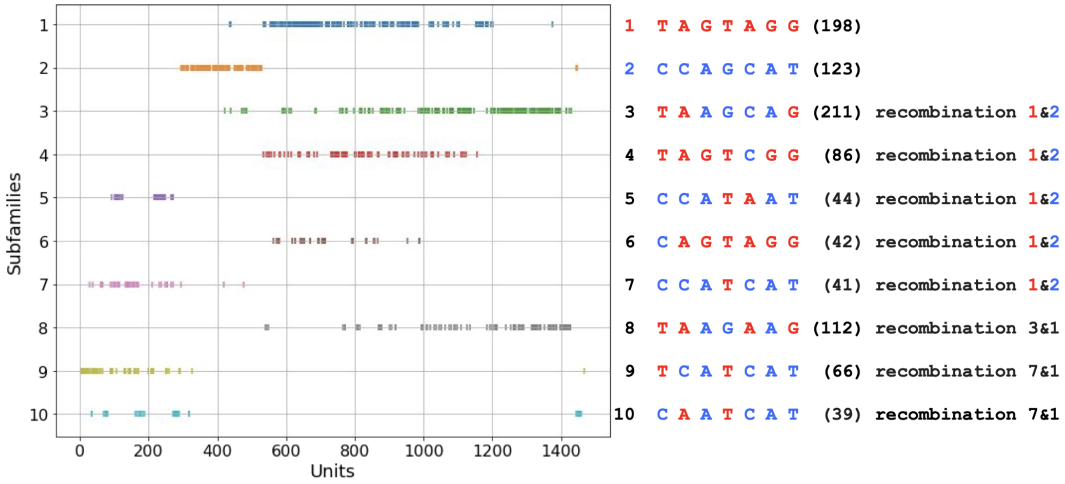


Figure 1.11. With exception of only eight units, each of seven divergent positions has only two nucleotides in each of the canonical HOR: T or C (position 167), A or C (position 437), G or A (position 580), T or G (position 584), A or C (position 673), G or A (position 695), and G or T (position 1332). We exclude these eight units from consideration and classify the remaining units into 27= 64 clusters depending on the string that they spell out in seven divergent positions. Two of these strings (the "red" string TAGTAGG and the "blue" string CCAGCAT) differ at each position. Ten largest clusters (shown above along with the number of units in each cluster) correspond to strings that represent recombination events between the red and the blue strings (string 3-7) or between the red string and a recombinant of red and blue strings (strings 8-10). The ancestral human cenX HOR is likely located within positions 500-1200 occupied by clusters 1, 4 and 6 with HORs that are most similar to the gorilla cenX HOR [48].

1.8 Data availability

centroFlye centromere 6 and X assemblies and all supporting data is available at Zenodo: https://doi.org/10.5281/zenodo.3897531. The ONT reads that were generated by the T2T consortium are deposited under accession number PRJNA559484.

1.9 Code availability

The codebase of the algorithm is available at https://github.com/seryrzu/centroFlye. The version of centroFlye that generates the assemblies described in the paper is in the branch: cF_NatBiotech_paper_Xv0.8.3-6v0.1.3. Jupyter notebooks for reproducing all figures in this study are provided in the Github repository https://github.com/seryrzu/centroFlye_paper_scripts.

1.10 Acknowledgements

We are indebted to I. Alexandrov, M. Kolmogorov, K. Miga and V. Shepelev for many insightful comments that improved centroFlye algorithm. We are grateful to A. Bankevich, A. Bzikadze, T. Dvorkina, A. Mikheenko, A. Phillippy, C. Wu and J. Yuan for helpful discussions and suggestions.

Chapter 1, in full, is a reprint of the material as it appears in Bzikadze, A. V., & Pevzner, P. A. (2020). Automated assembly of centromeres from ultra-long error-prone reads. *Nature Biotechnology, 38*(11), 1309-1316. The dissertation author is the primary developer of the centroFlye algorithm and the first author of this paper.

Chapter 2

TandemTools: mapping long reads and assessing/improving assembly quality in extra-long tandem repeats

2.1 Abstract

2.1.1 Motivation

Extra-long tandem repeats (ETRs) are widespread in eukaryotic genomes and play an important role in fundamental cellular processes, such as chromosome segregation. Although emerging long-read technologies have enabled ETR assemblies, the accuracy of such assemblies is difficult to evaluate since there are no tools for their quality assessment. Moreover, since the mapping of error-prone reads to ETRs remains an open problem, it is not clear how to polish draft ETR assemblies.

2.1.2 Results

To address these problems, we developed the TandemTools software that includes the TandemMapper tool for mapping reads to ETRs and the TandemQUAST tool for polishing ETR assemblies and their quality assessment. We demonstrate that TandemTools not only reveals errors in ETR assemblies but also improves the recently generated assemblies of human centromeres.

The codebase of TandemTools is available at https://github.com/ablab/TandemTools.

2.2 Introduction

Tandem repeats are formed by multiple consecutive nearly identical sequences that are often generated by unequal crossover [49]. The early DNA sequencing projects revealed that tandem repeats are abundant in eukaryotic genomes [50], [51]. Recent studies of tandem repeats revealed their role in various cellular processes and demonstrated that mutations in tandem repeats may lead to genetic disorders [14], [13], [52], [53].

We distinguish between extensively studied short tandem repeats [54], [55], [56] and extra-long tandem repeats (ETRs) that range in length from tens of thousands to millions of nucleotides. Centromeric and pericentromeric regions contain some of the longest ETRs that account for ˜3% of the human genome and span megabase-long regions [18]. Centromeres and pericentromeres represent the 'dark matter' of the human genome that evaded all attempts to sequence until recently and are the largest gaps in the reference human genome [27], [18]. The goal of the telomere-to-telomere (T2T) consortium is to generate a complete assembly of the human genome, including all centromeres and pericentromeres [18]. This effort recently resulted in assemblies of chromosomes X and Y [26], [34] but centromeres in other chromosomes are waiting to be assembled.

Human and primate centromeres are comprised of retrotransposon repeats and alpha-satellites, a DNA repeat based on a 171 bp monomer [57]. In humans and many primates, consecutive monomers are arranged tandemly into higher-order repeat (HOR) units [58]. The number of monomers and their order in a HOR are chromosome-specific. For example, the chromosome X HOR, referred to as DXZ1, consists of 12 monomers [59]. The monomer sequences are divided into five distinct monomer subtypes, denoted as A, B, C, D and E, where monomers from the same subtype are more closely related to each other than to monomers of other subtypes [59]. According to this classification, DXZ1 can be represented as C1D1E1

A1B1C2D2E2A2B2C3D3. For consistency with [35], we took the liberty to refer to the chromosome X HOR as ABCDEFGHIKL.

Emergence of long-read technologies, such as Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), have greatly altered the landscape of whole-genome sequencing. The development of long-read assemblers [1], [3], [5], [2], [46], [7] and hybrid assemblers that combine long and short reads [60], [61], [62] significantly increased the contiguity of assembled genomes compared to short-read assemblies. In addition, long reads contributed to successful semi-manual approaches for reconstructing human centromeres [26], [34]. The Flye assembler successfully resolves bridged tandem repeats that are spanned by long reads and even some unbridged tandem repeats that are not spanned by long reads [3]. The centroFlye assembler [35] was designed to automatically assemble unbridged ETRs, such as centromeres.

Various alternative strategies for ETR assembly and absence of the ground truth for benchmarking these assemblies raise the problem of their quality evaluation. Similar problems have been addressed by the short-read quality assessment tools for genome assemblies, such as GAGE [63] and QUAST [37], [64] as well as specialized quality assessment tools metaQUAST [65] and rnaQUAST [66]. However, these tools are based on known references and thus are not applicable to analyzing ETRs since their analysis requires reference-free approaches to evaluating assembly quality. At the same time, existing reference-free tools are based on analyzing gene content or mapping reads to the assembled sequences [67], [68], [69], [70] and are not applicable to ETRs either.

Existing reference-free assembly quality assessment approaches rely on sequence alignment tools [25], [71], [2], [72], [73] to accurately map reads to assemblies. However, our benchmarking revealed that these tools often fail in ETRs. The BWA-MEM tool [71], primarily designed for short-read mapping, incorrectly maps many long reads to ETRs. Minimap2 [72] incorrectly maps some long reads to ETRs (especially in regions with assembly errors) and thus is not well suited for ETR assembly quality evaluation. The recently developed Winnowmap tool [74] was specifically designed for mapping reads to repetitive genomic regions. However,

our benchmarking demonstrated that Winnowmap is limited with respect to detecting assembly errors: while it works well in the case of error-free assemblies, its accuracy deteriorates in the case of assembly errors (Table 2.1). We thus developed the TandemMapper tool that efficiently maps long error-prone reads to ETRs. TandemMapper not only enabled TandemQUAST development but also led to an improvement in ETR assemblies due to more accurate read mapping and subsequent polishing.

The initial attempt to evaluate the quality of ETR assemblies was centromere-specific [35] and has not resulted in a general quality assessment tool for ETR assemblies. Species- and chromosome-specific nature of centromeres prevents applications of the same approach to other ETRs. However, the common principles of centromere organization can be utilized for developing a universal assembly evaluation tool for ETRs.

Here, we present the TandemTools package that includes the TandemMapper tool for mapping reads to ETRs, and the TandemQUAST tool for evaluating and improving ETR assemblies. We used TandemTools and subsequent polishing to improve assemblies of the human centromere X (cenX) generated by both centroFlye [35] and the curated semi-manual approach [34]. These improvements suggest that TandemTools will become a useful tool for evaluating the quality and polishing of many assemblies since nearly all genomes have ETRs. We also applied TandemTools to the GAGE gene cluster at the human chromosome X [34] and to the assembly of the human centromere 8 generated by the recently developed HiCanu assembler [75] and demonstrated that it reveals assembly errors in these ETRs. The results are presented in Supplementary Appendices 'Analyzing ETRs in the GAGE locus at the human X chromosome' and 'TandemTools results on cen8 assembly' in [41].

TandemTools is open-source software that is freely available as a command-line utility on GitHub at https://github.com/ablab/TandemTools.

Table 2.1. *Note*: Minimap2 and Winnowmap were run using recommended parameters for mapping ONT reads (-cx map-ont). The best value for each column is indicated in bold. A read is considered correctly mapped if its starting position is within 100 bp from the read simulated position calculated for the longest read alignment (an alignment is elongated to both ends of a read). Only reads longer than 5 kb with alignments longer than 3 kb were considered. The total number of such reads in this read-set is 1180. Although minimap2 mapped 4 more reads than TandemMapper (1165 versus 1161), 3 out of these 4 reads came from the region of the deletion and 1 read was mapped incorrectly. The benchmarking was done on a server with Intel Xeon X7560 2.27 GHz CPUs using 16 threads.

	Correctly mapped reads	**Incorrectly mapped reads**	**# alignments extended through the deletion breakpoint**	**Running time (s)**	**Memory footprint (GB)**
TandemMapper (unique *k*-mers)	97.9% (1155)	**0.01% (1)**	**0**	511	5.2
TandemMapper (solid *k*-mers)	**98.3% (1160)**	**0.01% (1)**	**0**	590	5.6
minimap2	96.0% (1133)	2.7% (32)	58	357	5.8
Winnowmap	95.8% (1130)	2.8% (33)	58	**84**	**1.2**

2.3 Materials and methods

2.3.1 TandemTools input

As an input, TandemTools requires one or several ETR assemblies and the set of long reads (PacBio continuous long reads or ONT) that contributed to these assemblies. Additionally, error-prone long reads can be complemented by accurate long reads, such as PacBio high-fidelity (HiFi) reads. We do not consider short Illumina reads since it is nearly impossible to unambiguously map them to ETRs.

2.3.2 TandemTools modules

TandemTools consists of the read-mapping module that aligns reads to the assembly (TandemMapper), the polishing module for improving the assembly quality based on the identified read alignments and the quality assessment module (TandemQUAST). TandemQUAST uses *general metrics* for evaluating ETRs of any kind and *centromeric metrics* designed specially to account for the HOR structure of centromeric ETR.

2.3.3 Selection of *k*-mers

Selecting solid *k*-mers in ETRs

Most long-read mapping algorithms are based on *minimizers* [26], [2], [72], *k*-mers that are chosen as the anchors for the read mapping. However, mapping a long read to an ETR is a non-trivial problem since minimizers are expected to be reduced in numbers and irregularly arranged due to local expansions of identical tandem repeats. [35] used unique *k*-mers (that appear just once in the assembly) to improve read mapping to ETRs.

The density of unique *k*-mers may significantly vary along an assembly (Fig. 2.1), leading to drops in coverage or incorrect mappings in some regions. To address this problem, TandemMapper uses *rare* *k*-mers that appear less than *MaxOccurrences* times in the assembly. To obtain uniform *k*-mer density, we compute *MaxOccurrences* as the assembly length divided by 100 kb. Figure 2.1 illustrates that the density of rare *k*-mers is significantly larger than the density of unique *k*-mers, thus providing more 'signposts' for read mapping.

Since ETR assemblies can be error-prone, some rare *k*-mers may represent assembly errors rather than low-frequency *k*-mers in the genome. To filter out such rare *k*-mers, we analyze their frequencies in the read-set. We assume that a *k*-mer from an assembly was erroneously classified as rare if it has an unusually low frequency (lower than *MinFrequency*) or an unusually high frequency (higher than *MaxFrequency*) in reads. The *MinFrequency* (*MaxFrequency*) threshold is defined as a fifth (95th) quantile of *k*-mer frequencies in the read-set. We, thus classify a rare *k*-mer as solid if it occurs in reads at least *MinFrequency* and at most *MaxFrequency* times.

The *k*-mer selection procedure can be affected by the fact that ETRs may harbor various transposable elements (TEs), such as LINE repeats, Alu repeats, etc. Even a single copy of a TE within an ETR is likely to contain many solid *k*-mers that may affect the mapping accuracy and complicate further analysis. To minimize the influence of TEs on the choice of solid *k*-mers, we set the *MaxKmers* limit on the maximum number of solid *k*-mers that can be selected in

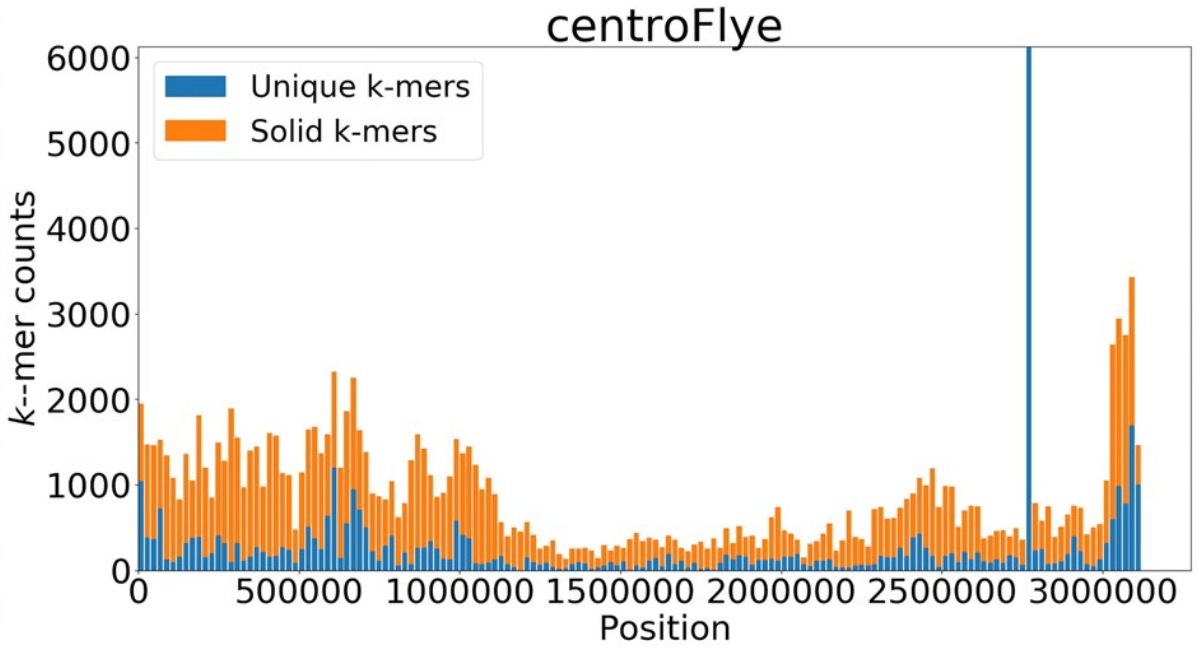


Figure 2.1. Distribution of unique and solid 19-mers along the cenX assembly of the CHM13 cell line constructed by centroFlye. Each bar shows the number of unique (solid) 19-mers in a bin of length 20 kb. The total number of unique (solid) 19-mers is 39 530 (57 318). The peak at ˜2750 kb corresponds to a LINE element and contains 6128 unique 19-mers and only 1499 solid 19-mers after filtration.

each window of a fixed length L (default value $L = 1000$ bp). Given an array *KmerDensity* of the number of k-mers in each window of length L in the assembly, *MaxKmers* is calculated as median(*KmerDensity*) + 2σ(*KmerDensity*), where σ is the SD within *KmerDensity*. Thus, if the number of solid k-mers in a window exceeds the threshold, we randomly select *MaxKmers* of them.

Compatible k-mers

The TandemMapper algorithm is inspired by the minimap2 [72] and Flye mappers [3], [46]. As solid k-mers are not necessarily unique in the assembly, we consider each occurrence of each solid k-mer separately.

Let a_R and b_R (a_A and b_A) be occurrences of solid k-mers a and b in the read R (assembly A). To make a_R and b_R uniquely defined for each read, we limit attention to solid k-mers that appear exactly once in this read. Note that, while a_R and b_R are uniquely defined, there may be multiple choices for a_A and b_A. We define $d(a_R, b_R)$ and $d(a_A, b_A)$ as distances between a and b in R and A, respectively.

We refer to the pair of a_A and a_R (b_A and b_R) as a match a_M (b_M) and define:

$$distance(a_M, b_M) = \min\{d(a_R, b_R), d(a_A, b_A)\},$$
$$diff(a_M, b_M) = |d(a_R, b_R) - d(a_A, b_A)|,$$
$$penalty(a_M, b_M) = diff(a_M, b_M)/distance(a_M, b_M).$$

To assess the distribution of differences between distances in reads and the assembly, we collect all penalties taken over all consecutive non-overlapping unique k-mers a and b in all reads where these k-mers appear once into the *Penalties* array. We define *distortion C* as *median*(*Penalties*) + *IQR(Penalties)*, where IQR stands for the interquartile range.

In addition, we define *MissedKmers*(a_M, b_M) as the number of solid k-mers in assembly

A between a_A and b_A. We call a_M and b_M compatible if $distance(a_M, b_M) < maxDistance$ (*maxDistance* is defined as the largest distance between two consecutive unique k-mers in the assembly), $MissedKmers(a_M, b_M) < maxMissed$ (the default value *maxMissed* = 500) and $diff(a_M, b_M) < C \cdot distance(a_M, b_M)$, where C is the distortion.

2.3.4 Tandemmapper module

Given a read, we define a directed weighted *compatibility graph* with a vertex-set equal to the set of all matches of solid k-mers between R and A. We connect vertices a_M and b_M by an edge if (i) a precedes b in R and (ii) a_M and b_M are compatible. We further define the weight of this edge as $premium|penalty(a_M, b_M)$, where *premium* is a constant selected to optimize the number of correctly mapped reads (default value $premium = 0.1$). A chain between a read R and an assembly A is defined as the longest path in the compatibility graph. Note, that since all considered solid k-mers appear just once in R, no solid k-mer can be present in the chain more than once.

A chain for a given read can be used to map this read to the assembly. TandemMapper finds a chain for each read using dynamic programing, filters out short chains (shorter than 3 kb in length or containing less than 20 solid k-mers) and constructs the corresponding nucleotide-level alignments within the derived chain boundaries for each remaining chain. Table 2.1 illustrates that TandemMapper improves on other long-read mapping tools in ETRs.

2.3.5 Polishing module

Due to the high error rate in reads, most long-read assemblers have a polishing step to improve base-calling accuracy of the assembly [76], [46], [77], [78]. However, [34] demonstrated that standard polishing tools may even decrease the assembly quality in ETRs due to incorrect and ambiguous read alignments against the assembly. On the other hand, [34] demonstrated that the marker-assisted read mapping (based on unique k-mers) significantly improves accuracy of ETR assemblies. TandemQUAST uses read alignments generated by TandemMapper as an input

for a modified Flye polishing module [3], [46]. We demonstrate that this polishing procedure fixes erroneous deletions and base-calling errors.

2.3.6 Quality assessment module (TandemQUAST)

To evaluate the assembly quality and reveal possible errors, we developed two *general* metrics (indel-based and k-mer-based) and a *centromeric* metric (monomer-based) that we describe below. Former metrics are applicable to any ETRs and the latter metric is applicable to centromeric ETRs only.

Indel-based metrics.

ETR assemblies are prone to large-scale deletions and duplications that lead to misassembly breakpoints. QUAST [37] defines a misassembly breakpoint based on differences between an assembly and a reference genome. In contrast, since the reference is not available, TandemQUAST detects breakpoints based on abnormalities in the read coverage. Below we describe the coverage metric and the breakpoint metric and use them to reveal putative breakpoints.

Coverage metric. Assembly errors may affect the coverage near the assembly breakpoints. TandemQUAST uses the read alignments (truncated with respect to their longest chains) to construct the coverage plot and reveal regions with abnormal coverage that may point to assembly errors (Fig. 2.3).

Breakpoint metric. Since long-read assemblers often fail to distinguish various repeat copies and erroneously collapse repetitive regions, indels represent the most frequent assembly errors in ETRs. The breakpoint metric was designed specifically to detect indels based on the analysis of mapped reads. In case, an assembly contains a breakpoint caused by a long indel, longest chains for the majority of reads spanning this indel breakpoint cannot be extended through this indel due to a substantial discrepancy in distances between solid k-mers in reads spanning this breakpoint and the assembly. Thus, if longest chains for many reads start or end in

a certain region, this region may contain an assembly breakpoint. However, stochastic differences in coverage and various biases may also result in drops or peaks in read coverage. Our goal is to distinguish these cases and reveal assembly breakpoints.

A chain for a read R defines its partitioning into $prefix(R)$, $middle(R)$ and $suffix(R)$, where $middle(R)$ is the mapped part of a read that starts (ends) at the first (last) k-mer in the chain. The region in the assembly corresponding to $middle(R)$ is referred to as a chain-segment. We also define an elongated chain-segment as a chain-segment extended by $|prefix(R)|$ and $|suffix(R)|$ nucleotides in the beginning and the end, respectively.

Given a solid k-mer $Kmer$, we define $breaks(Kmer)$ [$breaks^+(Kmer)$] as the number of chain-segments (elongated chain-segments) starting or ending in this k-mer (over all reads). We also define $number(Kmer)$ [$number^+(Kmer)$] as the number of chain-segments (elongated chain-segments) containing this k-mer. Finally, we define

$$breakpointRatio(Kmer) = breaks(Kmer)/number(Kmer),$$

$$breakpointRatio^+(Kmer) = breaks^+(Kmer)/number^+(Kmer).$$

While drops in values of $breakpointRatio$ usually correspond to poorly covered regions, peaks in values may reveal breakpoints in the assembly. We expect that regions, where $breakpointRatio(Kmer)$ has significantly higher values than $breakpointRatio^+(Kmer)$, contain assembly breakpoints because the longest chains for many reads were not extended through this region (Fig. 2.3).

K-mer-based metrics

In contrast to the TandemMapper tool (that considers k-mers that appear more than once in the assembly), the k-mer-based metrics need a reliable set of k-mers that appear just once in the assembly. We, thus filter out solid k-mers that occur more than once in the assembly or more than once in a single read and refer to the rest as unique solid k-mers.

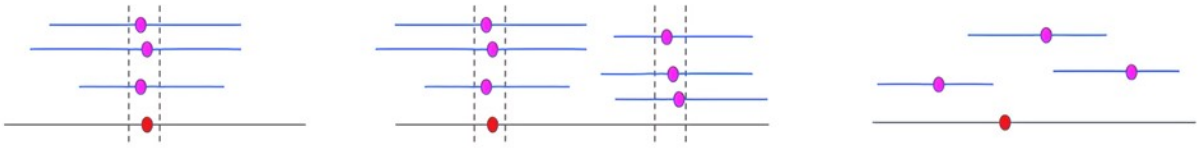


Figure 2.2. Coordinates of unique solid *k*-mers in the assembly and reads. Purple and red dots represent *k*-mer position in reads (shown as blue lines) and in the assembly (shown as a gray line), respectively. Clumps are flanked by vertical lines. (left) *k*-mers forming a single clump, (middle) *k*-mers forming multiple clumps in different parts of the assembly and (right) *k*-mers that do not form clumps (spurious *k*-mers).

Table 2.2. Distribution of different types of unique solid *k*-mers in the T2T4, T2T4$_{\text{polish}}$, T2T7, centroFlye and centroFlye$_{\text{polish}}$ assemblies. *Note*: Assemblies do not utilize information derived from accurate PacBio HiFi reads.

	T2T4	**T2T4$_{\text{polish}}$**	**T2T7**	**centroFlye**	**centroFlye$_{\text{polish}}$**
Single clump	13 130 (75%)	**15 158 (96%)**	**16 114 (97%)**	16 550 (96%)	15 858 (97%)
Multiple clumps	**1058 (6%)**	**524 (3%)**	**294 (2%)**	422 (2%)	396 (2%)
No clumps	3217 (17%)	197 (1%)	237 (1%)	302 (2%)	180 (1%)

After constructing read alignments, TandemQUAST finds where a unique solid *k*-mer in a read maps to the assembly and calculates coordinates of all found alignments across all reads containing this *k*-mer. Afterward, it clusters these coordinates (for a given unique solid *k*-mer) if they are located within *MaxClumpDistance* from each other (default value *MaxClumpDistance* = 1 kb). After single linkage clustering, we define a cluster as a clump if it contains more than *MinClumpSize* elements (default value $MinClumpSize = 2$). Ideally, all occurrences of a unique solid *k*-mer should form a single clump. We divide all *k*-mers having at least MinClumpSize occurrences in reads into three groups: a single clump, multiple clumps and spurious *k*-mers that do not form clumps (Fig. 2.2).

TandemQUAST reports absolute and relative abundance of such *k*-mers and generates a plot showing their distribution (Table 2.2 and 2.4). Multiple clumps or spurious *k*-mers appearing along the entire assembly may point to poor base-calling quality of this assembly. Multiple clumps or spurious *k*-mers appearing in a certain region of an assembly reflect either a poor base-calling quality in these regions or collapsed duplications with subsequent 'consensus' polishing with reads from both copies.

In the case when a complementary set of accurate PacBio HiFi reads is available, TandemQUAST compares *k*-mer frequencies in the assembly and the HiFi reads. If the assembly contains *k*-mers that do not occur in HiFi reads or frequent *k*-mers from reads have a low frequency or are even absent in the assembly, it is likely that the assembly requires additional polishing (Supplementary Fig. S8 in [41]).

Centromeric metrics.

The additional set of metrics takes into account the centromere organization into monomers and HOR units. When a set of specific monomer sequences is known, TandemQUAST can analyze the assembly using the monomer-based metric described below and the unit-based statistic described in Supplementary Appendix 'Unit-based statistic' in [41].

Centromere assemblies may include difficult-to-detect indels of multiple monomers. In case monomer sequences are known, TandemQUAST attempts to detect discrepancies between reads and the assembly at the monomer level. The assembled centromere and all reads are aligned to the provided monomer sequences and are subsequently translated into the monomer alphabet using the StringDecomposer tool [47], resulting in a *monocentromere* and *monoreads*.

For each monomer $ReadMonomer$ in each monoread, TandemQUAST uses nucleotide-based read alignments to identify the starting nucleotide position of $ReadMonomer$ in the monocentromere [referred to as $Start(ReadMonomer)$]. In case, $ReadMonomer$ is aligned against a deletion in the monocentromere, $Start(ReadMonomer)$ is recursively defined as $Start(NextReadMonomer)$, where $NextReadMonomer$ is the next monomer in the monoread. For each monomer $CenMonomer$ in the monocentromere, we define $Start(CenMonomer)$ as the starting position of this monomer in the centromere. We say that a monomer in a read ($ReadMonomer$) and a monomer in a centromere ($CenMonomer$) are co-located if

$$|Start(ReadMonomer) - Start(CenMonomer)| < MaxStartDistance$$

(the default value $MaxStartDistance = 50$ bp).

For each monomer *CenMonomer* in the monocentromere, TandemQUAST constructs the set $ReadMonomers(CenMonomer)$ of all monomers in reads that are co-located with this monomer. For an error-free assembly, we expect that the vast majority of monomers in $ReadMonomers(CenMonomer)$ coincide with *CenMonomer*, i.e. the ratio of *CenMonomer* in $ReadMonomers(CenMonomer)$ is high. If this ratio [denoted as $Ratio(CenMonomer)$] is below a threshold $MinRatio$ (the default value $MinRatio = 0.8$), the assembly is likely to have an error (Supplementary Fig. S1 in [41]). However, in the case of heterozygous monomers, this ratio is close to 0.5 as roughly half of the reads support (do not support) the monomer.

Although individual monomers may significantly vary in sequence, their length is fairly conserved within species that have alpha-satellites [79], [80]. Thus, variations in monomer length across the centromere in such species may point to flaws in the assembly. Using StringDecomposer output, TandemQUAST generates an interactive HTML-page that provides a general monomer-level overview of the assembly and demonstrates the distribution of monomer lengths (Fig. 2.5).

Comparison of various ETR assemblies

TandemQUAST performs pairwise comparison for each pair of analyzed assemblies using the *bi-mapping* plot and the *discordance* test.

A bi-mapping plot (Supplementary Fig. S2 in [41]) provides an overview of read alignments from the perspective of both assemblies. Each read aligned to both assemblies represents a dot with its starting mapping positions in two assemblies as the x- and y-coordinates. Positions of read alignments for two assemblies can be compared to reveal structural discrepancies between them.

The discordance test was introduced in [35] for comparing two assemblies. Supplementary Appendix 'Discordance test' in [41] describes its implementation in TandemQUAST.

2.4 Results

2.4.1 Simulated assembly

To benchmark TandemTools, we simulated an ETR of length ˜1.03 Mbp, which is a concatenation of 500 randomly mutated copies of the consensus HOR sequence on chromosome X (DXZ1) that diverge from the consensus sequence by 1% (substitutions only). Afterward, we simulated 1200 reads from this ETR using NanoSim [36] trained on the real ONT dataset enriched for ultra-long reads (longer than 50 kb) generated by the T2T consortium [34]. We refer to the centroFlye assembly of these reads as simulated. We further introduced various artificial errors (described below) into the simulated assembly and ran TandemTools. An additional example of TandemTools performance on a centromere with more complex structure is presented in Supplementary Appendix 'TandemTools results on the simulated datasets (D6Z1)' in [41].

Benchmarking TandemMapper, minimap2 and Winnowmap

We compared TandemMapper with minimap2, the widely used long-read mapper that achieves excellent results outside repeated regions, and Winnowmap [74] that is designed specifically for mapping reads to repetitive genomic regions. To analyze how these tools handle assembly errors, we generated simulateddel assembly by introducing an artificial deletion of length 10 kb in the simulated assembly at position 400 kb.

We benchmarked mapping tools by aligning simulated reads to the $simulated_{del}$ assembly and comparing their known exact positions in the assembly to the inferred positions 2.1. TandemMapper correctly stopped all read alignments at the breakpoint of this deletion, while minimap2 and Winnowmap erroneously extended alignments through this breakpoint due to the highly repetitive sequence of the ETR. Using solid k-mers instead of unique k-mers slightly increased the number of correctly mapped reads even in an easy case of the simulated assembly with the uniform density of unique k-mers.

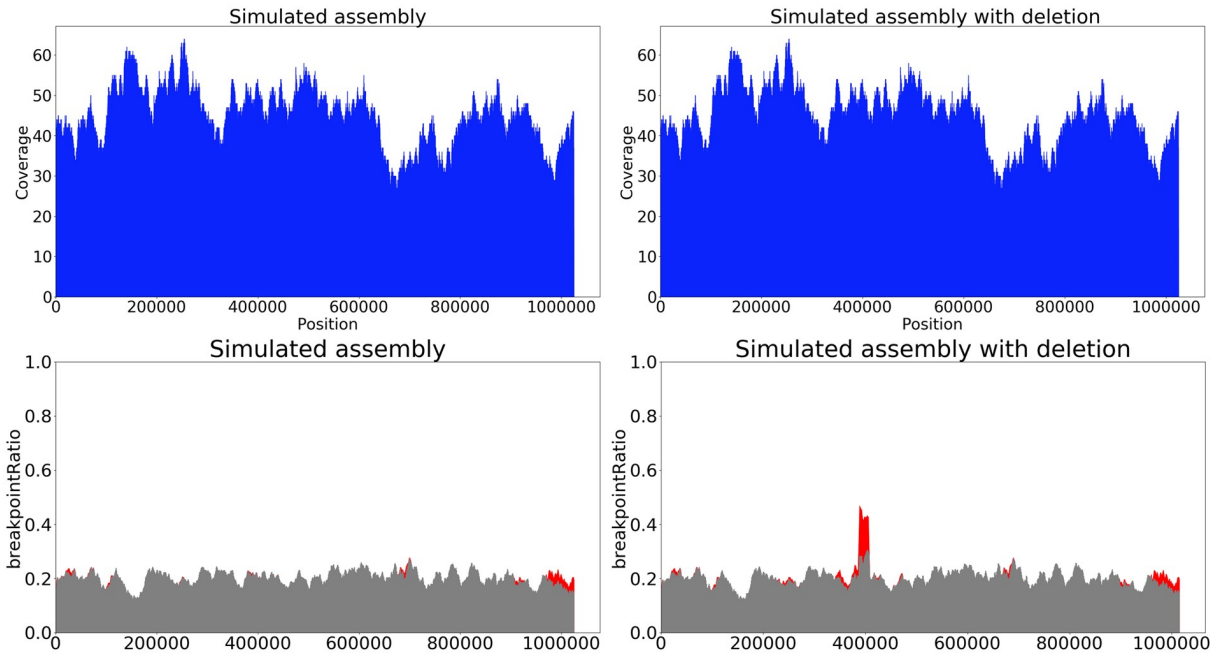


Figure 2.3. Coverage (top) and breakpoint (bottom) metrics for simulated (left) and $simulated_{del}$ (right) assemblies. The coverage plot does not show a significant drop at the point of the deletion but the breakpoint plot reveals peak at the position of the deletion (400 kb). The red plot is based on the $breakpointRatio(Kmer)$ values, the gray plot is based on the $breakpointRatio^{+}(Kmer)$ values.

Indel-based metrics

To analyze how these metrics capture breakpoints, we used the simulateddel assembly (Fig. 2.3). Although the coverage plot does not show a significant drop at the point of the deletion, the breakpoint plot reveals a peak at the position of the deletion (400 kb).

k-mer-based metrics

To benchmark metrics evaluating the base-calling accuracy of an assembly, we introduced 10 000 (˜1% of the sequence length) random single-nucleotide substitutions in the *simulated* assembly (we refer to this assembly as $simulated_{mut}$). TandemQUAST reports the number of each group of unique solid *k*-mers and their distribution in the assembly (Fig. 2.4). The percent of unique solid *k*-mers forming a single clump decreased from 91% in the *simulated* assembly to 74% in the $simulated_{mut}$ assembly, mostly due to the increased number of spurious *k*-mers.

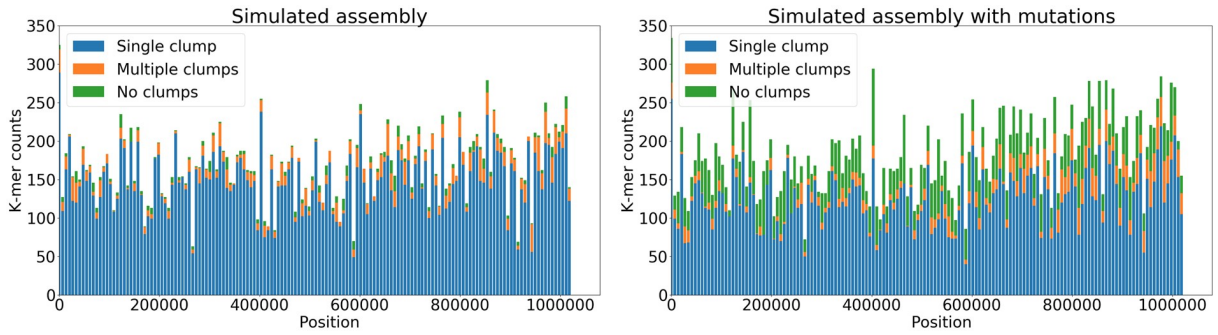


Figure 2.4. Distribution of different types of unique solid *k*-mers in the simulated (left) and $simulated_{mut}$ (right) assemblies. Each bar shows the number of different types of *k*-mers in a bin of length 5 kb.

Centromeric metrics

In order to illustrate the monomer-based metric and the unit-based statistic, we generated the $simulated_{del\_monomer}$ assembly by introducing a deletion of three consecutive monomers in the simulated assembly at position 226 kb. The results are presented in Supplementary Appendices 'TandemTools results on the simulated datasets (DXZ1)' and 'Unit-based statistic' in [41].

In addition, we demonstrated how these metrics might be affected by the assembly quality. Figure 2.5 shows that most monomers have conserved length across the assembly. However, the first monomer A and the last monomer L show surprising variability in length, suggesting that the accuracy of the simulated assembly deteriorates at the ends of HOR units due to imperfect polishing. This imperfect polishing is caused by limitations of the existing read-mapping tools in ETRs, forcing centroFlye to perform separate polishing for each HOR. Since the polishing procedure [46] is known to have limitations in the very beginning/end of each segment subjected to polishing, the beginning of the first (A) and the end of the last (L) monomers in each HOR can be cut off in a polished assembly. Just a single round of polishing with TandemQUAST resulted in the $simulated_{polish}$ assembly with an increased assembly length (by ˜4 kb) and corrected sequences of the first and the last monomers along the entire assembly (Fig. 5).

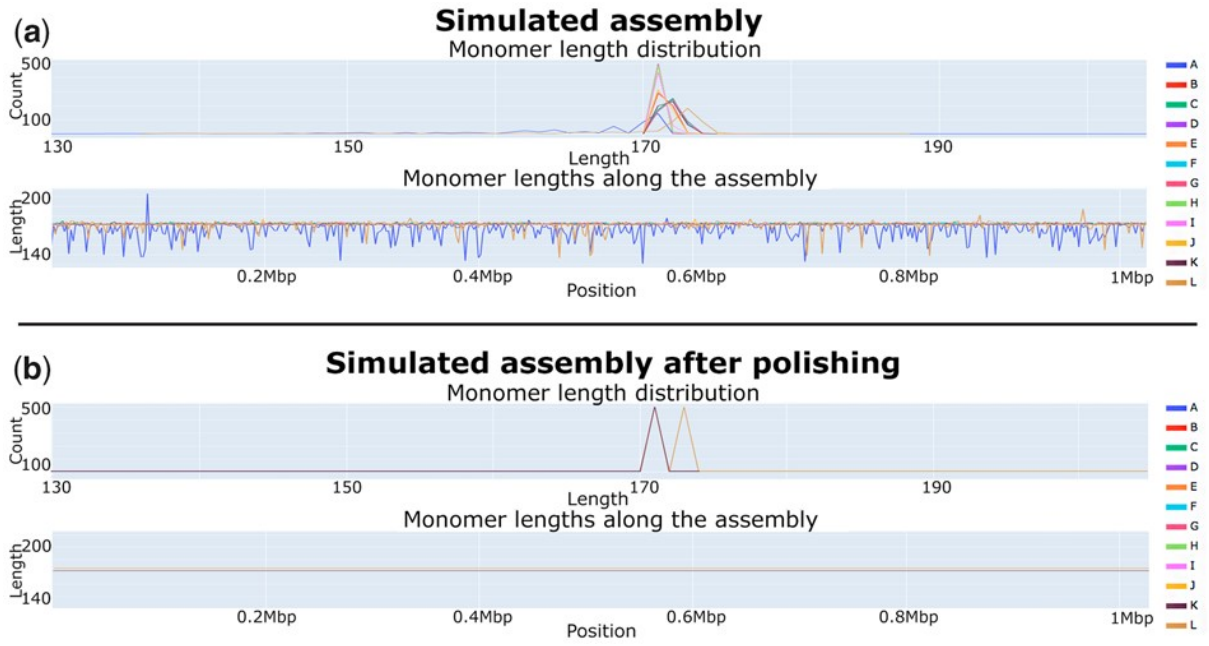


Figure 2.5. Monomer length distribution for the simulated (**a**) and *simulated*$_{polish}$ (**b**) assemblies. Monomer sequences forming a consensus DXZ1* sequence, derived in [35], were used for analysis. In the simulated assembly, the length of the A-monomers varies from 131 to 203 bp (mean 165 bp) and the length of the L-monomers varies from 137 to 187 bp (mean 171 bp). In the *simulated*$_{polish}$ assembly, the length of all A-monomers (L-monomers) is equal to 171 (173) bp. Since all monomers, except for L, have lengths 171 bp after polishing, they all are represented by the color corresponding to the K-monomer.

2.5 Analysis of cenX assemblies

We analyzed the following cenX assemblies: the T2T consortium assembly v0.4 (T2T4), v0.7 (T2T7) [34] and centroFlye v0.8.3 assembly (centroFlye) [35]. Note that, the T2T4 assembly is an interim version that was not polished with the marker-assisted methods described in [34]. We added it to the comparison to show how TandemQUAST analyzes unpolished assemblies. The T2T7 version was first semi-manually assembled and further improved based on centroFlye assembly as described in [34]The T2T7 and centroFlye assemblies were additionally polished using ONT reads.

We also applied our polishing method to the T2T4 and centroFlye assemblies (resulting in $T2T4_{polish}$ and $centroFlye_{polish}$ assemblies) to demonstrate how TandemQUAST improves assemblies.

Selecting solid *k*-mers in ETRs

The centroFlye assembly of the cenX has 39 530 unique 19-mers distributed across the 3.1 Mbp of the cenX length, with the largest distance between consecutive unique 19-mers =30 kb [35]. The number of rare 19-mers using *MaxOccurrences* = 30 is 66 785 (Fig. 2.1).

Applying the filtration of *k*-mers by *MinFrequency* and *MaxFrequency* removes 5801 out of 66 785 rare *k*-mers, leaving 60 984 solid 19-mers. Comparison with PacBio HiFi reads generated from the same cell line [8] revealed that 4844 of 5801 filtered out 19-mers are absent in the HiFi read-set or, on the contrary, have a very high frequency (higher than a frequency of 95% of 19-mers in the read-set). Applying the additional filtration by MaxKmers further reduces the number of solid 19-mers in the assembly from 60,984 to 55,173.

Indel-based metrics

Figure 2.6 illustrates that all assemblies have slightly lower read coverage at the center of the centromere at ˜1300–1600 kb that has a low concentration of unique *k*-mers (Supplementary Fig. S6 in [41]).

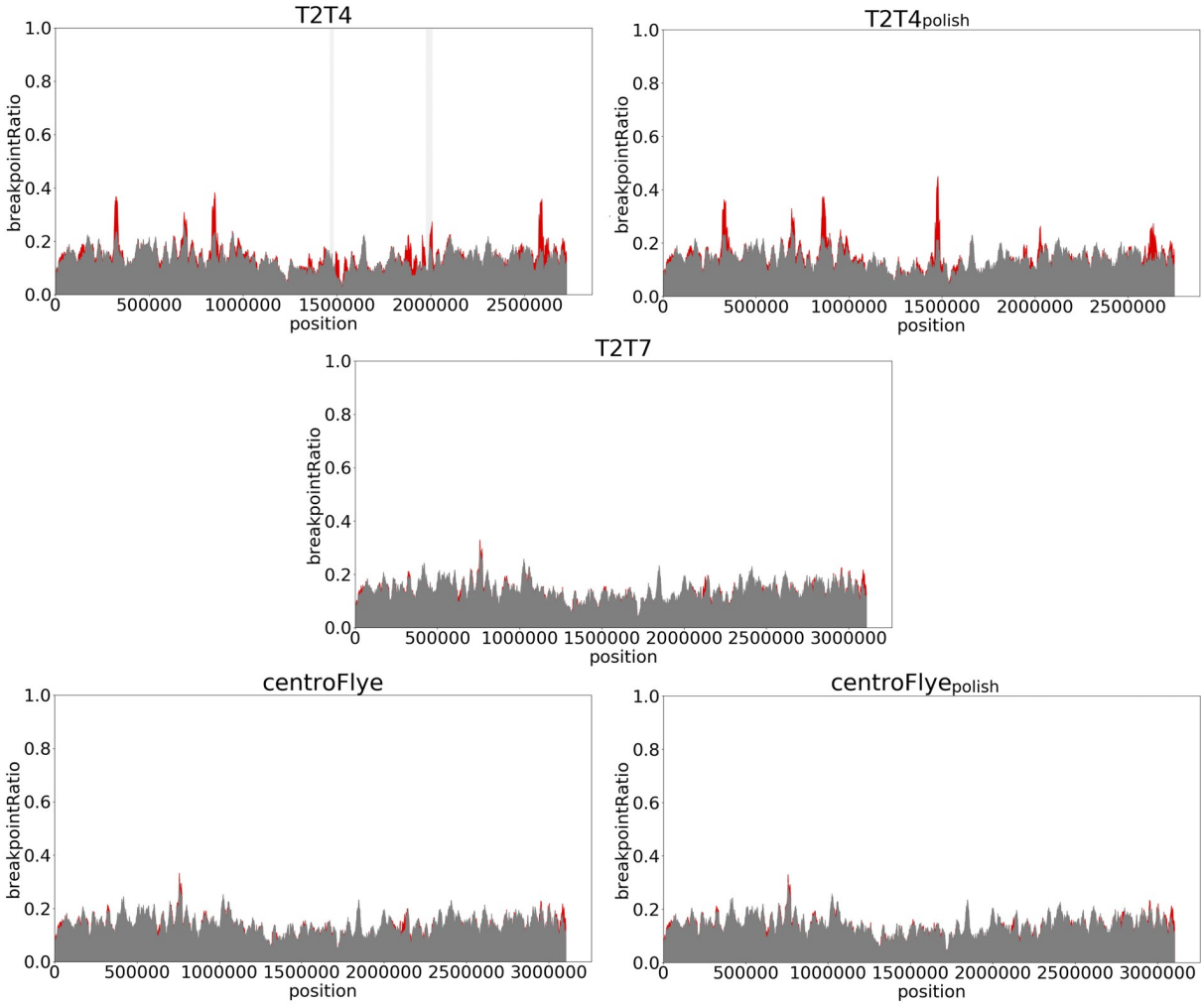


Figure 2.6. The breakpoint metric for the T2T4, $\text{T2T4}_{\text{polish}}$, T2T7, centroFlye and $\text{centroFlye}_{\text{polish}}$ assemblies. The red and the gray plot are based on the $breakpointRatio(Kmer)$ and $breakpointRatio^{+}(Kmer)$ values, respectively. The vertical light gray bands represent regions with low coverage ($< 10\times$). Discrepancies in these regions do not necessarily reflect flaws in an assembly.

Low base-calling accuracy of an assembly can prevent chain extension in TandemMapper. As a result, the longest chains for many reads may end in a poorly polished region, causing an increase in *breakpointRatio* values. Thus, to verify breakpoints found in the T2T4 assembly, we compared them to the $T2T4_{polish}$ assembly. Both assemblies have peaks in breakpointRatio values at 270, 800, 1500, 2000 and 2500 kb that correlate with the bi-mapping plot (Supplementary Fig. S7 in [64]). The breakpoint metric for centroFlye and T2T7 assemblies are generally consistent between $breakpointRatio(Kmer)$ and $breakpointRatio^{+}(Kmer)$ values, suggesting that these assemblies do not have large indels and rearrangements.

k-mer-based metrics

Supplementary Figure S6 in [64] and Table 2.2 show the distribution of different types of unique solid *k*-mers across the assemblies. The T2T4 assembly has a high number of spurious *k*-mers as expected for an unpolished assembly, while $T2T4_{polish}$ demonstrates significant improvement in base-calling accuracy across the assembly. The high percentage (92–96%) of *k*-mers forming a single clump in the T2T7 and centroFlye assemblies suggest a high base-level quality in these assemblies.

In addition, we compared *k*-mer frequencies in assemblies and in accurate PacBio HiFi reads generated from the same cell line CHM13 [8]. The number of *k*-mers that do not occur in the HiFi read-set was the highest in the unpolished T2T4 assembly (223,579) and the lowest (842) in the T2T7 assembly (Supplementary Fig. S4 in [35]).

Monomer metrics

Figure 2.7 presents the monomer length distribution across various assemblies. The T2T7 and centroFlye assemblies have a few unusually short (145–146 bp) A-monomers at ˜1 Mbp. We checked these monomers further and confirmed that they are supported by reads. Besides that, the T2T7 assembly has very conserved monomer lengths except for a few monomers at ˜2.15 Mbp.

In the centroFlye assembly, L-monomers significantly vary in length as in the simulated

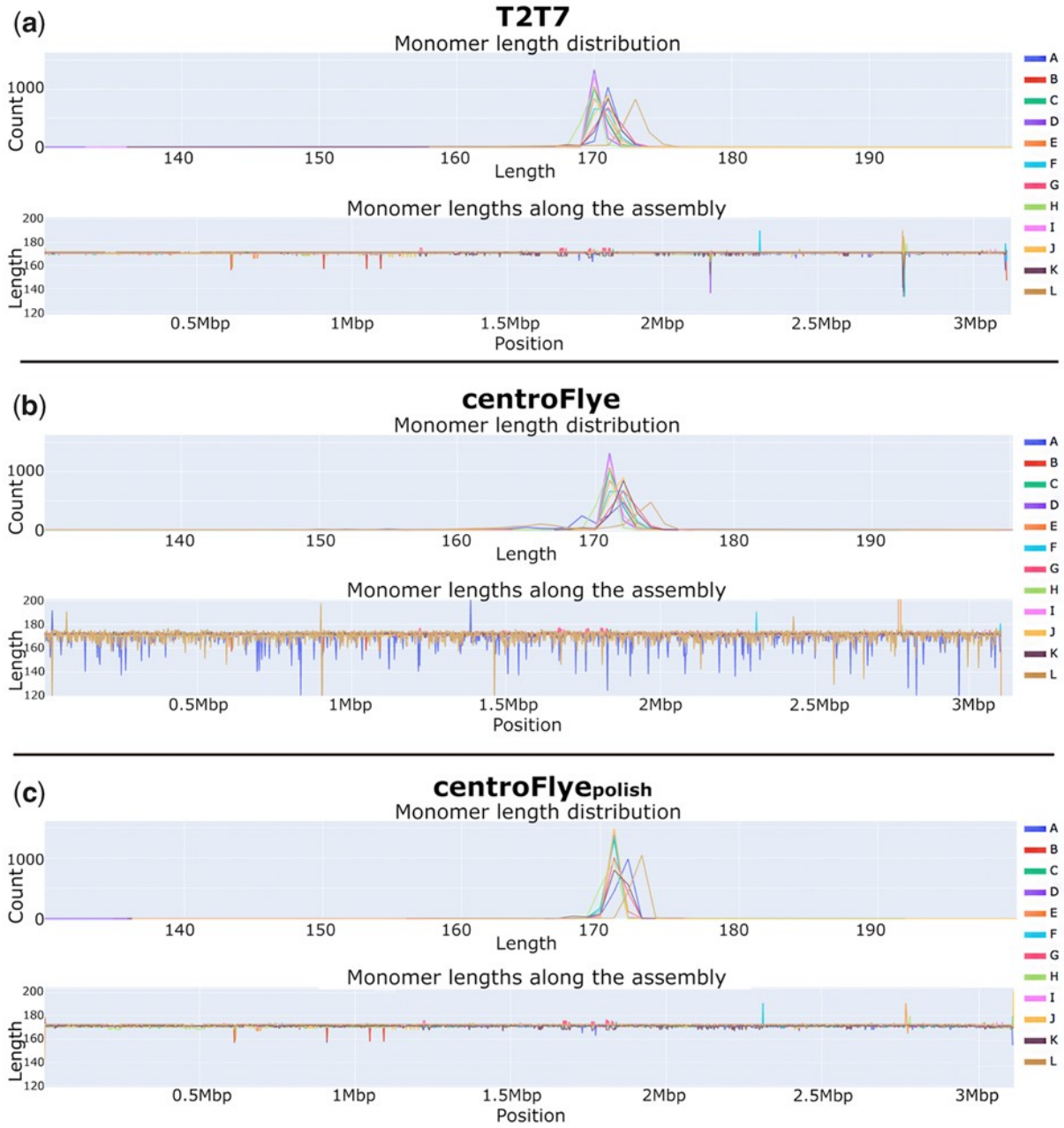


Figure 2.7. Monomer length distribution along the assembly in the T2T7 (**a**), centroFlye (**b**) and centroFlye$_{\text{polish}}$ (**c**) assemblies.

assembly (Fig. 2.5), suggesting that centroFlye assembly requires additional polishing of HOR unit ends. The centroFlye$_{\text{polish}}$ assembly has significantly more uniform monomer lengths as compared to the centroFlye assembly.

Pairwise comparison of assemblies

Supplementary Figure S7 in [41] shows bi-mapping plots for each pair of assemblies. As expected from the analysis of the breakpoint metric (Fig. 2.6), the centroFlye and T2T7 assemblies are nearly identical. The T2T4$_{\text{polish}}$ assembly differs from the T2T7 assembly around ˜350, 1600, 2100 and 2800 kb (coordinates are given for the T2T7 assembly).

2.6 Discussion

We presented the TandemMapper and TandemQUAST tools and applied them to various cenX assemblies. Although these tools detect flaws in ETR assemblies and provide a possibility to assess their quality, they have certain limitations discussed below.

2.6.1 False assembly errors

TandemQUAST is based on mapping reads to the assembly and subsequent analysis. Such an approach implies that inherent errors or systematic biases in the sequencing platforms may affect evaluation of the assembly and bring in some discrepancies that could be considered as false assembly errors. To reduce this effect, TandemQUAST has an option of using accurate PacBio HiFi reads.

2.6.2 Analysis of arbitrary ETRs in human and other genomes

Sequence and structural organization of ETRs, and particularly centromeres, varies widely across species. Since assembly of arbitrary ETRs remains an open problem, there is currently only one tool (centroFlye) for an automatic assembly of some ETRs and few examples of ETR assemblies. We thus limited the scope of our study to the recently completed assemblies of the human centromeres and the GAGE locus (Supplementary Appendix ‘Analyzing ETRs

in the GAGE locus at the human X chromosome' in [35]). Since the T2T consortium aims to generate a gap-free assembly of the entire human genome [34], we anticipate that more high-quality ETR assemblies will soon be generated. These new assemblies will help us to improve the TandemMapper and TandemQUAST tools.

2.6.3 Analysis of diploid assemblies

Since centroFlye is now limited to haploid assemblies, the current version of TandemQUAST also focuses on haploid assemblies. Extending TandemQUAST functionality to diploid assemblies presents a complex algorithmic challenge. However, even effectively haploid cell lines may contain somatic heterogeneity due to clonal genomic instability in the cell culture. In this case, TandemQUAST can report heterozygous sites based on the discrepancies in mapped reads.

2.6.4 Using additional data types for assessing quality of ETR assemblies

We used accurate HiFi PacBio reads to analyze various centromere assemblies but not *bacterial artificial chromosomes (BACs)* and other alternative technologies that represent valuable resources for analyzing tandem repeats (see Supplementary Appendix 'Alternative technologies for ETR assembly quality assessment' in [41]).

For example, a BAC from an ETR is often easier to assemble than an entire long ETR, such as a centromere. For example, centromere Y was recently sequenced using ONT reads to generate assemblies of BACs spanning this centromere [26]. However, certain limitations of the BAC technology make BACs a non-ideal option for ETRs sequence classification [34]. In particular, BACs (i) do not represent a high-throughput approach and thus limit the scope of studies, (ii) have severe differences in coverage that complicate the analysis, (iii) require partial restriction digests that introduce biases in cloning, (iv) may have secondary structures making them incompatible with a bacterial host and (v) since existing short-read assemblers are unable to assemble highly repetitive centromeric BAC from short reads (or even Sanger reads), it is not

clear how to reproduce the semi-manual assemblies of such BACs (some of them assembled two decades ago) with current state-of-the-art assemblers like SPAdes [33]. It is also difficult to accurately assemble BACs from centromeres using long error-prone reads, e.g. recent large BAC sequencing effort has not resulted in assembling such BACs [81]. Thus, if a BAC sequence and a centromere assembly disagree, it is not clear whether this disagreement is caused by an error in the BAC assembly or an error in the centromere assembly. A possible way to address this challenge is a hybrid BAC assembly that combines short and long reads like in [26].

2.7 Acknowledgements

We are grateful to Ivan Alexandrov for many insightful comments and Andrey Prjibelski for helpful discussions and suggestions. This work was supported by St. Petersburg State University, St. Petersburg, Russia [ID PURE 51555639].

Chapter 2, in full, is a reprint of the material as it appears in Mikheenko, A., Bzikadze, A. V., Gurevich, A., Miga, K. H., & Pevzner, P. A. (2020). TandemTools: mapping long reads and assessing/improving assembly quality in extra-long tandem repeats. *Bioinformatics, 36*(Supplement_1), i75-i83. The dissertation author is one of the two lead developers of the TandemTools algorithm and one of the two lead authors of this paper.

Chapter 3

Fast and accurate mapping of long reads to complete genome assemblies with VerityMap

3.1 Abstract

Recent advancements in long-read sequencing have enabled the telomere-to-telomere (complete) assembly of a human genome and are now contributing to the haplotype-resolved complete assemblies of multiple human genomes. Since the accuracy of read mapping tools deteriorates in highly-repetitive regions, there is a need to develop accurate, error-exposing (detecting potential assembly errors), and diploid-aware (distinguishing different haplotypes) tools for read mapping in complete assemblies. We describe the first accurate, error-exposing, and partially diploid-aware VerityMap tool for long-read mapping to complete assemblies.

3.2 Introduction

3.2.1 Emergence of "complete genomics"

The initial draft sequence of the human genome contained thousands of gaps and scaffolds [82], [83]. Even nowadays, the up-to-date human reference genome (GRCh38.p13) still contains 349 gaps and 472 scaffolds [31]. These gaps contain regions that include large and biomedically important multi-megabase arrays of tandem satellite repeats that orchestrate chromosome

segregation during cell division [18], [34]. For two decades, centromeric and pericentromeric satellite arrays evaded all assembly efforts resulting in a limited understanding of their sequence organization. Recently, the Telomere-to-Telomere (T2T) Consortium generated the first nearly complete assembly of an effectively haploid human cell line CHM13 that, in particular, includes assemblies of all satellite repeats [26], [34], [84], [85].

Long-read technologies, such as the ones developed by Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), have changed the landscape of computational methods for genome assembly and opened a possibility to generate haplotype-resolved complete assemblies [1], [46], [5], [61], [3], [7], [75], [86], [87]. Some of these assemblers utilize ultra-long error-prone reads in order to assemble the most repetitive genomic regions, such as satellite arrays [35], [34], [88]. The emergence of long high-fidelity (HiFi) PacBio reads has, once again, revolutionized the field of genome assembly [89], [84] and resulted in some of the most contiguous assemblies to-date [75], [87], [85].

3.2.2 Shifting focus: from detecting mutations to reference-free assembly evaluations.

The emergence of "complete genomics" [75], [87], [85] and "complete metagenomics" [90] is shifting the focus of read mapping from detecting mutations to detecting errors in the newly generated assemblies (see Supplementary Note "Similarities and differences between detecting mutations and detecting misassemblies" in [91]). Although most previous read mapping efforts were aimed at detecting mutations by aligning reads against the reference genome instead of assembling them (since genome assemblers typically fail to assemble highly-repetitive regions), HiFi assemblers accurately assemble even the most complex genomic regions, and while a limited number of misassemblies persist, these assemblies provide a path to potentially eliminate the need for mapping reads against a reference. They also result in extremely low rates of single-nucleotide errors in non-repetitive regions [75], [87], [85], largely eliminating the need for the follow-up read mapping to these regions: instead of mapping reads using external tools, existing

HiFi assemblers automatically provide highly accurate alignments of reads to the assemblies they generate. However, HiFi assemblers still make assembly errors in highly-repetitive regions and thus necessitate development of new mapping tools for detecting these errors by identifying discordant reads.

The need for evaluation of the complete genome assemblies and the absence of ground truth datasets has resulted in the development of specialized pipelines for assembly validation [92]. Even though the QUAST tools represent the state-of-the-art framework for quality assessment of genome assemblies [37], [65], [64], these tools mostly rely on the availability of a reference genome and are thus inapplicable for evaluating complete assemblies that include megabases of novel sequence. In particular, QUAST uses BUSCO [70] to evaluate the gene content of assemblies and assess completeness of an assembly without a reference genome. However, BUSCO only analyzes conserved single-copy genes and is inapplicable for gene-poor regions such as centromeres. Moreover, QUAST identifies assembly errors only in comparison with a reference genome. Alternative reference-free methods can be divided into alignment-free methods that compare *k*-mer spectra between reads and an assembly [93], [94] and alignment-based methods that use the read mapping tools [73], [95], [96], [2], [72], [74] to generate read alignments, use them for assessing the quality of an assembly, and correct the assembly based on the identified discordant reads. However, although alignment-free methods can estimate assembly quality and completeness, they do not output information about the positions of assembly errors.

3.2.3 Error-exposing and diploid-aware read mapping.

Mikheenko et al. [41] demonstrated that performance of existing read-mapping tools deteriorates in complex regions, especially in the presence of assembly errors, and developed the TandemMapper tool for mapping long reads to extra-long tandem repeat (ETRs) in the constructed assemblies. Even though TandemMapper accurately maps reads to centromeric regions and enables evaluations of their assemblies [35], [34], [84], it falls short of mapping reads in even more challenging megabase-long genomic regions like HSAT2,3. Moreover,

TandemMapper, designed specifically for ETRs, becomes prohibitively slow for evaluating complete genome assemblies.

Generating a complete assembly for a single haploid human genome represents a landmark achievement [85]. However, since a single genome does not represent the genomic diversity of the human population, the Human Pangenome Reference (HPR) Consortium is now generating high-quality diploid assemblies for hundreds of humans. This ambitious goal raises the bar for scalable diploid-aware quality assessment tools that can evaluate entire diploid assemblies rather than certain selected (albeit complex) loci and identify errors in these assemblies. To the best of our knowledge, no error-exposing tool for reference-free benchmarking of complete haplotype-resolved assemblies or even individual loci is currently available. Although Winnowmap2 [74], [97] maps reads to complex repetitive regions of the error-free assemblies, it is neither error-exposing nor diploid-aware since its performance deteriorates in the case of error-prone assemblies and heterozygous sites (Table 3.2) [41].

3.2.4 Toward fast, accurate, error-exposing, and diploid-aware read mapping algorithm

We present VerityMap, a fast, accurate, and error-exposing aligner for mapping long reads to the complete assemblies (see Table 3.1 for an informal comparison of various long-read aligners). VerityMap also represents the first step towards diploid-aware alignment by mapping reads to haplotype-resolved assemblies of individual loci (see Supplementary Note "The challenge of diploid-aware read mapping" in [91]). It was used for detecting and correcting misassemblies in the intermediate assemblies of the first complete human genome assembled by the T2T Consortium [92], [85], [34]. Section "Alpha-satellite and human satellite 1,2,3 validation" in [85] illustrates how VerityMap contributed to verifying the complete human genome assembly generated by the T2T Consortium.

All state-of-the-art long-read aligners are based on finding some shared k-mers between a read and an assembly followed by sparse dynamic programming in a graph where each vertex

represents a pair of the starting positions of a shared k-mer. Ideally, the parameter k (k-mer size) should be selected to be as large as possible (e.g., slightly below the read length) for accurate mapping of error-free reads. In practice, it is selected to be rather small to ensure that an error-prone read has some shared k-mer with an assembly. This condition dictates selecting a rather small k in the case of long error-prone reads (minimap2 v2.24 uses default $k = 15$ for mapping ONT reads, and $k = 19$ for mapping PacBio CLR reads https://lh3.github.io/minimap2/minimap2.html) and makes it difficult to map reads in highly-repetitive regions. Accurate HiFi reads allow one to increase k by an order of magnitude thus facilitating read mapping in complex regions.

Although an increased k-mer size looks like a rather minor parameter change in read mapping tools, it actually represents a new challenge. We show that developing a fast, accurate, error-exposing, and diploid-aware long-read mapping tool requires new algorithmic ideas, not unlike constructing the de Bruijn graph of large genomes for large k-mer sizes [33]. To achieve this goal, VerityMap first identifies all rare k-mers in the assembly, carefully selects a small subset of rare k-mers (solid k-mers), finds locations of solid k-mers in each read, constructs a compatibility graph with the vertex-set formed by all matches between the selected solid k-mers shared by a read and the assembly, finds an optimal path in this graph using sparse dynamic programming, and uses this path for read mapping. Since HiFi reads are accurate, VerityMap utilizes large k-mer sizes (e.g., $k = 300$) for constructing the compatibility graph to achieve accurate read mapping. It thus faces the algorithmic challenge of identifying rare k-mers in a large genome for a large k.

Although efficient indexing (finding locations) of all k-mers in the genome (and thus identifying rare k-mers) represents a backbone of many bioinformatics algorithms [98], it remains an open problem in the case of large genomes and large k-mer sizes. Indeed, a naive indexing algorithm with running time $O(|Genome| * k)$ becomes prohibitively slow in the case of accurate HiFi reads since mapping these reads is based on large k-mer sizes (e.g., $k = 300$). Jellyfish [98], KMC3 [99], and more scalable GPU–based k-mer counting approaches [100] generate a database of counts that allows a constant-time count query for any k-mer. However, even though one can

rapidly generate a counting database, existing implementations for indexing all rare k-mers still require $O(|Genome| * k)$ time. On the other hand, the Meryl k-mer counting algorithm [94] only works with $k \leq 64$, which is substantially lower than what is needed for accurate mapping of HiFi reads (e.g., $k = 300$). Inspired by Jellyfish, VerityMap overcomes limitations of existing approaches for rapid indexing of rare k-mers in the case of large genomes and large k-mer sizes by using multiple Bloom filters [101] and the count-min sketches [102].

VerityMap also addresses the challenge of identifying errors in genome assemblies in a reference-free mode. Even though the CHM13 cell line is effectively haploid, it features genomic instabilities between two haplotypes typically represented by insertions in one of the homologous chromosomes that are referred to as Het sites [85]. Interestingly, the ratio of two haplotypes in a Het site is not necessarily 1:1 suggesting that one of the haplotypes might be more prevalent. Automatic detection of Het sites in a haploid assembly and distinguishing them from misassemblies is an important prerequisite for validating diploid assemblies. In addition to its read-mapping module, VerityMap includes a misassembly detection module for identifying misassemblies, Het sites, collapsed haplotypes, and haplotype-switch errors. The T2T Consortium applied this module to verify and correct intermediate assemblies of the CHM13 cell line [85].

3.3 Results

3.3.1 Limitations of existing read-mapping approaches.

Since all existing assemblers generate some misassembled contigs, accurate mapping of reads that span the misassembly breakpoints is a critically important assembly validation step. Although minimap2 [72], [103] and Winnowmap2 [74], [97] accurately map reads to an error-free assembly (even in repetitive regions), their accuracy deteriorates in the case of error-prone assemblies or haplotype-resolved assemblies (since reads from highly similar regions often map to incorrect instances of these regions, albeit with several mismatches). Moreover, the dynamic

programming algorithm for sequence alignment (with standard scoring) often fails to correctly align reads in highly-repetitive regions, motivating the need for a new scoring (Supplementary Note "Frequency-aware sequence alignment scoring" in [91]).

Currently, TandemMapper is the only error-exposing aligner that accurately maps long reads to ETR assemblies that potentially contain misassemblies [41]. However, even though TandemMapper was used to validate the first centromere assemblies [35], [34], [84], it has limitations that prevent its applications to more complex regions of the genome, e.g., it is designed for analyzing various higher-order repeats (HORs) such as human centromeres and is not applicable to repeats without HORs or for analyzing non-repetitive parts of the genome.

3.3.2 Rare and solid *k*-mers

To speed-up the standard (time-consuming) dynamic programming algorithm for aligning long reads, many mappers [72], [74], [41], as well as fast algorithms for detecting overlapping reads in genome assemblers [1], [2], [46], [5], [61], [3], [7], [75], [86], [87] construct a compatibility graph on a carefully selected small subset of *k*-mers in each read. The key differences between these tools are reflected in algorithms for selecting these *k*-mers and weighting the edges of the compatibility graph (rather than constructing the compatibility graph). VerityMap uses a new approach for selecting *k*-mers and weighting edges in the compatibility graph that addresses limitations of the previously developed long-read mappers and enables error-exposing and partially diploid-aware read mapping. VerityMap further finds a longest path in this graph using sparse dynamic programming, and uses this path for read mapping.

A *k*-mer from an assembly is rare if it appears at most MaxRareOccurrences times in this assembly (otherwise a *k*-mer is called frequent). A rare *k*-mer is solid if it appears in a single contig, and its reverse-complementary *k*-mer does not appear in any contig. Note that this definition of a solid *k*-mer differs from the one used by TandemMapper [41]. Although the highly-repetitive regions in the newly assembled complete genome constitute the biggest challenge for read mapping, they form less than 8% of the human genome [85]. Since nearly all

k-mers in other regions are solid, VerityMap downsamples them to reduce the running time and memory footprint (Supplementary Note "Downsampling solid *k*-mers in non-repetitive regions", [91]).

3.3.3 VerityMap pipeline

Figure 3.1 illustrates the VerityMap pipeline. First, VerityMap stores all *k*-mers from the assembly in two Bloom filters [101] and count–min sketch (CMS) data structures [102] for estimating the count of each *k*-mer in the assembly and generating the set of solid *k*-mers that are later used for read mapping.

For each read, VerityMap finds all solid *k*-mers in this read and builds a compatibility graph with the vertex-set formed by pairs of all solid *k*-mers shared between a read and the assembly. Since the downsampled set of solid *k*-mers is small, the compatibility graph is typically small; the longest path in this graph can be rapidly found using sparse dynamic programming. The careful definition of edge-weights in the compatibility graph helps to select the correct path (that represents the primary read alignment) even in the presence of assembly errors. Moreover, since the graph stores information about discrepancies in distances between solid *k*-mers occurring in a read and solid *k*-mers occurring in the assembly, it can be used to detect approximate locations of misassembly breakpoints without a base-level alignment that might be quite unreliable in highly-repetitive regions. VerityMap first attempts to align the read to the forward strand, and then — to the reverse strand. Defaults for all parameters of VerityMap were selected based on performance analysis (Supplementary Note "VerityMap parameters", [91]). The output of VerityMap is in the standard SAM format.

3.3.4 Datasets

Recently, the T2T Consortium generated the first complete assembly of a human genome represented by the CHM13 cell line (referred to as CHM13 assembly; [85], ChrY in this assembly is from a male sample HG002). In addition to analyzing the entire assembly using VerityMap, we

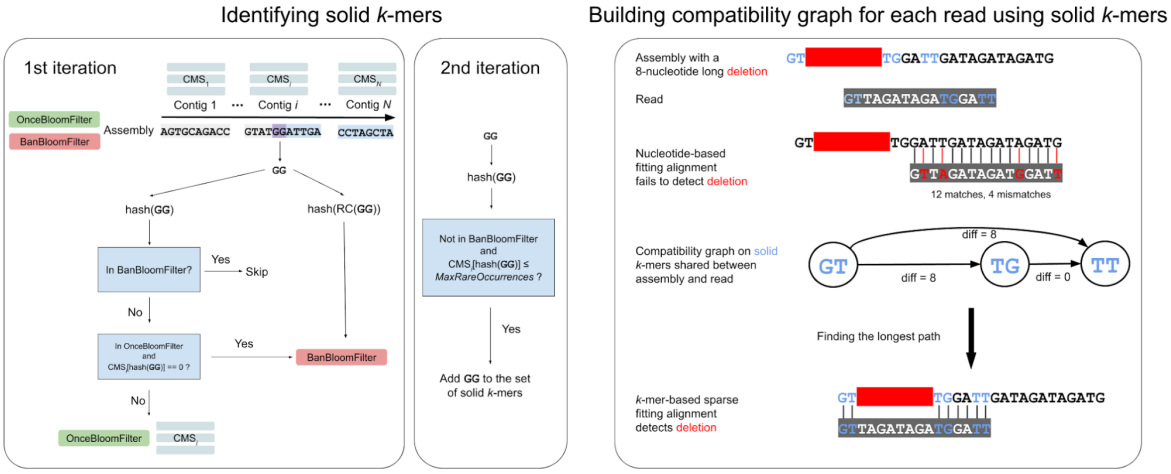


Figure 3.1. VerityMap pipeline. (**Left**) The input of VerityMap is an assembly (a set of contigs) and a set of reads that contributed to this assembly. VerityMap iterates through each contig twice in order to identify solid *k*-mers. At the first iteration, VerityMap stores *k*-mers that appear in multiple contigs and all reverse-complementary *k*-mers within the BanBloomFilter. For each contig, VerityMap constructs a CMS that counts occurrences of *k*-mers within this contig. Finally, VerityMap uses OnceBloomFilter (and BanBloomFilter) to distinguish between rare *k*-mers that appear within a single and multiple contigs. Both Bloom filters and the CMS corresponding to the current contig are being modified simultaneously during the first iteration through the assembly. At the second iteration, VerityMap queries the constructed data structures to identify the set of solid *k*-mers. (**Right**) Aligning a read GTTAGATAGATGGATT against a misassembled contig GTTGGATTGATAGATAGATG with an 8-nucleotide long deletion TAGATAGA (solid *k*-mers are shown in blue). The solid *k*-mer GT (TG) precedes (follows) the deletion breakpoint. The nucleotide-based fitting alignment fails to identify this deletion due to limitations of the standard scoring approaches in highly-repetitive regions. In contrast, VerityMap identifies this deletion using the *k*-mer-based sparse fitting alignment and a new scoring approach. To achieve this goal, it constructs a compatibility graph on all pairs of solid *k*-mers shared between a read and the assembly and finds a longest path in this graph. The new scoring reflects the discrepancies in distances between solid *k*-mers in the assembly (distance 2 between GT and TG in the assembly) and solid *k*-mers in the read (distance 10 between GT and TG in the read), resulting in diff(GT,TG)=8. VerityMap incorporates these discrepancies into the edge-weights of the compatibility graph and outputs a longest path in this graph as the primary read alignment.

focused on a particularly complex centromeric satellite region in chromosome 9. Supplementary Note "Extended benchmarking of long-read mapping tools" in [91] describes benchmarking results on a multitude of even more challenging datasets. Below we focus on one of these datasets that was generated by extracting the centromeric region from chromosome 9 of the CHM13 assembly (referred to as Cen9). This region contains a human satellite 3 (HSat3) array that represented some of the most difficult challenges for the assembly effort of the T2T Consortium (Supplementary Note "Resolution of Chr6 and Chr9" in [85]). To analyze how VerityMap handles assembly errors we introduced an artificial deletion of length 10 kbp into the Cen9 region at position 20 Mbp (referred to as Cen9del10). We further introduce a series of assemblies with various artificial misassemblies including a deletion, insertions, duplications, and a relocation in the Cen9 region: $\mathbf{Cen9_{del5}}$, $\mathbf{Cen9_{ins5}}$, $\mathbf{Cen9_{ins10}}$, $\mathbf{Cen9_{tandem}}$, $\mathbf{Cen9_{dup}}$, $\mathbf{Cen9_{reloc}}$ (detailed description of all datasets is presented in Supplementary Note "Extended benchmarking of long-read mapping tools" in [35]).

It remains unclear how to benchmark VerityMap on haplotype-resolved diploid assemblies since the HPR Consortium is still validating these assemblies. We thus used the assembled centromere of chromosome X (CenX) from the HG002 HiFi read-set (from a male) generated by the HPR Consortium and the assembled CenX from the haploid CHM13 HiFi read-set generated by the T2T Consortium to mimic a diploid assembly of CenX. To analyze how VerityMap handles diploid assemblies we took alpha-satellite regions from chromosome X of CHM13 and HG002 assemblies (combined, we refer to these chromosomes as **ChrXDiploid** — a synthetic diploid chromosome) and merged them into one file (referred to as **CenX-Diploid**). Then we cut each assembly at one of the canonical HOR units and concatenated them to mimic a haplotype-switch error (referred to as **CenX-Diploid-Switch**). Afterward, we simulated reads from both HG002 and CHM13 assemblies of ChrX to check whether VerityMap can detect the haplotype-switch error. We refer to alpha satellite arrays in chromosome X in CHM13 (HG002) genome as **ASat-X** (**ASat-X-HG002**).

To illustrate that VerityMap identifies errors in real assemblies, we consider Cen10 in the

interim version of **CHM13** that predates the earliest publicly released version v0.9 (referred to as **CHM13-Cen10-interim**). **CHM13-Cen10-interim** contains a structural error that was detected by VerityMap and was later corrected by the T2T Consortium (**Cen10** in current **CHM13 assembly** does not contain this error).

Below we list some benchmarking datasets (see Supplementary Note "Extended benchmarking of long-read mapping tools" for the list of all benchmarking datasets in [91]). All simulated datasets are generated with pbsim2 [104].

- **CHM13-SimHiFi** dataset contains HiFi reads simulated from the CHM13 assembly.
- **Cen9Sim** dataset contains HiFi reads simulated from **Cen9**, while **Cen9Sim-Het** dataset contains HiFi reads simulated from both **Cen9** and $\mathbf{Cen9_{del10}}$ (1:1 ratio).
- **CenXDiploid-Sim** dataset contains HiFi reads simulated from **ASat-X** and **ASat-X-HG002**.
- **CHM13-RealHiFi** dataset contains real HiFi reads for the CHM13 sample generated by the T2T Consortium (20 kbp library; accession numbers: SRX7897685-8). This dataset is extended by reads originating from the ChrY of HG002 sample.
- **ChrXDiploid-RealHiFi** dataset contains HiFi reads from **CHM13-RealHiFi** recruited to **ChrXDiploid** using Winnowmap2.

3.3.5 VerityMap maps nearly all reads with an extremely low number of incorrectly mapped reads.

Analysis of the CHM13-SimHiFi dataset (Supplementary Table 2 in [91]) illustrates that VerityMap mapped more than 97% of reads in all chromosomes (except for chr21 where it mapped 94.25% of reads) with an low number of incorrectly mapped reads (76 out of 1.7M reads for the entire genome). Mapped reads cover 99.5% of the entire genome (only 14M uncovered bases).

3.3.6 VerityMap identifies assembly errors and heterozygous sites even in highly-repetitive genomic regions

Table 3.1 illustrates that VerityMap, Winnowmap2, and minimap2 accurately map long and accurate reads to error-free assemblies. However, assembly errors and Het variants often trigger incorrect alignments, especially in highly-repetitive regions. To benchmark the ability to map reads in a vicinity of the misassembly breakpoint, we aligned reads from **Cen9Sim** dataset to $\mathbf{Cen9_{del10}}$ assembly containing an artificial deletion of length 10 kbp (Table 3.2, top left). Although minimap2 and Winnomap2 map more reads than VerityMap, they have a high rate of incorrectly mapped reads while VerityMap yields few incorrectly mapped reads. Since incorrect read alignments may prevent the detection of errors and heterozygous sites in downstream analysis, it is preferable to classify some reads as "unalignable" instead of generating erroneous alignments. Both minimap2 and Winnowmap2 failed to detect this deletion: minimap2 did not report any primary alignment in a 1 kb region before the breakpoint (all reads were incorrectly mapped somewhere else) while Winnowmap2 incorrectly extended alignments through the deletion breakpoint in the $\mathbf{Cen9_{del10}}$ region (Figure 3.2). In addition, to reproduce a scenario of a heterozygous deletion (Table 3.2, top right), we aligned reads from the **Cen9Sim-Het** dataset containing HiFi reads simulated from both **Cen9** and $\mathbf{Cen9_{del10}}$ to the $\mathbf{Cen9_{del10}}$ assembly. Table 3.2 (top right) illustrates that both minimap2 and Winnowmap2 incorrectly mapped reads simulated from Cen9 near the deletion breakpoint. VerityMap either reports reads that are correctly spanning the deletion breakpoint or clips read alignments from **Cen9Sim** dataset at the breakpoint and thus indicates a putative heterozygous deletion. Table 3.2, bottom provides further benchmark of VerityMap, Winnowmap2, and minimap2 of aligning simulated reads from the **Cen9Sim** read-set to the $\mathbf{Cen9_{del5}}$, $\mathbf{Cen9_{ins5}}$, $\mathbf{Cen9_{ins10}}$, $\mathbf{Cen9_{tandem}}$, $\mathbf{Cen9_{dup}}$, $\mathbf{Cen9_{reloc}}$ assemblies (detailed description of these datasets, and benchmarking on more datasets is presented in Supplementary Note "Extended benchmarking of long-read mapping tools"). In all cases, VerityMap has the minimal number of incorrectly mapped reads and either reports

Table 3.1. Aligning 137,507 simulated reads to chromosome 1 (248 Mb length) and 40,272 simulated reads to the centromere of chromosome 9 (42 Mb length) of the CHM13 assembly. Simulated reads are generated as described in Supplementary Note "Extended benchmarking of long-read mapping tools" in [91]. VerityMap, Winnowmap2, and minimap2 incorrectly mapped only 7, 20, and 22 reads to chromosome 1, respectively. Since TandemMapper was designed for accurate read mapping to HOR arrays (rather than large regions without an HOR-like structure), it incorrectly mapped many reads in other regions, so we rate it "+/-" in terms of accuracy for error-free assemblies. VerityMap allows accurate mapping to haplotype-resolved assemblies of individual loci (Figure 3.3) and thus provides the initial step towards diploid-aware mappers for complete haplotype-resolved assemblies (we rate it "+/-"). The best value for each column is indicated in bold. Precise definitions of terms "accurate", "error-exposing", and "diploid-aware" are given in Supplementary Note "Summary of benchmarking results" in [91].

read mapping tool	accurate in error-free assemblies	error-exposing (accurate in error-prone assemblies)	diploid-aware	CPU time (minutes)		memory footprint (GB)	
				chr1	cen9	chr1	cen9
VerityMap	**+**	**+**	**+/-**	275	**500**	**6**	**4**
TandemMapper	**+/-**	+/-	–	**7012**	—	212	—
Winnowmap2	**+**	–	–	**257**	1720	8	32
minimap2	**+**	–	**–**	**28**	**33**	7	8.5

reads that are correctly spanning the breakpoint or clips read alignments at the breakpoint.

3.3.7 VerityMap correctly distinguishes haplotypes in diploid samples and identifies haplotype-switch errors

We aligned reads from the CenXDiploid-Sim dataset to CenXDiploid-Switch assembly containing a haplotype-switch error. We launched VerityMap in the special DiploidVerityMap mode that utilizes a more permissive strategy for selecting solid k-mers. Figure 3.3 illustrates that VerityMap does not incorrectly extend alignments of any reads through the haplotype-switch breakpoint (and thus detects this haplotype-switch error), while Winnowmap2 and minimap2 extend them through it.

We also aligned reads from the ChrX-RealHiFi dataset (total number of reads 621,522) to ChrXDiploid assembly. VerityMap (Winnowmap2, minimap2) mapped 4474 (10,190, 10,255) reads to the incorrect haplotype and 493,012 (553,953, 552,051) — to the correct haplotype. Here, we filtered all secondary alignments, as well as all alignments with mapping quality 0.

Table 3.2. The concept of a correctly/incorrectly mapped read is defined in Supplementary Note "Measuring performance of mapping software" in [91]. Only primary alignments were taken into account. The majority of reads incorrectly mapped by minimap2 and Winnomap2 have secondary alignments to the correct positions: in cases of nearly perfect duplications, minimap2 and Winnomap2 might incorrectly choose primary alignments, while VerityMap classifies a read as "unalignable". Aligning simulated reads (top left) from the Cen9Sim read-set to the Cen9del10 region with an artificial deletion, (top right) from the Cen9Sim-Het dataset to the Cen9del10 sequence. VerityMap reports 2741 (5692) "unalignable" reads in Cen9Sim (Cen9Sim-Het) dataset. (Top left) The total number of reads spanning the deletion breakpoint is 9. VerityMap correctly identifies 6 reads that span the breakpoint and reports 17 read alignments that are clipped at the breakpoint site (note that some of these reads did not span the breakpoint during simulation). (Top right) The deletion breakpoint in this assembly is spanned by 24 simulated reads (15 reads simulated from a haplotype with deletion and 9 reads from a haplotype without deletion). Even though all mappers correctly map all 15 reads from the haplotype with the deletion, only VerityMap correctly identifies two reads from a haplotype without the deletion spanning the breakpoint. Additionally, it reports 14 alignments for reads from haplotype without the deletion clipped at the breakpoint site. The best value for each column is indicated in bold. (Bottom) Benchmarking VerityMap, Winnowmap2, and minimap2 when aligning simulated reads from the Cen9Sim read-set to the Cen9del5, Cen9ins5, Cen9ins10, Cen9tandem, Cen9dup, Cen9reloc assemblies. The number of correctly (incorrectly) mapped reads ranges from 37467 to 37477 (31 to 47) for VerityMap, 39167 to 39218 (1047 to 1096) for Winnowmap2, 39139 to 39191 (1072 to 1124) for minimap2. The best value for each column group is indicated in bold.

	Cen9Sim. Total reads: 40263			**Cen9Sim-Het. Total reads: 80559**		
	#correctly / incorrectly mapped reads	**#correctly / incorrectly extended through misassembly breakpoint**	**# alignments from haplotype w/o deletion clipped at breakpoint**	**#correctly/ incorrectly mapped reads**	**#correctly / incorrectly extended through misassembly breakpoint**	**# alignments from haplotype w/o deletion clipped at breakpoint**
VerityMap	**37498/33**	**6/0**	**17**	747945/73	**2/0**	**14**
Winnowmap2	**39180/1083**	0/9	0	**78368/2176**	0/9	0
minimap2	**39140/1123**	0/0	0	**78286/2230**	0/0	0

Dataset Total reads: 40263	**# alignments correctly / incorrectly extended through breakpoint**			**# alignments clipped at breakpoint**		
	VerityMap	**Winnowmap2**	**minimap2**	**VerityMap**	**Winnowmap2**	**minimap2**
Cen9del5	**8 / 0**	**0 / 0**	**0 / 0**	6	**9**	**6**
Cen9tandem	**6 / 0**	0 / 9	0 / 0	**5**	0	0
Cen9dup	**19 / 0**	4 / 11	15 / 4	**2**	0	0
Cen9ins5	**14 / 0**	9 / 0	**14 / 0**	1	1	1
Cen9ins10	**12 / 0**	0 / 0	2 / 0	**7**	0	0
Cen9reloc	**5+0 / 0+0**	0+0 / 8+0	0+0 / 0+0	**10+8**	0+2	0+0

Figure 3.2. Alignments of reads from **Cen9Sim** dataset to $\text{Cen9}_{\text{del10}}$ generated by minimap2, Winnowmap2, and VerityMap shown using IGV browser [105]. Minimap2 does not report any primary alignments (shown in gray) spanning the deletion breakpoint and all secondary alignments (shown in white) indicate systematic mismatches indicating mapping to an incorrect copy of the repeat. Winnowmap2 reports mappings that are incorrectly spanning the breakpoint. VerityMap either reports an alignment correctly spanning the deletion breakpoint (indicates a 10kb deletion), or clips read alignment at the breakpoint.

Figure 3.3. Visualization of VerityMap, Winnowmap2, and minimap2 alignments of simulated reads from the CenX-Diploid dataset to the CenX-Diploid-Switch sequence using the IGV browser [105]. VerityMap, in difference from Winnowmap2 and minimap2, reveals the haplotype-switch breakpoint.

VerityMap uses a more conservative strategy for mapping reads to the diploid reference to achieve higher accuracy (that is critically important for detecting misassemblies) at the cost of mapping fewer reads. Even though VerityMap presents an advancement in accurate mapping of reads to diploid assemblies, this problem remains a difficult challenge.

3.3.8 VerityMap identifies assembly errors and Het sites using HiFi datasets.

The CHM13-Cen10-interim dataset contains an interim version of Cen10 in CHM13 that predates the earliest publicly released CHM13 assembly by the T2T Consortium. VerityMap was used to detect a deletion of length ˜2.4 kbp in this version, an error that was later corrected by the T2T Consortium and is fixed in the current version of Cen10 in the CHM13 assembly (Figure 3.4). We also aligned the CHM13-RealHiFi dataset to CHM13 assembly to reveal possible assembly errors and Het sites (Supplementary Table 4 in [91]). Overall, we found 86

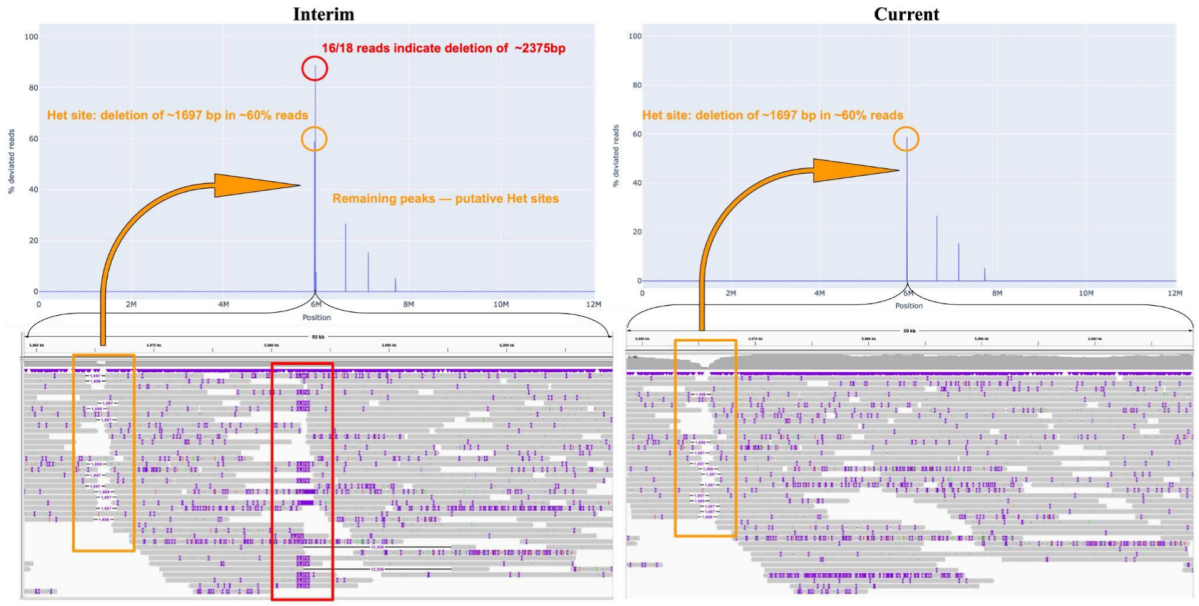


Figure 3.4. The top panel shows the Distance Concordance test applied to CHM13-Cen10-interim (left) and Cen10 in CHM13 assembly (right). The bottom panel shows read alignments around 6 Mbp in two versions of the assembly. (Bottom left) VerityMap reveals a deletion of length ˜2.4 kbp in CHM13-Cen10-interim. The remaining peaks correspond to the Het sites with various multiplicities in the assembly. In particular, one heterozygous deletion of ˜1.7 kbp that is supported by ˜60% of reads is only 20 kbp upstream to the misassembly site. (Bottom right) The misassembly is corrected in the current version of Cen10 assembly, while the putative Het sites remain.

heterozygous sites with *Concordance* = 0.2-0.8, and one site with *Concordance* = 1. The manual validation of this site confirmed a likely 2.4 kbp insertion in the chr19 (Figure 3.5, right; [85]). Importantly, this site was not detected by the state-of-the-art methods for detecting structural variations [92].

3.4 Discussion

We presented the VerityMap tool — the first accurate, error-exposing and partially diploid-aware tool for long-read mapping to complete assemblies. We utilized it to validate and improve the recently published complete human assembly during several rounds of its evaluations [85], [92]. Even though [92] used VerityMap for validating this assembly, they refer to TandemMapper since VerityMap was still in development and did not even have a name at the

time when this paper was submitted. Below we discuss some limitations of VerityMap that we plan to address. All state-of-the-art read-mappers use multiple (often poorly justified) parameters for selecting a small set of k-mers in each read, weighting edges in the compatibility graph, etc. For example, minimap2, the workhorse of long-read mapping, has many parameters (including some hidden parameters) and it remains unclear how they were optimized (no description in [72]). Even though we used the default parameters for Winnowmap2 and minimap2, we do not rule out that other parameter values could improve the performance of these tools. VerityMap also suffers from the "curse of multi-parameters": even though we carefully selected its parameters using alignments of reads to the human genome, a systematic approach using diverse organisms for learning these parameters is needed. In particular, scoring of the edges in the compatibility graph might suffer overfit to the human genome k-mer distribution.

The output of VerityMap is in standard SAM/BAM format and is, in principle, compatible with downstream variant calling tools for detection of Het variants or misassembles. However, VerityMap detects approximate locations of a structural event rather than its exact coordinates which may present a challenge for variant calling tools especially in highly-repetitive regions such as centromeres.

The definition of a solid k-mer requires that it appears in a single contig of the assembly. Even though solid k-mers are not uniformly distributed over the human genome, our benchmarking revealed that VerityMap detected many true positive and never reported false positive misassemblies. However, this definition might be too restrictive for a highly inbred organism with long stretches of homologous chromosomes sharing the same sequence. A highly-fragmented assembly presents a similar challenge since rare k-mers within a single chromosome might be shared between several contigs rendering them non-solid. As a result, little to no solid k-mers will be selected in such regions of the genome leading to gaps in read coverage. Allowing a solid k-mer to be shared by multiple contigs, albeit potentially more error-prone, might mitigate this issue. Similarly, the requirement that the reverse-complement of a solid k-mer does not appear in any contig, needs to be relaxed. We selected a conservative strategy for the initial

VerityMap development since our primary goal was to evaluate the complete haploid complete assembly. Our next goal is to overcome these limitations and apply VerityMap for validating haplotype-resolved assemblies generated by the HPR Consortium. The described benchmarking suggests that VerityMap has a potential for effective evaluation of complete haplotype-resolved assemblies. Supplementary Note "Validation of fragmented haploid and diploid assemblies" [91] introduces a more general k-mer indexing scheme as well as modifications to the compatibility-graph that allow validation of more general fragmented haploid or diploid assemblies (rather than limited to only complete haploid assemblies). Supplementary Note "Extended benchmarking of long-read mapping tools" [91] presents an example of application of VerityMap to a fragmented assembly. Accurate estimation of VerityMap's statistical power for detecting misasemblies in fragmented assemblies remains an open problem.

Even though HiFi reads allow validation of the most regions of the CHM13 assembly, ONT reads provide important complementary quality assessment [85], [35]. Even though this paper focuses on HiFi reads, VerityMap has an ONT mode for ONT reads. We have chosen to focus on HiFi reads since ONT reads currently represent a moving target: over the last year, the error-profile of ONT reads has been rapidly improving with the latest chemistry approaching error-rate as low as 1% [106]. While HiFi reads are more accurate (hence they allow selection of longer solid k-mers), they are shorter than ONT reads. As a result, the length of the longest paths in the compatibility graph for ONT reads can exceed those lengths for HiFi reads and thus connect solid k-mers surrounding "solid k-mer deserts" — long stretches of DNA without any solid k-mers. Together with rapidly improving chemistry, ONT reads might become a self-sufficient alternative for quality assessment of genome assembly.

VerityMap has been designed primarily to detect errors in the assemblies rather than identify structural variations (SVs) in genomes. Even though, many misassemblies and SVs can be represented as indels, the distribution of misassemblies and SVs over the universe of all possible indels are substantially different. First, a prominent source of genomic diversity is presented by short SVs and copy number variants (CNVs) in short tandem repeats (STRs). Such

differences can hardly ever be observed as misassemblies since they do not present challenge for long-read assemblers. Previous studies demonstrated that HiFi assemblers are very accurate with respect to single-nucleotide substitutions and short indels [85], [87], [107] and these inaccuracies in assemblies typically present polishing artifacts rather than imperfections of an assembly algorithm itself. Second, misassemblies typically involve large (multi-kbp) long indels (or inversions) that correspond to the incorrect traversal of the assembly graph (or, worse, its faulty topology). While it is possible that an alternative (and incorrect) traversal of the graph might result in a haplotype that is present in the population (the authors are aware of a single case that involves a large 600 kbp inversion on ChrY recently assembled by the T2T Consortium), it remains unclear whether this is a widespread phenomenon. Finally, some of the variations in the genome (for example, in human centromeres) are much more complex than simple CNVs of the tandem repeat units and it seems infeasible that an assembly algorithm might produce such a complicated misassembly as a result of an incorrect graph traversal.

When developing VerityMap, we have concentrated on simulated examples that resemble misassemblies that we have observed in practice of developing assembly algorithms. Specifically, (1) the indels that were introduced in the early versions of centromere assemblies [34] that were subsequently detected by an early version of TandemMapper [41] and were later corrected in [35], (2) misassemblies that were present in interim versions of CHM13 assemblies generated by the T2T Consortium, (3) misassemblies generated by LJA assembler [107]. Supplementary Note “Similarities and differences between detecting mutations and detecting misassemblies” in [91] demonstrates that VerityMap can be used as an input to various SV detection tools and that VerityMap’s own DistanceDiscordant test can complement these tools.

3.5 Methods

3.5.1 Selecting solid *k*-mers

Similar to TandemMapper [65], VerityMap uses rare *k*-mers as anchors for the read mapping. Selection of the *k*-mer size is dictated by the error rate in long reads. Since long error-prone reads have a high error rate, long genomic *k*-mers rarely "survive" without errors in such reads. For example, the probability that a 19-mer "survives" in an error-prone ONT read (generated with basecaller Guppy flip-flop 2.3.1) is only 0.34 [35]. However, since long and accurate reads have much smaller error rates (0.7% for ONT reads generated with flow cell R10.4 https://nanoporetech.com/accuracy and 0.2% for HiFi reads [85]), one can set the large default *k*-mer size to accurately map such reads even in highly-repetitive regions such as Cen9 (see Supplementary Note "The choice of *k*-mer size and the shortest unique substrings" in [91]). VerityMap sets the default value k = 301 and MaxRareOccurrences = 10 (see Supplementary Note "VerityMap parameters" in [91]).

HSAT2,3 array in the Cen9 region is an ETR containing long inverted repeats. Importantly, for 6,344 out of ≈ 22 million rare *k*-mers ($k = 301$) in this region, their reverse-complementary *k*-mer is frequent in the genome. Since the strandedness information is not available, using such *k*-mers for read mapping might reduce the mapping accuracy. Similarly, utilizing a rare *k*-mer that is shared between two contigs (for example, two homologous or non-homologous chromosomes) might also lead to incorrect read mapping. Since VerityMap aims primarily to reduce the number of incorrect alignments, it implements a rather conservative strategy and uses solid rather than rare *k*-mers for read mapping to complete assemblies.

3.5.2 Sparse dynamic programming for read mapping

A standard dynamic programming approach for computing the fitting alignment of a query onto a target finds a longest path in the grid-like directed acyclic graph (DAG) where vertices correspond to all pairs of positions (one in the target and one in the query). Analysis of

the CHM13 assembly by the T2T Consortium revealed that, although the fitting alignment of a read usually finds its correct position in the assembly, it may misplace the reads that originated from highly-repetitive regions (e.g., centromeres and other ETRs), particularly in regions with assembly errors (see Supplementary Note "Frequency-aware sequence alignment scoring"). Thus, mapping reads to ETRs requires a new algorithm and a new scoring approach.

Sparse dynamic programming [108] is a common way to speed-up sequence alignment by selecting a small subset of points in a grid that are likely traversed by an optimal path, constructing an edge-weighted DAG on these points (rather than on all points in the entire grid), and finding a longest path in the resulting smaller graph. We note that although sparse dynamic programming approximates rather than reconstructs the optimal sequence alignment, this approximation is usually sufficient for read mapping. However, to achieve an accurate approximation, one has to (i) carefully select a vertex-set of the DAG, and (ii) carefully define the edge-weights in the sparse DAG to address the ETR mapping challenge. E.g., a k-mer-based sparse fitting alignment amounts to finding a longest path in the DAG where vertices correspond to all pairs of k-mer-matching positions from the target and the query (positions are k-mer-matching if k-mers starting at these positions are identical) and directed edges connects all pairs of vertices (i, j) and (i', j') with $i < i'$ and $j < j'$. However, the standard scoring approach for the sparse fitting alignment, that scores all edges with unit weights, fails to address the ETR mapping challenge since it does not account for greatly varying frequencies of k-mers in some targets.

Although the sparse fitting alignment is faster than the standard fitting alignment, it still may be computationally infeasible in long highly-repetitive regions due to a quadratic running time in the number of matching positions. Below we describe how VerityMap addresses the mapping challenge by considering matches of (undersampled) solid k-mers rather than all k-mers.

3.5.3 Compatible k-mers

As solid k-mers are not necessarily unique in the assembly, we consider each occurrence of each solid k-mer separately. Let a_R and b_R (a_A and b_A) be occurrences of solid k-mers a and b in the read R (contig A) such as a_R precedes b_R (a_A precedes b_A). To make a_R and b_R uniquely defined for each read, we limit attention to solid k-mers that appear exactly once in this read. We define $d(a_R,b_R)$ ($d(a_A,b_A)$) as the distance between a_R and b_R in R (a_A and b_A in A). We refer to the pair of a_A and a_R (b_A and b_R) as a match a_M (b_M) and write $a_M < b_M$ if a_A precedes b_A and a_R precedes b_R. We define $distance(a_M,b_M) = \min d(a_R,b_R), d(a_A,b_A)$, $diff(a_M,b_M) = |d(a_R,b_R) - d(a_A,b_A)|$, and $bias(a_M,b_M) = \ln(diff(a_M,b_M))$ (Supplementary Note "Frequency-aware sequence alignment scoring" in [91]).

We call matches a_M and b_M close if $distance(a_M,b_M) < MaxJump$ (the default value $MaxJump = 100,000$) to exclude k-mers locating far apart from each other but preserve consecutive rare k-mers even if they are separated by a long stretch of frequent k-mers. Matches a_M and b_M are *overlapping* if $distance(a_M,b_M) < k$. Close matches a_M and b_M are *synced* if either (i) a_M and b_M are overlapping with $diff(a_M,b_M) = 0$, or (ii) they are not overlapping. Finally, we call a_M and b_M compatible if (i) $a_M < b_M$ and (ii) a_M and b_M are synced (see Supplementary Note "Frequency-aware sequence alignment scoring" in [91]). Note that VerityMap modifies the notion of compatible k-mers introduced in [41] to detect assembly errors that evade detection by TandemMapper.

3.5.4 Compatibility graph

Given a read R, an assembly A, and a set S of solid k-mers, VerityMap constructs a weighted compatibility graph $G(R,A,S)$ on the vertex-set of all pairs of identical solid k-mers from R and A. Vertices $a_M = (a_A,a_R)$ and $b_M = (b_A,b_R)$ are connected by an edge if a_M and b_M are compatible. Edge-weights in this graph should be carefully chosen since a match of a unique solid k-mer is more valuable than a match of a non-unique solid k-mer and since

non-overlapping matches are more valuable than overlapping matches. VerityMap thus defines $weight(a_M, b_M) = premium(a_M, b_M) - penalty(a_M, b_M)$ in such a way that (i) $premium(a_M, b_M)$ is larger in the case when b_M represents a unique k-mer as compared to the case when it represents a non-unique k-mer, (ii) $premium(a_M, b_M)$ is larger for non-overlapping matches a_M and b_M as compared to overlapping matches, and (iii) if a_M and b_M represent a pair of k-mers that flank misassembly breakpoints, $penalty(a_M, b_M)$ should not be too large, thus allowing VerityMap to detect a misassembly.

To address the goal (i), VerityMap assigns the score *UniqueScore* to all unique k-mers and the smaller score *RareScore* to all non-unique solid k-mers in the assembly (default values *UniqueScore* = 3 and *RareScore* = 0.1). The score $Score(b_M)$ of a match b_M is defined as the score of the k-mer it represents. To address the goal (ii), VerityMap defines the score $EdgeScore(a_M, b_M)$ of the edge (a_M, b_M) in the compatibility graph as 1 for non-overlapping matches and as a smaller $distance(a_M, b_M)/k$ for overlapping matches. Finally, it defines $premium(a_M, b_M) = Score(b_M) * EdgeScore(a_M, b_M)$. To ensure that the longest path in the compatibility graph correctly aligns a read against the assembly, it is important to penalize edges with large bias, e.g., to define $penalty(a_M, b_M) = bias(a_M, b_M)$. However, to address the goal (iii) and to design an error-exposing read-mapper, it is important that the penalties of edges connecting k-mers that represent misassembly breakpoints are not excessive, even in the case when these edges have a large bias. VerityMap thus limits the maximum possible penalty using the *MisassemblyPenalty* threshold (default value is 5) and defines $penalty(a_M, b_M) = \min(MisassemblyPenalty, bias(a_M, b_M))$. Supplementary Note "Finding a longest path in the compatibility graph" in [91] describes how VerityMap finds a longest path in the compatibility graph.

3.5.5 The challenge of finding positions of all solid k-mers in a large genome

TandemMapper uses a naive strategy of indexing solid k-mers in a genome *Genome* (finding a list of positions for each solid k-mer) by traversing *Genome* twice. During the first traversal, it counts all k-mers by storing them in a hash-table. During the second traversal, it checks if a k-mer is solid using the constructed hash-table and records positions of all solid k-mers (total time and memory is $O(|Genome| * k)$). This strategy becomes prohibitively time-consuming unless mapping reads against relatively short regions (e.g., several Mbp) using short k-mers (e.g., with $k < 30$). VerityMap uses a different strategy that enables mapping reads against the entire human genome using long k-mers (e.g., with $k > 300$).

The complexity of pattern matching can be reduced by indexing rolling hashes of k-mers rather than the k-mers themselves [109]. Given a rolling hash function and a hash value for a k-mer $a_i \dots a_{i+k-1}$ in a genome $\dots a_i \dots a_{i+k-1} a_{i+k} \dots$, one can compute the hash value of the next k-mer $a_{i+1} \dots a_{i+k}$ in $O(1)$ time, thus, improving the run-time complexity from $O(|Genome| * k))$ to $O(|Genome|)$. Note, that a hash collision will lead to an overestimated frequency for k-mers with collided hashes. This might lead to misclassification of some solid k-mers as non-solid, but never leads to recruiting non-solid k-mers. Unfortunately, the existing k-mer counting tools do not allow querying a count of a rolling hash instead of a k-mer, and thus do not immediately lead to a fast indexing algorithm with $O(|Genome|)$ running time. Moreover, since most of the k-mers in the non-repetitive parts of the genome are solid, we aim to select only their small subset for constructing the (small) compatibility graph (see subsection "Filtering solid k-mers in non-repetitive regions"). Unfortunately, the existing k-mer counting tools do not produce a downsampled database of k-mer counts, which leads to an unnecessary computational burden. To overcome the limitations of existing approaches, VerityMap uses a probabilistic procedure for indexing solid k-mers inspired by the Jellyfish algorithm [98]. Below we briefly describe the Bloom filter and the count-min sketch data structures that VerityMap uses for indexing solid

k-mers.

3.5.6 Bloom filter and count-min sketch

The Bloom filter [101] is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set and that may report false positives (with a small false positive probability) but never false negatives. Given a genome *Genome* (or its assembly), its Bloom filter is formed by *BloomNumber* independent hash functions, each mapping a k-mer from *Genome* into a bit array of size *BloomSize*. Storing all k-mers in a Bloom filter allows one to quickly query whether an arbitrary k-mer occurs in the genome. Given hash functions $h_1, h_2, \ldots, h_{BloomNumber}$ and a k-mer a, one can quickly check whether all bits $h_1(a), h_2(a), \ldots, h_{BloomNumber}(a)$ of the Bloom filter are equal to 1, an indication that the k-mer a may occur in the genome. Thus, a Bloom filter allows one to efficiently query whether an arbitrary k-mer occurs in the genome (with a small false positive probability).

Bloom filter allows one to check whether an element belongs to a multiset but does not evaluate how frequent it is in this multiset. The count–min sketch (CMS) is a space-efficient probabilistic data structure that provides an upper bound on the count of an element in a multiset [102]. CMS is a modification of a single-bit Bloom filter array that uses a counter array of size *CMSNumber* * *CMSSize* with cells containing several bits. This property allows it to count repeatedly inserted elements: when an element is inserted, the values of all *CMSNumber* hash functions are computed and the corresponding cells in the counter arrays are incremented by 1 (in case an increment leads to an overflow, the corresponding cell gets frozen). To provide an upper bound for the count of a given element, CMS computes the values of all *CMSNumber* hash functions and reports the minimum of the corresponding cells in the counter arrays (a frozen cell votes for the infinity count). Thus, CMS sometimes overestimates the count of an element but never underestimates it. VerityMap uses CMS to distinguish unique, rare, and frequent k-mers and with a controlled probability misclassifies a rare (unique) k-mer as frequent (rare).

3.5.7 Identification of rare k-mers using CMS

For each *Contig* in genome *Genome*, VerityMap counts all k-mers in this contig by storing them in the count-min-sketch *CMS(Contig) = CMS(Contig, k, Bits, CMSNumber, CMSSize)* formed by CMSNumber hash functions, each mapping a k-mer into a counter array of size *CMSSize* where each counter takes *Bits* bits. The memory footprint of CMS is *CMSNumber*Bits*CMSSize* bits. The total run-time for inserting all k-mers from *Contig* into this CMS is $O(|Contig| * CMSNumber * Bits * k)$. Storing rolling hashes instead of k-mers further reduces the complexity to $O(|Genome| * CMSNumber * Bits)$. Supplementary Note "VerityMap parameters" in [91] describes how VerityMap sets the parameters of the CMS and the Bloom filters.

3.5.8 Identification of solid k-mers using the Bloom filter

To verify whether a rare k-mer is solid, VerityMap checks that (i) its reverse-complement does not appear in the genome and (ii) it appears only in a single contig. Storing all k-mers from the genome *Genome* in a Bloom filter allows VerityMap to efficiently query whether an arbitrary k-mer is present in *Genome*. In order to check conditions (i) and (ii) for identifying solid k-mers, VerityMap constructs two Bloom filters — *BanBloomFilter(Genome)* and *OnceBloomFilter(Genome)* of an optimal size [101] for the expected number of k-mers ($|Genome|$ is an upper bound) and for the target false positive probability *BloomFPP* (default value *BloomFPP* = 0.00001).

For every rare k-mer a, we ensure that its reverse complement a^* does not appear in the *Genome* by inserting a^* in the *BanBloomFilter* and, in the end, checking that a is not present in *BanBloomFilter*. To check if a k-mer a appears in a single contig, VerityMap scans all contigs and synchronously inserts a k-mer a from a contig *Contig* in both *OnceBloomFilter(Genome)* and *CMS(Contig)*. This synchronous construction of both *OnceBloomFilter(Genome)* and *CMS(Contig)* allows one to check whether the k-mer a occurs in a single contig, albeit with a small probability of error. Indeed, if it is already present in *OnceBloomFilter(Genome)*

but missing in *CMS(Contig)*, it likely appeared in a previously scanned contig. In this case, VerityMap filters out the k-mer a by inserting it into *BanBloomFilter(Genome)*. However, if it is present in both *OnceBloomFilter(Genome)* and *CMS(Contig)*, the k-mer a is retained since it may occur in a single contig.

To reduce the complexity, instead of inserting k-mers in the Bloom filter, we store their rolling hashes. The total running time of finding all solid k-mers and determining their genomic positions is $O(|Genome| * (BloomHashNumber + CMSNumber))$. The total memory footprint is $O(|Genome| + (CMSNumber * CMSSize * Bits + BloomHashNumber * BloomSize))$.

Since the Bloom filter is a probabilistic data structure, some solid k-mers might be falsely removed as breaking either condition (i) or (ii). In practice, very few solid k-mers are filtered out (as controlled by the parameter BloomFPP) and that does not lead to deterioration of read mapping. Importantly, no true non-solid k-mer might be misclassified as solid. For example, a naive implementation of an "exact" strategy (that uses deterministic albeit more memory consuming data structures) extracts 149,548,373 solid k-mers from ChrX in CHM13 assembly, while the "approximate" strategy described above (based on Bloom filters and CMSs) extracts 147,105,002 solid k-mers from the same dataset (more than 98% of truly solid k-mers). For the sake of easy comparison, we did not downsample solid k-mers in this example as described in Supplementary Note "Downsampling solid k-mers in non-repetitive regions" [91].

3.5.9 Distance Concordance test

VerityMap detects approximate locations of misassembly breakpoints without requiring a time-consuming nucleotide-level alignment. Indeed, if a read R spans a misassembly breakpoint, a longest path in the compatibility graph $G(R,A,S)$ often provides a hint for finding this breakpoint. For example, a deletion of length L in an assembly typically triggers an edge (a_M, b_M) with a surprisingly large difference $diff(a_M, b_M) \approx L$, where a_M and b_M represent solid k-mers flanking the deletion breakpoint in such a way that a_M (b_M) precedes (follows) the deletion breakpoint as illustrated in Figure 3.1.

However, a single read may have non-zero values of $diff(a_M, b_M)$ even in the case of error-free assembly, especially when $d(a_R, b_R)$ is high. However, this difference is usually low for accurate HiFi reads and error-free assemblies, suggesting that large differences can be used for detecting misassemblies. To demonstrate that it is indeed the case, we considered 18 millions of unique k-mers ($k = 301$) in Cen9 and selected 100,000 pairs of unique k-mers such that k-mers in each pair are 10,000 bp apart in the assembly. The mean distance between the same pairs of k-mers in reads was $10,002 \pm 3.4$ bp. Thus, $diff(a_M, b_M)$ usually does not exceed 3*3.4=10.2 bp if $d(a_A, b_A) = 10,000$ bp. Given our upper limit for a distance between two consecutive k-mers equal to 100,000 bp, we set a lower threshold for an indel size *MinIndelSize* to 100 bp. Thus, if nearly all longest paths containing k-mers a and b show a systematic bias in values of $diff(a_M, b_M)$ and $diff(a_M, b_M) > MinIndelSize$, then an indel of approximate size $diff(a_M, b-M)$ is likely contained between a_A and b_A. We classify each read that shows systematic bias in values of $diff(a_M, b_M)$ as a discordant read and report the fraction of discordant reads connecting each pair of solid k-mers in the assembly.

Formally, for each read R and for each pair a_M and b_M of consecutive matches in a longest path in compatibility graph we add the number $diff(a_M, b_M)$ to a set $Diffs(a_A, b_A)$. We test the hypothesis that the mean of $Diffs(a_A, b_A)$ is 0 and refer to the pair a_A and b_A as significant if this hypothesis is rejected (two-sided one-sample t-test). We use Bonferroni family-wise error rate correction and set up significance level α_{DC} (the default value $\alpha_{DC} = 0.05$). For each significant pair a_A and b_A of consecutive k-mers in a contig A, we compute the proportion $Concordance(a_A, b_A)$ of the most frequent difference $TopDiff(a_A, b_A)$ in $Diffs(a_A, b_A)$. Figure 3.5 (left and middle) shows the $Concordance(a_A, b_A)$ across the $\text{Cen9}_{\text{del10}}$ region when aligning simulated reads from the dataset Cen9Sim (Cen9Sim-Het) and reveals the artificial deletion (Het site) of length 10 kbp at position 20 Mbp. Figure 3.5 (right) shows the $Concordance(a_A, b_A)$ across cen19 when aligning real HiFi reads and reveals the likely assembly error and several Het sites (see Results).

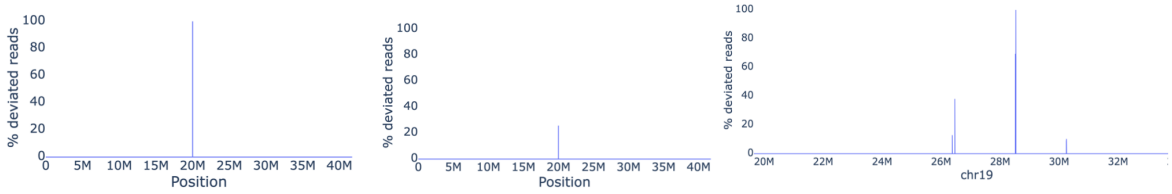


Figure 3.5. Distance concordance test applied to simulated and real read-sets. (Left) Mapping reads from the dataset Cen9Sim reveals an artificial deletion at location 20 Mbp. $Concordance(a_A, b_A)$ = 1 and $TopDiff(a_A, b_A)$ = 10,003 for solid k-mer a_A at position 19,999,691 and b_A at position 20,000,001 revealing the approximate location of the deletion of size 10 kbp. (Middle) Mapping reads from the dataset Cen9Sim-Het reveals an artificial Het site at position 20 Mbp with $Concordance(a_A, b_A) = 0.33$ for the same solid k-mers a_A and b_A and $TopDiff(a_A, b_A) = 10,005$ pointing at insertion in Cen9del10 of size 10 kbp existing in one of the artificial haplotypes. (Right) Mapping HiFi reads reveals several Het variants and a likely assembly error (with $Concordance = 1$ at 28.5 Mbp) in the centromere of chromosome 19 (see Supplementary Note "Extended benchmarking of long-read mapping tools" in [91] for description of all datasets).

3.6 Acknowledgements

This work was supported by St. Petersburg State University, St. Petersburg, Russia [ID PURE 73023672]. We are indebted to Daniel Baker, Daniel Lemire, and Thomas Mueller for their suggestion to use count-min-sketch. We are grateful to Alexander Bzikadze and Cynthia Wu for helpful discussions and suggestions.

Chapter 3, in full, has been submitted for a journal publication and may appear as Bzikadze, A. V., Mikheenko, A., & Pevzner, P. A. (2022). Fast and accurate mapping of long reads to complete genome assemblies with VerityMap. The dissertation author is the primary developer of the VerityMap algorithm and the first author of this paper.

Chapter 4

Automated annotation of human centromeres with HORmon

4.1 Abstract

Recent advances in long-read sequencing opened a possibility to address the long-standing questions about the architecture and evolution of human centromeres. They also emphasized the need for centromere annotation (partitioning human centromeres into monomers and higher-order repeats [HORs]). Although there was a half-century-long series of semi-manual studies of centromere architecture, a rigorous centromere annotation algorithm is still lacking. Moreover, an automated centromere annotation is a prerequisite for studies of genetic diseases associated with centromeres and evolutionary studies of centromeres across multiple species. Although the monomer decomposition (transforming a centromere into a monocentromere written in the monomer alphabet) and the HOR decomposition (representing a monocentromere in the alphabet of HORs) are currently viewed as two separate problems, we show that they should be integrated into a single framework in such a way that HOR (monomer) inference affects monomer (HOR) inference. We thus developed the HORmon algorithm that integrates the monomer/HOR inference and automatically generates the human monomers/HORs that are largely consistent with the previous semi-manual inference.

4.2 Introduction

Recent advances in long-read sequencing technologies led to rapid progress in centromere assembly in the past year [35], [34], [75], [85], [84], [110] and, for the first time, opened a possibility to address the long-standing questions about the architecture and evolution of human centromeres [111], [112]. "Alpha satellite arrays" of live human centromeres that organize the kinetochore (which we refer to simply as "centromeres") are tandem DNA repeats that are formed by units repeating thousands of times with limited nucleotide-level variations but extensive variations in copy numbers in the human population [14]. We refer to "live" centromeres as those that host the kinetochore as revealed by CENPA=CENH3 binding ("live" corresponds to "active" in [110]. Each such unit represents a tandem repeat formed by smaller repetitive building blocks (referred to as "monomer blocks"), thus forming a "stacked tandem repeat" (4.1. Partitioning all monomer blocks into clusters of similar monomer blocks defines "monomers," where each monomer represents the consensus of all monomer blocks in a given cluster. The emergence of centromere-specific stacked tandem repeats is a fascinating and still poorly understood evolutionary puzzle [49], [113], [111], [42].

Each human monomer is of length ˜171 bp, and each higher-order unit is formed by multiple monomers that differ from each other. A monomer is "frequent" if the number of monomer blocks in its cluster exceeds a frequency threshold, and "infrequent," otherwise. Recently, [42], [35], [47], [114] revealed still underexplored "hybrid" monomers (each hybrid monomer is a concatenate of two or even more frequent monomers) and hypothesized that they may drive the "birth" of new frequent monomers. Different human centromeres typically have different monomers and units, and the number of the frequent monomers in a unit varies from two for Chromosome 19 to 19 for Chromosome 4.

A "canonical (cyclic) order of monomers" (referred to as a "higher-order repeat" [HOR]) is specific to each centromere and is defined evolutionarily as the ancestral and chromosome-specific order of frequent nonhybrid monomers that has evolved into the complex organization

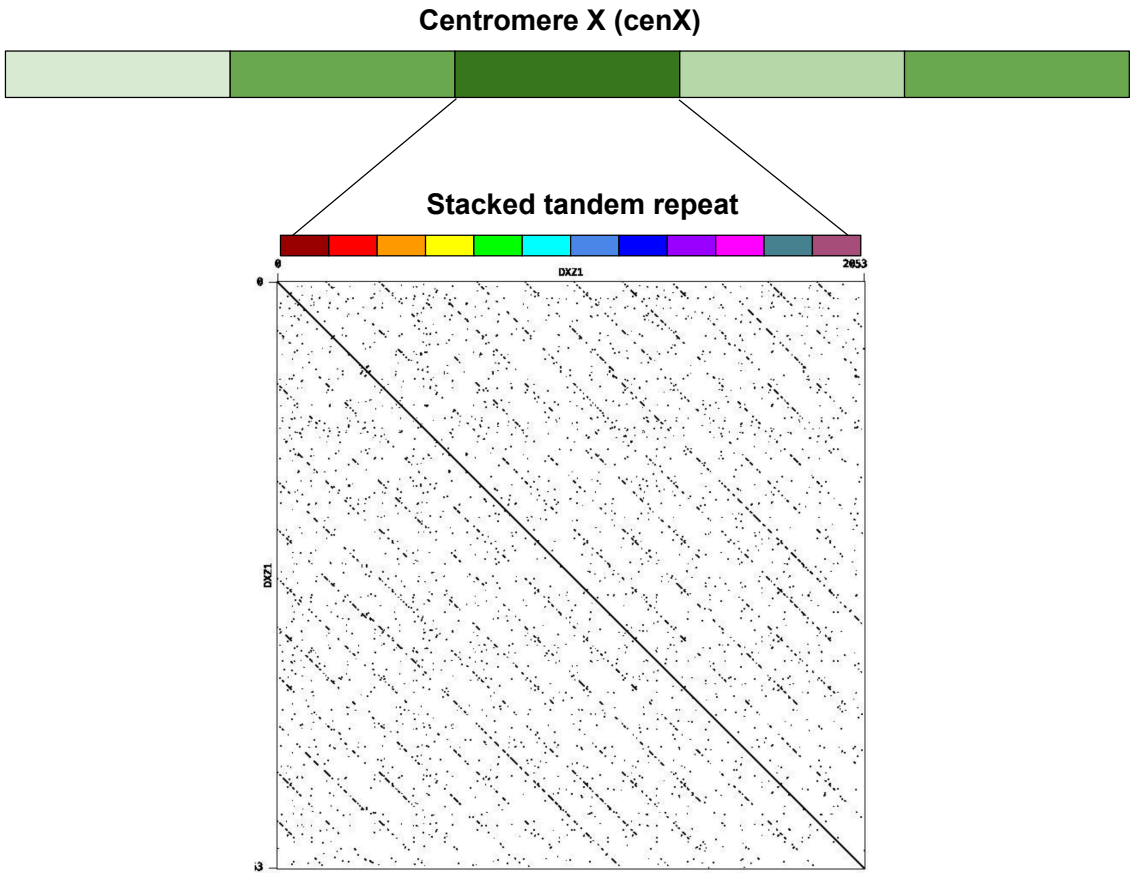


Figure 4.1. The architecture of centromere on Chromosome X. The centromere of Chromosome X (cenX) consists of ˜18,100 monomers of length ˜171 bp each based on the cenX assembly in [35]; the T2T assembly [85] represents a minor change to this assembly. These monomers are organized into ˜1500 units. Five units are colored by five shades of green illustrating unit variations. Each unit is a stacked tandem repeat formed by various monomers. The vast majority of units in cenX correspond to the canonical HOR, which is formed by 12 monomers (shown by 12 different colors). The figure on top represents the dot plot of the nucleotide sequence of the canonical HOR that reveals 12 monomers. Although the canonical units are 95%–100% similar, monomers are only 65%–88% similar. In addition to the canonical 12-monomer units, cenX has a small number of partial and auxiliary HORs with varying numbers of monomers.

of extant centromeres. This definition, however, is computationally nonconstructive because the ancestral order is unknown and no algorithm for its inference has yet been described. The current view of centromere evolution can be summarized by the following framework that we refer to as the "Centromere Evolution (CE) Postulate":

- Each extant human centromere has evolved from a "single" ancestral HOR formed by "k different" monomers. Hence, each monomer occurs in a HOR only once. The parameter k (number of monomers in a HOR) varies between various centromeres.

- Each frequent nonhybrid monomer in a centromere has evolved from a single ancestral monomer. The number of ancestral monomers equals the number of frequent nonhybrid monomers in the extant centromere.

- Each hybrid monomer has evolved from a concatenate of two (or even more) ancestral monomers and does not participate in the ancestral HOR.

- In addition to units formed by canonical HORs, there exist units formed by "partial HORs" (substrings of canonical HORs). All other units consist of a single hybrid monomer and are referred to as "auxiliary HORs." Although the canonical HOR corresponds to the most frequent unit for most human centromeres, it is not always the case.

Although the CE postulate is widely accepted [115], [116], [117], [110], we are not aware of a rigorous proof of this postulate or an algorithm that, given an extant centromere, derives its canonical HOR (Supplemental Note 1 in [118]). Moreover, because the concept of a HOR is parameter-dependent, the CE postulate may hold for some parameters and fail for others. However, it is not clear how to select various parameters such as the frequency threshold parameter (for defining the concept of a frequent monomer), the percent identity parameter (for deciding which monomer blocks correspond to the same monomer), and parameters for classifying a monomer as a hybrid [114].

Moreover, the CE postulate implicitly assigns the inferred HOR to a particular (and unspecified) moment in the past. For example, although the HOR for centromere X (referred to as cenX) consists of 12 monomers, this 12-monomer HOR evolved from an even more ancient 5-monomer ancestral HOR [115], [116]. It is thus not clear how an algorithm for HOR inference should choose between a 12-monomer HOR and a 5-monomer HOR for cenX. Further, even if the CE postulate holds, it may be impossible to infer canonical HORs if nearly all information about the ancestral HOR was erased by millions of years of evolution; for example, it is unclear how to derive HORs in mouse centromeres [112].

Recent evolutionary studies of centromeres [42], [35], [119] revealed the importance of partitioning them into monomers, the problem that was addressed by the StringDecomposer algorithm [47]. Given a nucleotide string *Centromere* and a monomer set *Monomers*, StringDecomposer decomposes *Centromere* into monomer blocks (each block is similar to one of the monomers) and transforms it into a "monocentromere" string $Centromere^*$ over the alphabet of monomers. For each monomer M, it generates the set of "M blocks" in the centromere that are more similar to M than to other monomers (ties broken arbitrarily).

StringDecomposer opened a possibility to automatically generate all HORs and annotate human centromeres (i.e., partition them into canonical, partial, and auxiliary HORs), the problem that remains unsolved despite multiple studies in the last four decades [44], [116], [120], [20], [121], [28], [117], [42]. However, the challenge of properly defining the set of all human monomers remained outside the scope of the StringDecomposer tool. Although [28] presented a large set of human monomers, it is unclear if this set is compatible with the CE postulate. As a result, it remains unclear how to computationally define the complete set of monomers (a prerequisite for launching StringDecomposer) and HORs in human centromeres.

Previous semi-manual studies inferred many HORs and greatly contributed to our understanding of the architecture of human centromeres [116], [117]. However, they did not specify an "algorithmically constructive definition" of a HOR. Instead, an order of monomers in a consensus HOR was implicitly defined as the "ancestral order" without specifying how to derive this order

Table 4.1. Comparison of methods for monomer/HOR inference and annotation

Method	Objective for HOR inference	Compliant with CE Postulate	Automated
HORdetect [20]	-	**?**	**+**
ColorHOR [120]	**-**	**?**	**+**
Alpha-CENTAURI [28]	-	?	+
GlobalRepeatMap [123]	-	?	+
CentromereArchitect [47]	+	-	+
T2T [110]	-	+	-
HORmon (this study)	+	+	+

and how to prove that it is correct and unique. Although [120], [20], [28] described various HOR inference heuristics (ColorHOR, HORdetect, and Alpha-CENTAURI, respectively), these studies have not specified the exact objective function for HOR inference (Table 4.1). As such, the concept of a HOR is highly dependent on the parameters used for generating the monomer set. Moreover, the nucleotide sequences for human HORs of live human centromeres have been manually extracted at the dawn of the sequencing era and used reads (often sampled from a single clone from a specific centromere) rather than completely assembled centromeres, raising questions about their accuracy [35], [122]. For example, HOR DXZ1 (S3CXH1L) on cenX, the first inferred human HOR, was derived based on limited sequencing data from a single clone [115]. The sequence of this HOR differs from the HOR extracted from the complete cenX assembly, suggesting that either (1) reads used for deriving DXZ1 were limited to a small region of cenX that does not adequately represent the entire centromere, or (2) HORs extracted from different individuals may be different.

These limitations prevent future evolutionary studies of centromeres across multiple species. Addressing them is important because long and accurate Pacific Biosciences (PacBio) HiFi reads have already been used for centromere assembly in fish [124] and because various HiFi assembly projects are currently underway, opening a possibility to assemble vertebrate centromeres in the near future. On the other hand, the Telomere-to-Telomere (T2T) Consortium and the Human Pangenome Reference Consortium (HPRC) are now assembling centromeres from

multiple humans. Because their manual annotation (including monomer and HOR inference) is prohibitively time-consuming, automated annotation is a prerequisite for any centromere analysis in the future.

[114] developed the CentromereArchitect tool that addressed the monomer and HOR inference as two separate problems. In particular, the HOR inference was addressed as a data compression problem rather than an evolutionary problem that takes into account the CE postulate. Thus, although CentromereArchitect enabled an automated inference of monomers, it remains unclear whether its HORs inference adequately reflects the centromere evolution. Our analysis revealed that to generate a biologically adequate centromere annotation, the monomer and HOR inference should be viewed as two interconnected problems in such a way that HOR (monomer) generation affects monomer (HOR) generation.

In the past, the monomer generation problem was addressed as clustering of monomer blocks without considering the follow-up inference of HORs derived from the resulting monomers [114]. Because this is a complex clustering problem, any clustering algorithm may merge some biologically distinct monomer blocks into a single cluster and split a single cluster into multiple ones. Another complication is the inference of hybrid monomers that by definition do not participate in canonical HORs.

Below, we describe the HORmon algorithm that addresses these complications by incorporating the monomer and HOR generation into a single pipeline (Fig. 4.2). HORmon generated the first automated centromere annotation that is largely consistent with the CE postulate and previous manual centromere annotations. Recognizing that HORs represent an important evolutionary concept, we show how HORmon can be used to automatically derive the currently known HORs.

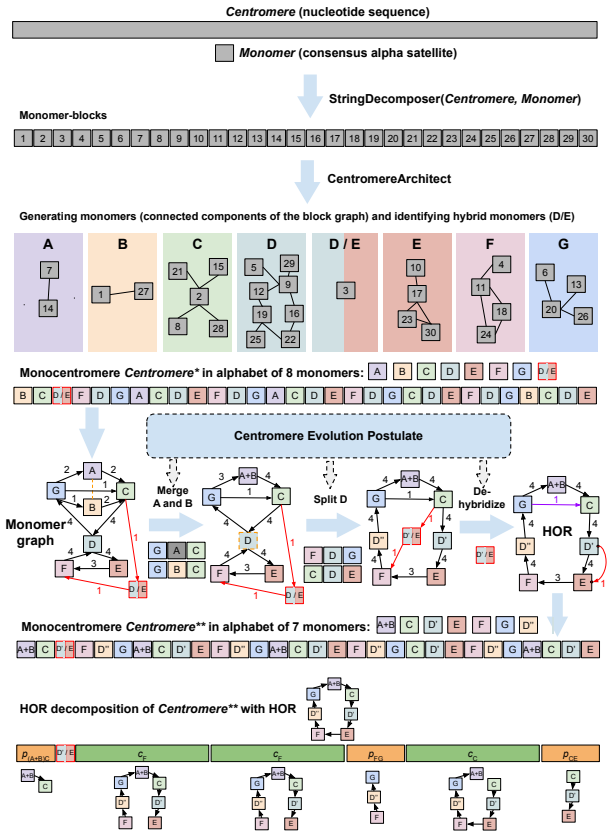


Figure 4.2. HORmon pipeline. Given the nucleotide sequence *Centromere* and a consensus alpha satellite sequence *Monomer*, HORmon iteratively launches StringDecomposer [47] to partition *Centromere* into monomer blocks. After each launch of StringDecomposer, HORmon launches CentromereArchitect [114] to cluster similar monomer blocks into monomers, identify hybrid monomers (represented by a single hybrid D/E of monomers D and E), and transform *Centromere* into the monocentromere *Centromere**. Afterward, HORmon uses the generated monocentromere to construct a monomer graph (red edges connect the hybrid monomer D/E with the rest of the monomer graph). To comply with the centromere evolution postulate, HORmon performs split/merge transformations and dehybridizations on the initial monomer set. The orange dotted undirected edge connects similar monomers A and B to indicate that they represent candidates for merging. The breakable monomer D is shown as a dotted vertex to indicate that it is a candidate for splitting into monomers D' and D''. The dehybridization substitutes the hybrid vertex D'/E by a single red edge that connects the prefix of D' with the suffix of E. Split, merge, and dehybridization operations result in a new monomer set and transform *Centromere* into the monocentromere *Centromere***. The black cycle in the monomer graph of *Centromere** represents the HOR; the purple edge connecting monomers G and C is a low-frequency chord in this cycle. HORmon uses this HOR to generate the HOR decomposition of *Centromere** into the canonical (c_F, c_C), partial ($p_{(A+B)C}$, p_{FG}, p_{CE}), and auxiliary (the single block D'/E) HORs. c_F and c_C refer to traversing the (canonical) HOR starting from monomers F and C, respectively. $p_{(A+B)C}$, p_{FG}, and p_{CE} refer to partial traversals of the HOR from monomer A + B to C, from F to G, and from C to E, respectively.

4.3 Results

4.3.1 A brief description of the HORmon algorithm

Figure 4.2 illustrates the various steps of the HORmon algorithm for monomer and HOR inference. Supplemental Note 2 in [118] summarizes the notation that we use throughout the paper.

4.3.2 Data sets

We extracted the alpha satellite arrays from the assembly (public release v1.0) of the effectively haploid CHM13 human cell line constructed by the T2T Consortium [34], [84], [110], [85]. We also extracted the alpha satellite array of the newly assembled centromeres of Chromosome X and Chromosome Y from the HG002 cell line sequenced by the HPRC. For simplicity, we refer to these two genomes as the CHM13 and HG002 genomes. Supplemental Note 3 in [118] provides information about the extracted regions for all live human centromere arrays.

4.3.3 Monomer inference

HORmon launches CentromereArchitect [47] to generate the initial monomer set and further modifies it by using the monomer-HOR feedback loop described in Methods (Fig. 4.2). Because all chromosomes considered in this study except Chromosome Y originated from the CHM13 cell line, we launch HORmon three times: on centromeres that originated from the CHM13 cell lines, on Chromosome X from the HG002 cell line, and on Chromosome Y from the HG002 cell line. Supplemental Note 4 in [118] describes how HORmon assigns names to monomers and provides correspondence between these names and the traditional names described in [42].

Because CentromereArchitect identifies many infrequent monomers, comparing its monomer set with the previously identified monomer sets, for example, the monomer set

MonomersT2T [110] used by the T2T Consortium (based on the monomer set derived in [121], [42], is not straightforward. HORmon thus filters the monomer set generated by CentromereArchitect as described below.

We refer to the set of frequent monomers obtained from CentromereArchitect as MonomersNew. Supplemental Note 5 in [118] describes the procedure for construction of *MonomersNew* and shows that it provides a minor improvement over the (manually constructed) *MonomersT2T* monomer set with respect to standard clustering metrics. However, as with any clustering approach, the parameter-dependent CentromereArchitect may both split and aggregate monomers as compared to the biologically adequate clustering. Moreover, the monomer set *MonomersT2T* does not attempt to solve the monomer inference problem that CentromereArchitect addresses [114]. Instead, it generates clustering that is consistent with CE postulate, which can be suboptimal with respect to standard clustering metrics that do not take into account any evolutionary assumptions.

4.3.4 The challenge of monomer generation

Although it is unclear what is a biologically adequate clustering of monomer blocks, positional information about these blocks (i.e., pairs, triples, etc., of consecutive monomers in the monocentromere) often reveals monomers that were erroneously split/aggregated. This positional information helps one to generate a more adequate monomer set with respect to the CE postulate, not unlike the positional information about orthologs in comparative genomics studies [125]. Two monomers are called "similar" if the percent identity between them exceeds a threshold *minPI* (default value 94%). In the subsection "Positionally similar monomers" (Fig. 4.2), we define the concept of positional similarity and classify two similar monomers as "positionally similar" if their positional similarity exceeds a threshold *minPosSim* (default value 0.4).

To illustrate the challenge of generating a biologically adequate clustering, we consider similar frequent monomers M' and M'' from the monomer set *Monomers* that would be merged into a single monomer if the clustering parameters were slightly relaxed. Because it is unclear

how to select clustering parameters, it is also unclear whether such merging would represent a biologically adequate clustering as opposed to the clustering that separates these monomers. However, one may argue that if M' and M'' are always flanked by the same frequent monomers X and Y in a monocentromere (resulting in triples $XM'Y$ and $XM''Y$), these two monomers are likely erroneously split and should be merged into a single monomer M, defined as the consensus of all M' blocks and M'' blocks. Such merging is justified from the perspective of the CE postulate because each nonhybrid monomer occurs exactly once in a HOR. Specifically, unless monomers M' and M'' are merged, the HOR cannot traverse monomers X and Y exactly once as required by the CE postulate.

On the other hand, a frequent monomer M that is flanked either by frequent monomers X' and Y' (resulting in a triple $X'MY'$) or by different frequent monomers X'' and Y'' (resulting in a triple $X''MY''$) conflicts with the CE postulate. Because this monomer is likely erroneously aggregated from two different monomers, it can be split into monomers M' and M'', resulting in triples $X'M'Y'$ and $X''M''Y''$, respectively. The monomers M' (M'') can be defined as the consensus of all M' blocks (M'' blocks) in triples $X'M'Y'$ ($X''M''Y''$).

Although such transformations are not necessarily justified with respect to optimizing the standard clustering metrics, Supplemental Note 5 in [118] illustrates that the monomer set transformed by merging/splitting operations in HORmon is largely comparable to the monomer set generated by CentromereArchitect with respect to various clustering metrics.

In addition to generating the monomer set, CentromereArchitect includes a HOR inference algorithm based on iteratively defining the units as the "heaviest" substrings of a monocentromere [114]. Although this definition is adequate from the perspective of data compression, it does not necessarily reflect the evolutionary history of a centromere (although many resulting units correspond to canonical, partial, and auxiliary HORs). Moreover, [114] derived monomers independently from HORs without accounting for hybrid monomers, positional information, and the CE postulate. Below, we show that positional information, as well as information about hybrid monomers, is important for both monomer and HOR inference. The Methods section

describes how to identify erroneously aggregated/split monomers and split/merge them.

We further introduce the concept of a "breakable" monomer, that is, a monomer that is amenable to splitting into two or more monomers in such a way that the enlarged monomer set still adequately represents the centromere architecture. In contrast, splitting an unbreakable monomer results in an inadequate representation of the centromere architecture. We show that a series of split and merge operations results in unbreakable monomers for cen1, cen13, and cen18 that prevent HORmon from reporting HORs in these centromeres. We further describe a special procedure for splitting unbreakable monomers in these problematic centromeres (subsection "Splitting unbreakable monomers reveals HORs in cen1, cen13, and cen18").

Split and merge operations on the monomer set MonomersNew result in a monomer set MonomersNew^{+}, whereas further hybridization of hybrid monomers (Fig. 4.2) and splitting of unbreakable monomers result in the monomer set MonomersFinal described in Supplemental Table S1 in [118].

4.3.5 Monomer graph

Given a monocentromere, its directed "monomer graph" is constructed on the vertex set of all its monomers and the edge set formed by all pairs of its consecutive monomers. The "multiplicity" of an edge (M, M') in the monomer graph is defined as the number of times the monomer M' follows the monomer M in the monocentromere (Fig. 4.2). We note that the monomer graph of a monocentromere $Centromere^{*}$ is the "de Bruijn graph" DB($Centromere^{*}$, 2) [126]. Figure 4.3 presents the monomer graph for cenX in the CHM13 genome (top) and the HG002 genome (bottom) built using the monomer set extracted by CentromereArchitect from CHM13 genome [114]. Both graphs reveal the cycle formed by 12 high-multiplicity edges that form the canonical 12-monomer HOR in cenX. In addition, the monomer graph for CHM13 reveals two infrequent hybrid monomers and 10 low-multiplicity edges. In contrast, the monomer graph for HG002 reveals only one infrequent hybrid monomer and only five low-multiplicity edges. These differences suggest that hybrid monomers represent a rather recent evolutionary

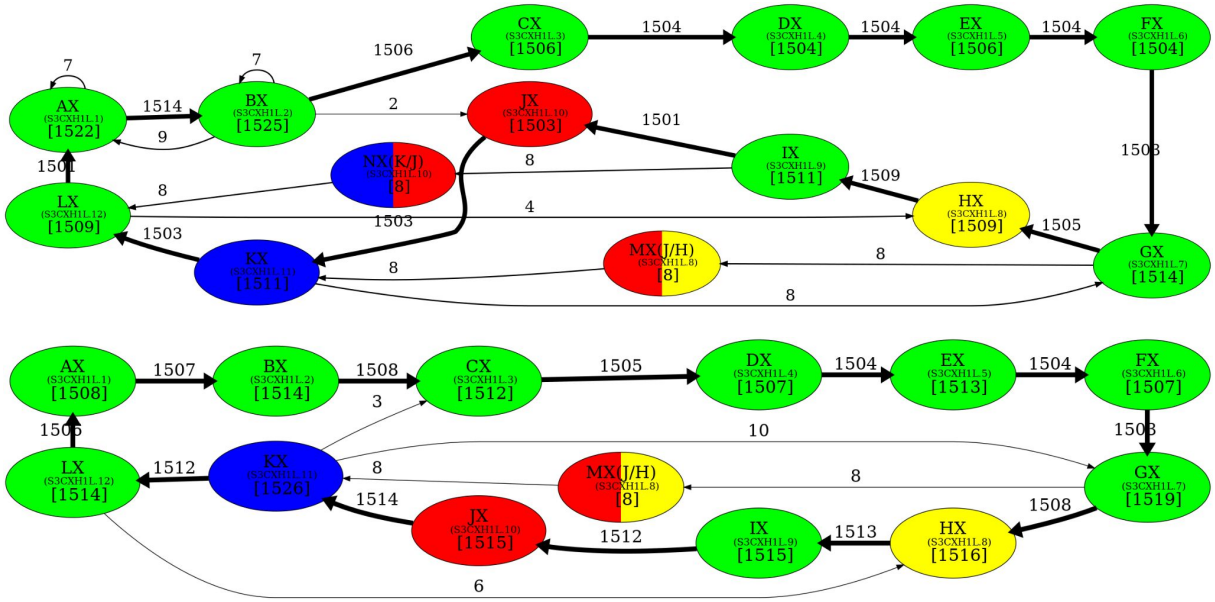


Figure 4.3. The monomer graph of cenX in the CHM13 (top) and HG002 (bottom) genomes. The monomer graphs of cenX were constructed on the monocentromere that was generated from the monomer sets consisting of two infrequent hybrid monomers (labeled as MX and NX) and 12 frequent canonical monomers (labeled as AX, BX, CX, ..., KX, and LX) that contribute to the canonical DXZ1 HOR in cenX [114]. Small font corresponds to the naming conventions introduced in [121]. The hybrid monomers M and N are inferred in [47]. A hybrid monomer formed by frequent monomers X and Y is represented as a bicolored vertex (two colors correspond to the colors of X and Y) and is denoted as (X/Y). Only edges of the monomer graph with multiplicity exceeding 1 are shown (edges with multiplicity exceeding 100 are shown in bold). The cycle formed by bold edges (with multiplicities above 1500) traverses the 12 most frequent monomers that form the canonical cenX HOR.

innovation and illustrate large variations in centromeres across the human population.

Figure 4.3 creates a false impression that simply ignoring the low-multiplicity edges and hybrid monomers in the monomer graph of a centromere would result in a graph with a single cycle that forms a HOR. Although this is indeed true for centromeres 3, 11, 14, 16, 17, 19, 20, 21, 22, X, and Y (after performing a series of split/merge transformations on the monomer set generated by CentromereArchitect) [114], the remaining human centromeres have a more complex evolutionary history, resulting in complex architectures that we analyze below.

4.3.6 Monomer graphs of human centromeres

Given a monomer set *Monomers* and a monocentromere $Centromere^{*}$, we define *minCount(Monomers)* $\min_{M \in Monomers} count(M, Centromere^{*})$. HORmon uses the set $MonomersNew^{+}$ to generate the monocentromere $Centromere^{**}$ (split and merge operations on the monomer set *MonomersNew* result in the monomer set $MonomersNew^{+}$), generates the monomer graph as the de Bruijn graph DB(Centromere**, 2), and removes edges that have multiplicity below $\min(MinEdgeMultiplicity, minCountFraction \times minCount(MonomersNew^{+}))$ with the default values $MinEdgeMultiplicity = 100$, $minCountFraction = 0.9$ (Fig. 4.2). Supplemental Figure S1 in [118] provides information about the generated monomer graphs for all human centromeres.

The monomer graphs of 10 centromeres (3, 11, 14, 16, 17, 19, 20, 21, 22, X, and Y) are formed by cycles that immediately reveal HORs. The monomer graph for cen17 contains two cycles: the higher-multiplicity cycle corresponds to the D17Z1 HOR, whereas the lower-multiplicity cycle corresponds to its sister HOR D17Z1-B (for discussion of sister HORs, see [122]). The remaining monomer graphs contain (albeit implicitly) information about HORs but represent a more detailed view of the evolutionary history of centromeres. To reveal HORs in these monomer graphs, HORmon constructs simplified monomer graphs described in Methods.

Figure 4.4 shows that the simplified monomer graphs represent cycles (corresponding to HORs) for all centromeres but centromeres on Chromosomes 1, 5, 8, 9, 13, and 18 that do not have Hamiltonian cycles and represent special cases that we consider below.

4.3.7 Splitting unbreakable monomers reveals HORs in cen1, cen13 and cen18

A monomer is breakable if it is amenable to splitting into two or more monomers in such a way that the enlarged monomer set still adequately represents the centromere architecture (Methods). In contrast, splitting an unbreakable monomer leads to conflicts and results in an inadequate representation of the centromere architecture. Even if a monomer is breakable, splitting it into two very similar monomers (e.g., monomers M' and M'' that differ in a single position)

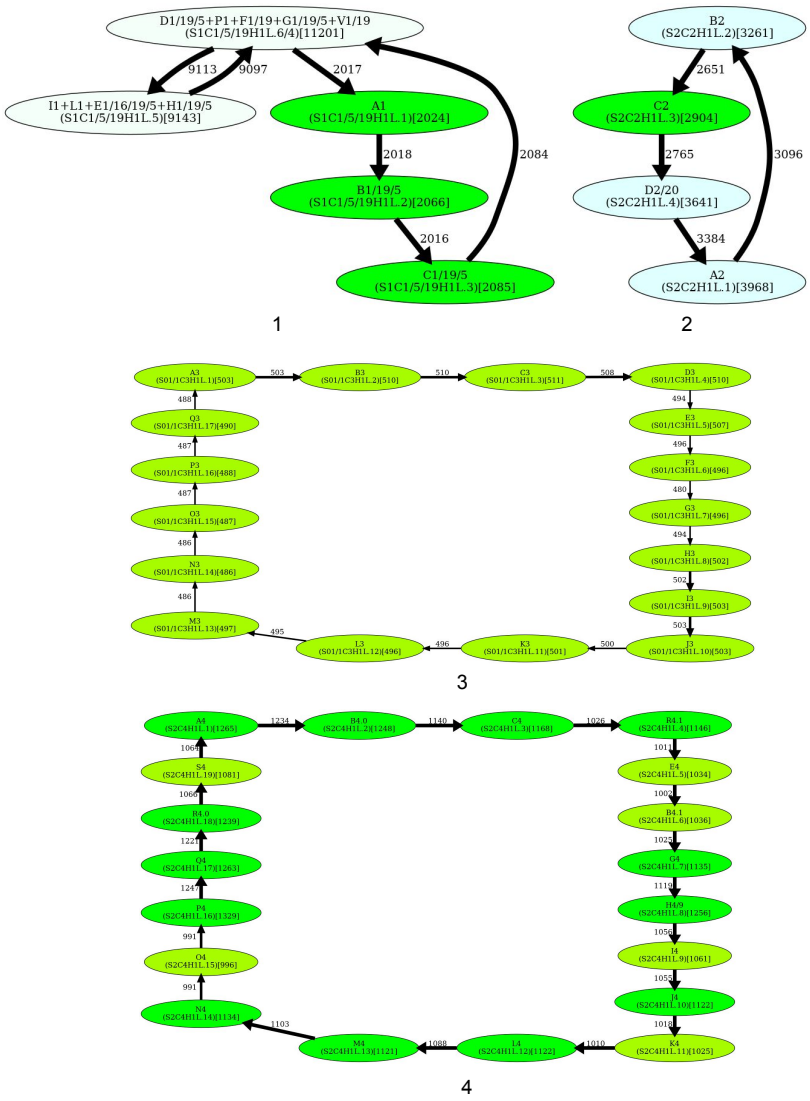


Figure 4.4. The simplified monomer graphs of human centromeres. The first 23 subfigures contain simplified monomer graphs for all live human centromeres in the CHM13 cell line (centromere ID shown in the subcaption). The 24th subfigure corresponds to the centromere on Chromosome Y in the HG002 genome. In each graph, vertices represent the monomer set *Monomers* of the corresponding *Centromere*. The label of each vertex represents the monomer ID and its count in the monocentromere $Centromere^*$ (in parentheses). The ID of the monomers follow the naming convention introduced in [121]. Two monomers are connected by an edge if they are consecutive in monocentromere $Centromere^*$. The weight of an edge connecting monomers M and M' is defined as the number of times M is followed by M' in $Centromere^*$. The width of an edge (color of a vertex) reflects its multiplicity (count of a monomer). In each graph, HORmon detects heavy nonoverlapping cycles and paths and removes chords in such cycles (for details, see Methods). The isolated cycles in 18 centromeres (2, 3, 4, 6, 7, 10, 11, 12, 14, 15, 16, 17, 19, 20, 21, 22, X, and Y) represent HORs in these centromeres. *(Figure continues on following pages.)*

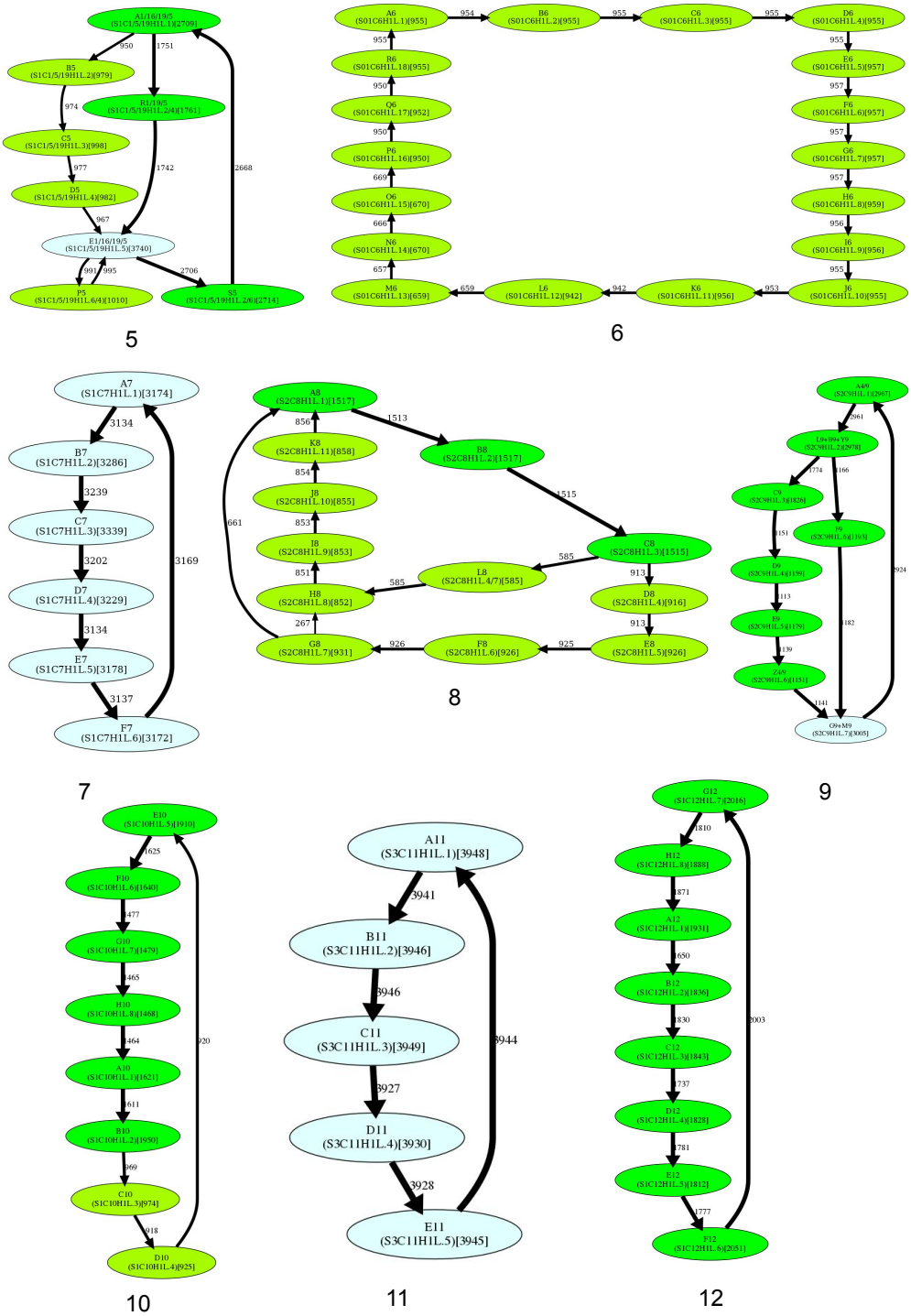


Figure 4.4. The simplified monomer graphs of human centromeres. *(Figure continues on following pages.)*

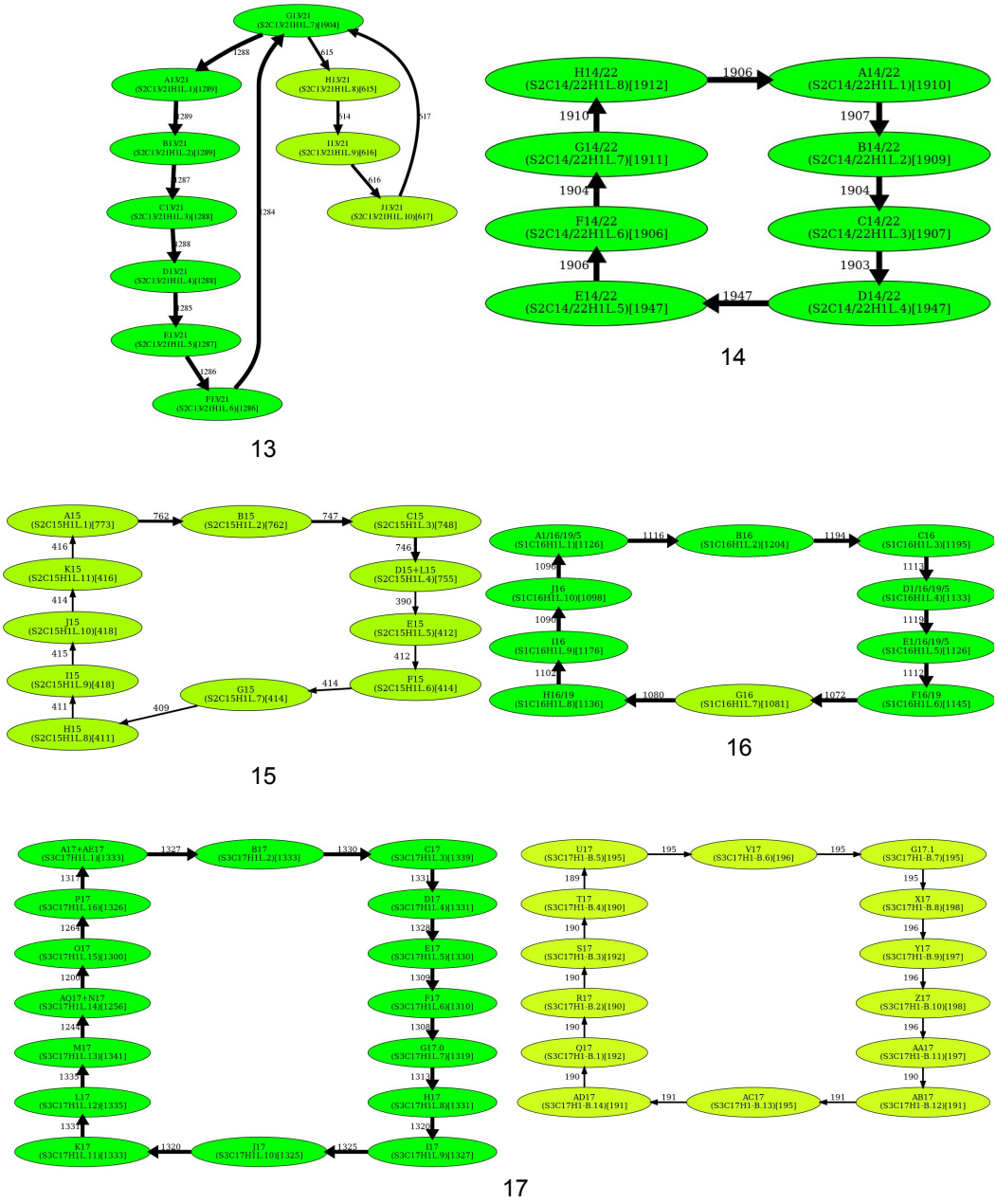


Figure 4.4. The simplified monomer graphs of human centromeres. *(Figure continues on following pages.)*

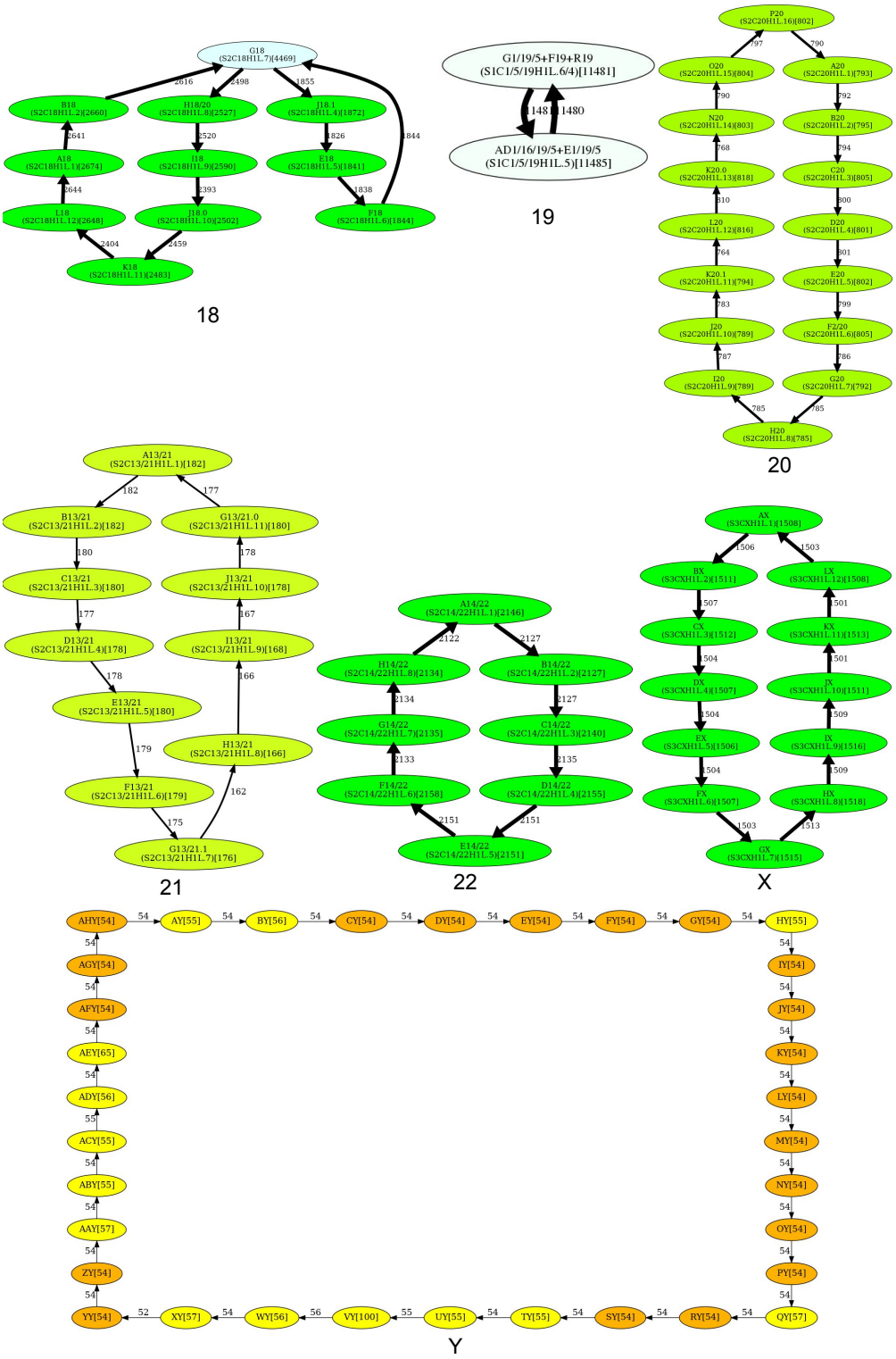


Figure 4.4. The simplified monomer graphs of human centromeres.

may lead to a misclassification of monomer blocks because all centromere decomposition tools, including StringDecomposer, often misclassify an M' block as a very similar M'' block and vice versa. Such misclassified monomer blocks may lead to downstream challenges in analyzing centromere architecture and evolution.

Although the simplified monomer graphs of cen1, cen13, and cen18 are formed by two cycles that share a junction vertex (that deviate from the definition of a HOR as a single cycle), these two cycles can be transformed into a single cycle by splitting the junction vertices (Fig. 4.5). However, because these junction vertices correspond to unbreakable monomers, splitting them raises concerns. Indeed, it either conflicts with some frequent traversals through the junction vertex or results in a pair of highly similar monomers that would be merged into a single monomer even under extremely stringent values of HORmon parameters.

Splitting a junction monomer in cen1 results in two monomers that differ in 11 nt. This transformation results in a simplified monomer graph that contains a cycle that corresponds to a HOR and a dimer formed by two high-multiplicity anti-parallel edges (Fig. 4.5). In fact, this dimer was originally reported as a HOR in cen1 [127], [116], [117].

Splitting a junction monomer in cen13 (cen18) results in two monomers that differ in only 3 (1) nt. The split of the unbreakable vertex G into vertices G.0 and G.1 results in two traversals F-G.0-H and J-G.1-A (Fig. 4.5). Further launch of StringDecomposer (using monomers G.0 and G.1 instead of G) confirms that there are no traversals F-G.0-A and J-G.1-H.

Splitting a junction monomer in cen18 results in two nearly identical monomers that differ in a single nucleotide and raises a concern about the applicability of the CE postulate to cen18. Splitting the unbreakable monomer G in cen18 should result in two traversals B-G.0-J and F-G.1-H. However, the further launch of StringDecomposer shows 547 B-G.1-J traversals and 10 F-G.0-H traversals. Importantly, in all B-G.1-J (F-G.0-H) traversals, the monomer block G.1(G.0) is more similar (or even identical) to the monomer G.1(G.0). Although this raises a concern about the validity of splitting the unbreakable monomer G in cen18, we proceed with the split to be consistent with the CE postulate.

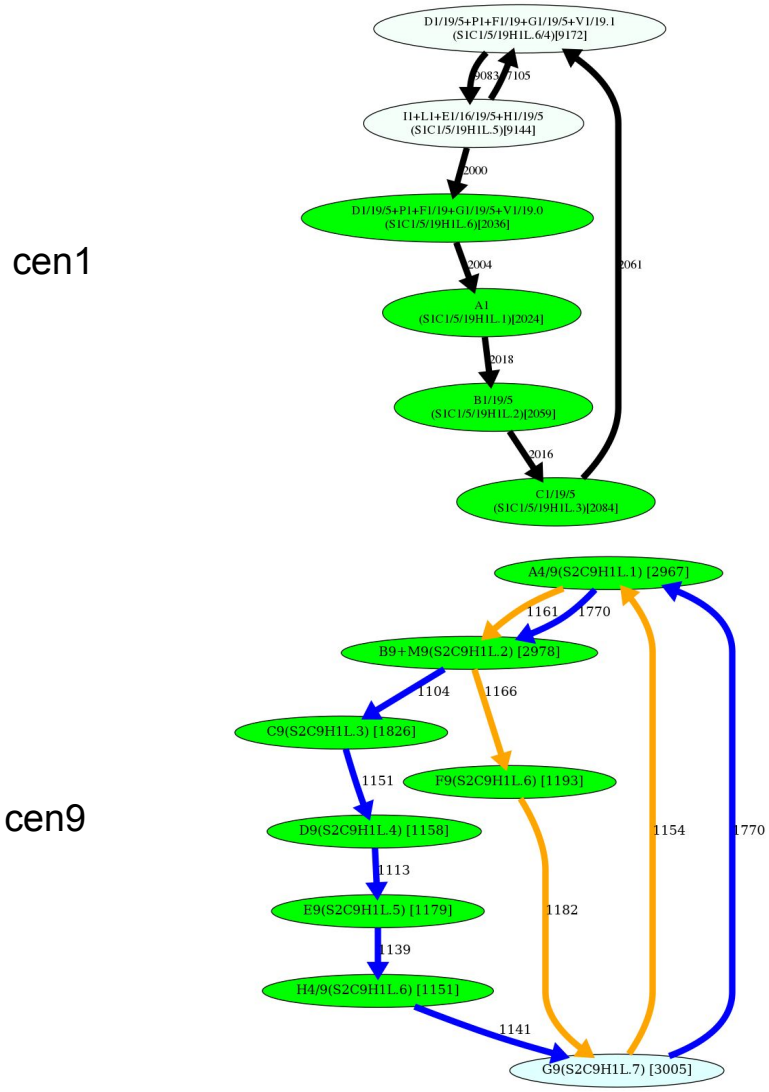


Figure 4.5. Inferring HORs for cen1, cen9, cen13, and cen18. (First row) Splitting an unbreakable junction monomer in cen1 results in two monomers with an 11-nt difference and transforms the monomer graph of cen1 into a cycle with a single chord. (Second row) The manually inferred HOR of cen9 [117], shown as the blue cycle, is in conflict with the CE postulate because the frequently traversed yellow cycle contains a monomer that does not belong to the blue cycle. (Third row) Splitting an unbreakable junction monomer in cen13 results in two similar monomers with an only 3-nt difference and transforms the monomer graph of cen13 (Fig. 4.4) into a cycle with a single chord shown on the left. The resulting simplified monomer graph (shown on the right) reveals the canonical 11-monomer HOR in cen13. (Fourth row) Splitting an unbreakable junction monomer in cen18 results in two monomers with only a single-nucleotide difference and transforms the simplified monomer graph of cen18 (Fig. 4.4) into a cycle with three chords (shown on the left). The resulting simplified monomer graph (shown on the right) reveals the canonical 12-monomer HOR in cen18. *(Figure continues on the following page.)*

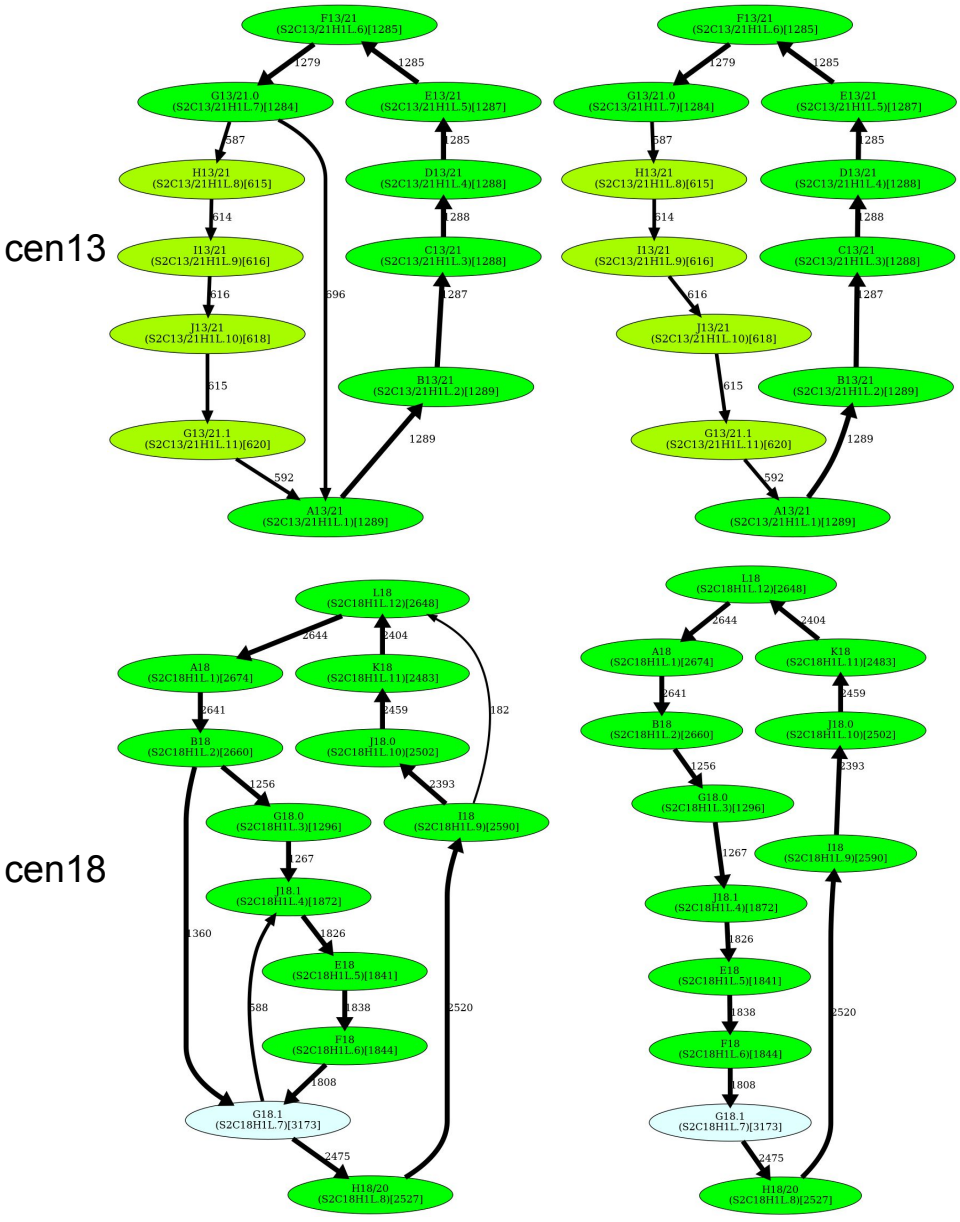


Figure 4.5. Inferring HORs for cen1, cen9, cen13, and cen18.

4.3.8 Dehybridization reveals HORs in cen5 and cen8

We identified all hybrid monomers (among monomers in $MonomersNew^{+}$ across all centromeres) using the approach described in Methods ("Inference of hybrid monomers"). This analysis revealed only three frequent hybrid monomers: P5, R1/5/19, and L8. Below, we describe the "dehybridization" operation on monomer graphs that reveals HORs in cen5 and cen8.

Dehybridization in cen5

P5 is a hybrid monomer of S5 and D5 that differs from S5(50)/D5(120) in 5 nt, whereas R1/5/19 is a hybrid monomer of B5 and D5 that differs from B5(92)/D5(78) in 6 nt. Figure 4.6 (top) illustrates that dehybridization of P5 and R1/5/19 results in a graph with a single Hamiltonian cycle that is classified as a HOR.

Dehybridization in cen8

L8 is a hybrid monomer of D8 and G8 which differs from the consensus D8(60)/G8(111) by only 2 nt (Supplemental Note 4 in [118]). Figure 4.6 (bottom) illustrates the dehybridization of L8 that models it as a hybrid edge of the monomer graph, resulting in a graph with a single Hamiltonian cycle (and two chords) that is classified as a HOR.

4.3.9 What is a HOR in cen9?

Splitting unbreakable junction vertices (cen1, cen13, and cen18) and dehybridization (cen5 and cen8) reveal HORs for all centromeres except for cen9. This centromere represents a difficult case from the perspective of the CE postulate because it is unclear how to infer a HOR from the monomer graph of this centromere. The blue traversal of this graph (Fig. 4.5) corresponds to the currently known (manually inferred) HOR. The monomer F9 (that does not belong to the HOR in cen9) cannot be represented as a hybrid monomer and is quite different from its most similar monomer in cen9 (it differs from Z4/9 by 12 nt). Thus, it is not clear how to automatically derive a HOR for cen9.

One can argue that merging monomers F9 and Z4/9 would reveal a Hamiltonian cycle

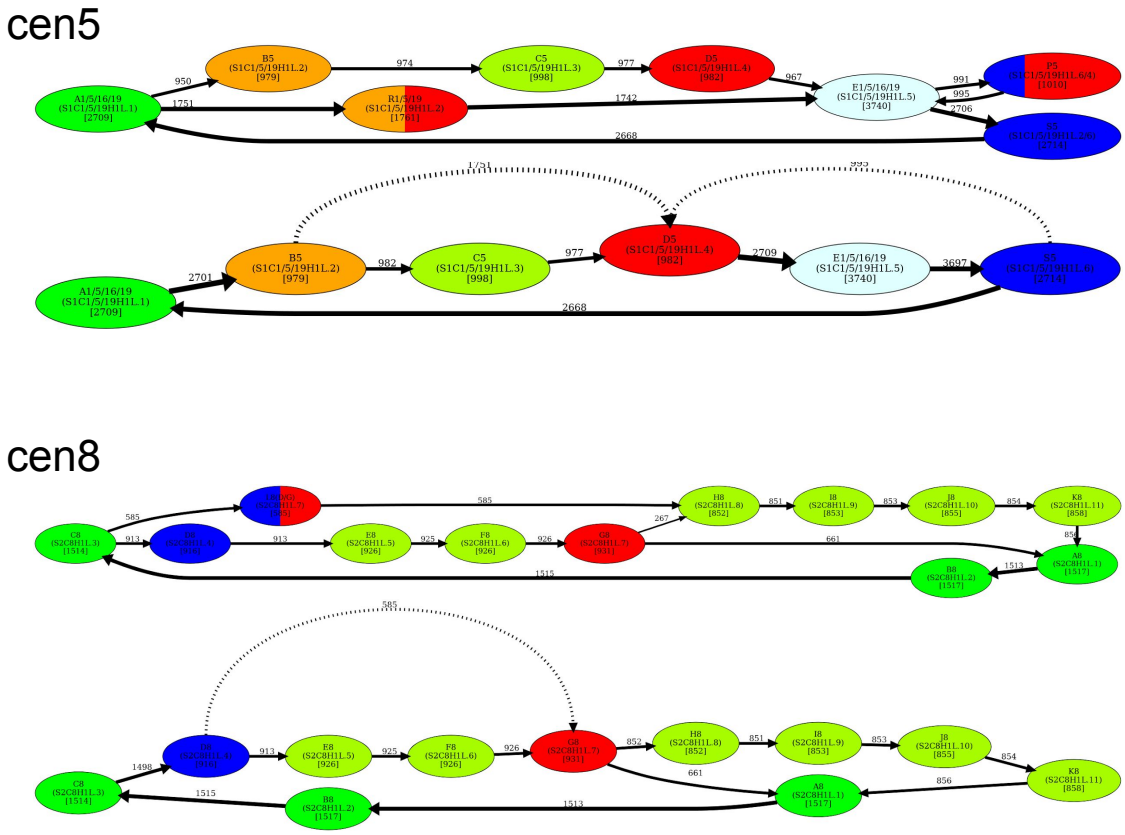


Figure 4.6. Dehybridization substitutes hybrid vertices (monomers) by hybrid edges in the monomer graph. (Top) Dehybridization of P5 and R1/5/19 in cen5. (Bottom) Dehybridization of L8 in cen8.

(HOR) in the resulting monomer graph, thus extending the CE postulate to cen9. This argument reflects the difficulty of developing an automated approach to centromere annotation and defining parameters of these approaches that work across all centromeres. Indeed, the CE postulate is highly dependent on parameters; for example, relaxing the parameter for monomer merging will affect the monomer graphs for all centromeres and may "break" the CE postulate for some of them. Although by manually fitting parameters for each centromere, one can make it look consistent with the CE postulate, such an approach does not represent solid supporting evidence for this postulate. As described in Supplemental Note 6 in [118], because of the limited data (only a single human genome has been completely assembled so far), it is challenging to avoid overfitting even for the default parameters of HORmon, let alone for a more complex procedure. Our approach represents the first automated analysis (with the same parameters for all centromeres) demonstrating that the CE postulate holds for nearly all centromeres. Subsection "Limitations of the CE postulate" (Methods) highlights further limitations of the CE postulate.

4.3.10 Generating the centromere decomposition into HORs

HORmon decomposes each monocentromere into canonical, partial, and auxiliary HORs as described in subsection "Decomposing a centromere into HORs" in Methods (Fig. **??**). Given a canonical HOR $H = M_1, \ldots, M_n$, each canonical HOR $M_i, \ldots, M_n, M_{n+1}, \ldots, M_{i-1}$, in the decomposition is labeled as c_i. We use the notation c_i^m to denote m consecutive occurrences of a canonical HOR and refer to each such element in the HOR decomposition as a "HOR run." According to the CE postulate, hybrid and infrequent monomers do not belong to the HOR. Supplemental Note 7 in [118] discusses the advantages of the HORmon approach over more traditional methods.

The "length" of the HOR decomposition is defined as the total number of elements in this decomposition (each entry x^y is counted as a single element). Figure 7 in [118] shows the HOR decompositions of cenX under the assumption that the monomer set includes 12 monomers AB…KL forming the HOR on cenX, as well as hybrid monomers M and N identified

in [47]. Supplemental Table S2 and Supplemental File 1 in [118] provide information about the HOR decompositions for all human centromeres. Supplemental Note 8 in [118] describes how these HOR decompositions are used to generate the nucleotide consensus of each HOR. Supplemental Note 4 in [118] summarizes information about these consensuses for live human centromeres. Because these consensus sequences are computed for the first time using a complete human genome assembly, they characterize the CHM13 cell line more accurately than previously inferred sequences. The question of whether they are representative for other individuals remains open.

Pairs of centromeres (13, 21) and (14, 22), as well as triple of centromeres (1, 5, 19), have been reported to share the same HOR [117]. Contrary to previous studies, we conclude that HORs in these centromeres are rather different, at least in the CHM13 cell line. The edit distance between the consensus of HORs in cen13 and cen21 is rather high (20 differences, 1% divergence), whereas the edit distance between the consensus of HORs in cen14 and cen22 is much lower (three differences, 0.2% divergence). Previous studies reported two frequent nonhybrid monomers for centromeres 1, 5, and 19 [117]. We report six frequent nonhybrid monomers for cen1 and cen5, and two for cen19. We hypothesize that these differences are a result of the absence of a complete genome assembly in prior studies. Sequence comparison shows that the edit distance between the consensus of HOR in cen1 and cen5 is large (34 differences, 3.3% divergence).

4.4 Discussion

Recent advances in long-read sequencing technologies and genome assembly algorithms opened new horizons for centromere genomics. For the first time, studies of human alpha satellite arrays can be based on a complete centromere assembly rather than individual reads or "satellite reference models" [31]. The development of an automated centromere annotation tool is a prerequisite for future centromere research that quickly moves to the stage when the complete

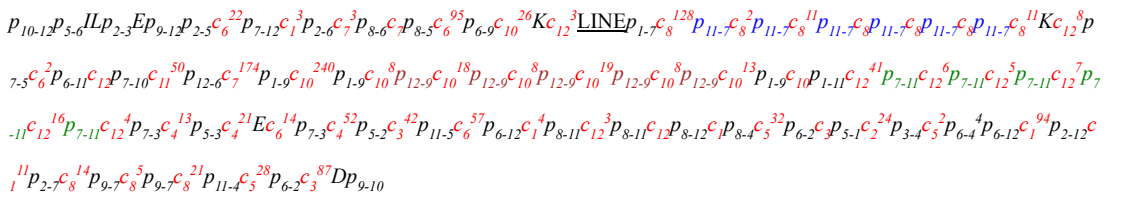


Figure 4.7. Decomposition of cenX into HORs. The 12-monomer HOR for cenX is represented as M1… M12 = AB…KL. The monomer set includes these 12 frequent monomers as well as hybrid monomers M (a hybrid of monomers J and H) and N (a hybrid of monomers K and J) identified in [47]. Each occurrence of this HOR that starts from the monomer M_i is labeled as c_i (shown in red). Each occurrence of a partial HOR that includes monomers from i to j is labeled as $p_{i,j}$. We use the notation c_m (p_m) to denote m consecutive occurrences of a canonical (partial) HOR. The most frequent partial monomers p_{3-7}, p_{7-3}, and p_{5-2} in cenX are colored in blue, green, and brown, respectively. The HOR decomposition of cenX has a length 72 and includes 1486 complete HORs that form 34 HOR runs. Only 257 of 18,089 (1.4%) monomer blocks in cenX are not covered by complete HORs. The "LINE" entry shows the position of the LINE element. To ensure that all monomers are shown in the forward strand, we decompose the reverse complement of cenX and take reverse-complements of all monomers in cenX (Supplemental Note 4 in [118]).

genomes of hundreds of individuals will be assembled. These studies include population-wide analysis of human monomers and HORs, evolutionary studies of centromeres across primates and other species, and biomedical studies of diversity of human centromeres and their associations with genetic diseases.

We developed HORmon, the first annotation tool for live alpha satellite arrays that considers monomer and HOR inference as two interconnected problems and automatically generates the monomer and HOR set that mirror the four decades of centromere research. HORmon not only provides the first automatic procedure for extracting monomers and HORs in live alpha satellite arrays but also establishes their nucleotide consensus sequences. This is important because the currently used nucleotide sequences for many of these monomers and HORs have been extracted more than two decades ago [116] in the absence of centromere assemblies. In centromeres 1, 2, 5, and 15, HORmon reported a different number of monomers than [117]. We hypothesize that these differences result from the absence of a complete genome assembly in

prior studies. Contrary to previous studies, we found that HORs in pairs of centromeres (13, 21) and (14, 22) are rather different ("Generating the centromere decomposition into HORs"). We note that because human centromeres are very divergent between individuals, it remains unclear how well the inferred nucleotide consensus of HORs in the CHM13 cell line represents other individuals.

HORmon uses a heuristic approach for monomer and HOR inference rather than popular clustering algorithms (such as k-means or hierarchical clustering) because the monomer inference problem differs from the classical clustering problem. For example, the set of data points (monomer blocks) is not explicitly given but is implicitly encoded in the centromere and depends on the selection of centers (monomers). Although the choice of the consensus alpha satellite results in the initial set of monomer blocks, each selection of a monomer set affects this initial set and results in a slightly different clustering problem. Moreover, it is not clear how to select the biologically adequate function to measure the distances between data points and centers. For example, the sequence divergence function that HORmon uses is clearly limited (it does not take into account the positional information), necessitating the merging/splitting modules in HORmon. It is also unclear how to incorporate hybrid monomers in the framework of the classical clustering problems. To address all these complications, we have designed the HORmon heuristic instead of using the standard clustering approaches. Supplemental Note 9 in [118] presents information about time and memory footprint of HORmon.

HORmon introduced a procedure for decomposing a centromere into HORs and generated the UCSC Genome Browser tracks representing this decomposition for the CHM13 genome. Although the recently assembled CHM13 genome does not include Chromosome Y, we project that HORmon will be able to generate monomers and HORs for cenY once its complete assembly becomes available. [42] classified a HOR as "homogeneous (divergent)" if its copies have an average divergence less than 5% (greater than 10%). In addition to live centromeres that we analyzed in this paper, human chromosomes have nearly 60 pseudocentromeric and divergent HOR arrays. Our next goal is to use HORmon for generating monomers and HORs for these

HOR arrays that are still only manually annotated (inferred) in the T2T assembly.

Although HORmon relies on the CE postulate to rationalize the series of splits, merges, and dehybridizations, computational validation of this postulate remains outside the scope of this paper (Supplemental Note 1 in [118]). Indeed, rigorous statistical analysis of the CE postulate (together with formulating and analyzing alternative evolutionary hypotheses) is currently lacking. Because the CE postulate was formed implicitly at the dawn of the sequencing era, we do not rule out a possibility that it might be revised once the statistical significance of HOR extraction for all centromeres is rigorously assessed. In fact, development of HORmon already revealed difficulties of extending CE postulate to cen9 (subsection "What is a HOR in cen9?") and cen18 (subsection "Splitting unbreakable monomers reveals HORs in cen1, cen13, and cen18").

Because only a single human genome remains completely assembled, the selection of HORmon parameters was based on this genome only and thus may suffer from overfitting. Supplemental Note 6 in [118] provides intuition and justification for parameter selection. Moreover, without a rigorous statistical assessment of the CE postulate versus alternative models of centromere evolution [128], [129], [111], it is unclear how to verify that the HORs extracted by HORmon represent the most likely solution of the HOR inference problem. To complicate the issue even further, the existing nucleotide sequences of canonical HORs have been extracted decades ago, limiting the available "ground truth" to benchmark HORmon against. We anticipate that the HORmon pipeline will become an important stepping stone for the development of a fully automatic tool for the extraction of HORs and centromere annotation across the human population once more complete assemblies become available. In fact, [110] already show that extracting monomers and HORs and centromere annotation assists with analysis of CENPA ChIP-seq enrichment and DNA methylation in satellite arrays.

Because the rapidly evolving centromeres are very diverse across the human population [31], [130], we anticipate that the concepts of the monomer graph will assist in comparing centromeres across multiple individuals. Supplemental Notes 10 and 11 in [118] show early application of HORmon to centromeres beyond the human genome. Although HORmon proved

to be useful for analyzing live centromeres, automatic procedures for annotating other alpha satellite domains (both HOR and monomeric) are currently not established. We project that HORmon will work just as well on all homogeneous HORs (not only the live ones). Other HOR arrays however are known to be more divergent than the live arrays, and monomeric arrays are yet more divergent, so it is currently unclear how to universally select HORmon parameters to annotate all alpha satellite arrays in the human genome.

4.5 Methods

4.5.1 Positionally similar monomers

Given a monomer M in a monomer set *Monomers* for a given monocentromere, we identify all triples of consecutive blocks XMY that appear in this monocentromere, and construct the $|Monomers| * |Monomers|$ matrix $Triples_M$, where $TriplesM(X,Y)$ is the count of the number of triples XMY in the monocentromere. We further construct a normalized matrix $NormalizedTriples_M(X,Y)$ by multiplying $Triples_M(X,Y)$ by a constant so that the squared sum of all its entries is equal to one.

Given two equally sized $n \times m$ matrices A and B, we define their similarity as the dot-product of the $n \times m$-dimensional vectors representing these matrices:

$$sim(A,B) = \sum_{i,j} A(i,j) \times B(i,j).$$

Given two monomers M and M', we define their positional similarity $PosSim(M,M')$ as

$$sim(NormalizedTriples_M, NormalizedTriples_{M'}).$$

Two monomers are called "similar" if the percent identity between them exceeds a threshold *minPI* (default value 94%). Two similar monomers are called "positionally similar" if their positional similarity exceeds a threshold minPosSim (default value 0.4).

4.5.2 Merging positionally similar monomers

Because two different positionally similar monomers point to a potentially erroneous splitting of a single monomer, HORmon checks if there are positionally similar monomer pairs in the monomer set *Monomers*. If such monomer pairs exist, it iteratively identifies a pair of the most positionally similar monomers (similar monomers with the highest positional similarity of all similar monomers), merges them into a new monomer, recomputes the consensus of the new monomer, launches StringDecomposer on the new (smaller) monomer set, and iterates until there are no positionally similar monomers left. Similarly to constructing the triple matrices for all triples XMY of a monomer M, HORmon constructs similar matrices for all triples XYM and MXY and merges monomers based on these two matrices in the same way it merges monomers for all triples XMY.

4.5.3 Splitting aggregated monomers

To decide whether to split a monomer M, HORmon analyzes all frequent triples XMY in a monocentromere. Given a monomer M, we refer to the largest element in the matrix $NormalizedTriples_M(X,Y)$ as the "M champion." We classify elements (X,Y) and (X',Y') in the matrix $NormalizedTriples_M(X,Y)$ as "M comparable" if

$$NormalizedTriples_M(X',Y')/NormalizedTriples_M(X,Y)$$

exceeds a "splitting threshold" *splitValue* (default value 1/8). HORmon uses the single linkage clustering to iteratively identify all monomer pairs (X,Y) that are M comparable with the M champion and refer to them as "M-candidate pairs."

Monomer pairs (X,Y) and (X',Y') are called "independent" if all four monomers X,Y,X', and Y' are different. A monomer M that has M-candidate-pairs is called breakable if all M-candidate pairs are (pairwise) independent, and "unbreakable," otherwise. Given a breakable monomer M, HORmon considers all M-candidate pairs $(X_1,Y_1),\ldots,(X_t,Y_t)$ and splits

the monomer M into t monomers $M_1, \ldots, M_t$ by separately deriving the monomers M_i as the consensus of all M blocks that arise from triples $X_i M Y_i$ in the monocentromere for $1 \leq i \leq t$.

Supplemental Note 12 in [118] describes the pseudocode of the SplitAndMerge module that HORmon uses for modifying the initial monomer set.

4.5.4 Simplified monomer graphs

Given a monomer graph, HORmon constructs the "complete bipartite graph"where each part represents all vertices (monomers) of the monomer graph. A monomer M' in the "upper" part is connected with a monomer M'' in the "bottom" part by an edge of the weight equal to the multiplicity of the edge (M', M'') in the monomer graph. Afterward, HORmon solves the "assignment problem" to find the "maximum weight bipartite matching" in the bipartite graph [131]. Edges of this bipartite matching, which also represent edges of the monomer graph, form a set of nonoverlapping cycles and paths in the monomer graph. An edge of a monomer graph is classified as "removable" if it forms a chord in one of these cycles/paths (a chord of a path is defined as an edge connecting two internal vertices of this path). Removal of all removable edges from the monomer graph results in the "simplified monomer graph."

4.5.5 Inference of hybrid monomers

HORmon's algorithm for inferring hybrid monomers differs from the approach in [114]. For monomers A, B, and C, we define $HybridDivergence_A(B, C)$ as the divergence between A and a concatenate of a prefix of B and a suffix of C that is most similar to A. A monomer A from a monomer set *Monomers* is a "hybrid candidate" of monomers B and C if $HybridDivergence_A(B, C)$ is below the *maxResolvedDivergence* threshold and $HybridDivergence_{A(B,C)}$ does not exceed divergence between A and any another monomer from *Monomers*. HORmon first generates a set *HybridCandidates* by iterating over concatenates of all possible prefixes and suffixes for every pair of distinct monomers B and C. Afterward, if there is a single pair of monomers B and C that give rise to a hybrid candidate A, we classify A as a

hybrid of B and C. If several pairs of such monomers exist, we select a pair of monomers B and C that are not hybrid candidates themselves, form a concatenate with the minimal divergence from the monomer A, and classify A as a hybrid of B and C.

4.5.6 Decomposing a centromere into HORs

We defined the monomer graph as the de Bruijn graph with low-multiplicity vertices and edges removed. We now consider the complete de Bruijn graph $DB(Centromere^*, 2)$ and classify an edge in this graph as a "HOR edge" if it connects two consecutive monomers in a HOR, and a "non-HOR edge," otherwise. A monocentromere defines a traversal of edges in the de Bruijn graph (that contains both HOR edges and non-HOR edges) and each non-HOR edge in this traversal corresponds to two consecutive monomers in the monocentromere that we refer to as "breakpoint." We break the monocentromere at all breakpoints defined by non-HOR edges, resulting in multiple short substrings. These substrings, that define the HOR decomposition of a centromere, represent one of the following scenarios:

- a canonical HOR or multiple consequently traversed canonical HORs that may be followed by a partial HOR;
- partial HOR that includes monomers from i to j denoted $p_{i,j}$. Because a HOR is a cycle, i might exceed j, for example, $p_{4,2}$ corresponds to the partial 4-monomer HOR M_4, M_5, M_1, M_2 for a 5-monomer HOR M_1, M_2, M_3, M_4, M_5; and
- auxiliary HOR represented by a hybrid or an infrequent monomer (denoted by the identifier of this monomer).

4.5.7 Limitations of the CE postulate

Figure 4.8 shows a toy example of two "monocentromeres" that result in identical monomer graphs (formed by cycles AB and BC connected via the junction vertex B) yet represent very different evolutionary scenarios. Although one can come up with a plausible

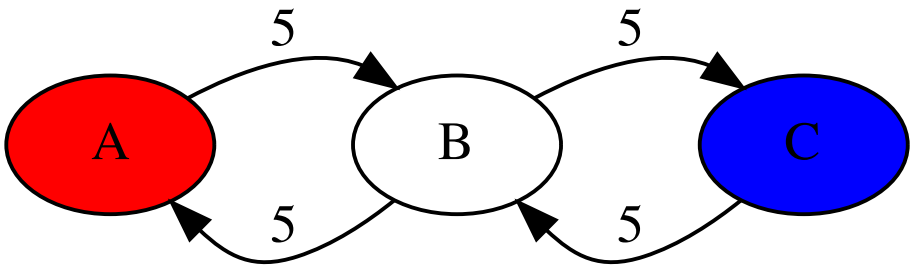


Figure 4.8. Two different "monocentromeres" BABABABABABCBCBCBCBCB and BABCBABCBABCBABCBABCB have the identical monomer graphs.

"evolutionary" scenario for these centromeres, it is not clear how to find out their HORs that would be compliant with the CE postulate. The first monocentromere can be described as two cycles (one formed by vertices A and B and another formed by vertices C and B), whereas the second one can be described by a single cycle (formed by vertices A, B, C, and B) in the monomer graph.

The concept of a HOR does not allow one to adequately describe the differences between the monocentromeres shown in Figure 4.8 because it requires that each monomer participates in a HOR once, necessitating the sequence ABC (that does not adequately reflect the centromere architecture) as the only possible HOR candidate. Although this example might be considered artificial, any algorithm for centromere annotation should adequately handle such cases, even if they rarely appear in the human centromeres. As we show below, cen13 and cen18 represent an evolutionary scenario that is similar to the toy centromere described in Figure 4.8.

The previous approaches to centromere annotation were based on the CE postulate and described centromeres in terms of complete and partial HORs. Given toy monocentromeres ABCABCABCABCABABABAB and ABCABABCABABCABABCAB, they described these

very different architectures in the same way: as the complete HOR ABC and the partial HOR AB, each repeating five times. Because this representation does not distinguish these two very different centromere architectures, there is a need for a more general representation of the centromere architecture that will adequately reflect all complete and partial HORs.

4.6 Data access

The codebase of HORmon is available as Supplemental Code to [118] and at GitHub (https://github.com/ablab/HORmon/tree/HORmon). Monomer and HOR decompositions of alpha satellite arrays in the CHM13 cell line are available as Supplemental Material to [118] and at Figshare (https://figshare.com/articles/dataset/HORmon/16755097/2). Jupyter notebook that reproduces figures in this paper is available at GitHub (https://github.com/TanyaDvorkina/hormon_paper/blob/dev/HORmon_paper.ipynb).

4.7 Acknowledgements

A.V.B. and P.A.P. were supported by the National Science Foundation EArly-concept Grants for Exploratory Research (EAGER) award 2032783. O.K., T.D., and I.A.A. were supported by Saint Petersburg State University, Russia (grant ID PURE 73023672). We are grateful to Karen Miga, Aleksei Shpilman, and Cynthia Wu for many insightful comments.

Author contributions: HORmon algorithm development, A.V.B., O.K., T.D., and P.A.P.; HORmon code development, O.K. and T.D.; manually curated ground-truth data, I.A.A.; manuscript draft, A.V.B. and P.A.P.; editing, all authors; conceptualization, P.A.P.

Chapter 4, in full, is a reprint of the material as it appears in Kunyavskaya, O., Dvorkina, T., Bzikadze, A. V., Alexandrov, I. A., & Pevzner, P. A. (2022). Automated annotation of human centromeres with HORmon. *Genome Research.* The dissertation author is one of the three lead developers of the HORmon algorithm and one of the three lead authors of this paper.

Chapter 5

Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads

5.1 Abstract

Although most existing genome assemblers are based on de Bruijn graphs, the construction of these graphs for large genomes and large k-mer sizes has remained elusive. This algorithmic challenge has become particularly pressing with the emergence of long, high-fidelity (HiFi) reads that have been recently used to generate a semi-manual telomere-to-telomere assembly of the human genome. To enable automated assemblies of long, HiFi reads, we present the La Jolla Assembler (LJA), a fast algorithm using the Bloom filter, sparse de Bruijn graphs and disjointig generation. LJA reduces the error rate in HiFi reads by three orders of magnitude, constructs the de Bruijn graph for large genomes and large k-mer sizes and transforms it into a multiplex de Bruijn graph with varying k-mer sizes. Compared to state-of-the-art assemblers, our algorithm not only achieves five-fold fewer misassemblies but also generates more contiguous assemblies. We demonstrate the utility of LJA via the automated assembly of a human genome that completely assembled six chromosomes.

5.2 Introduction

The emergence of long and accurate reads opened a possibility to generate the first complete (telomere-to-telomere) assembly of a human genome and to get a glimpse into biomedically important genomic regions that evaded all previous attempts to sequence them [85]. The Telomere-to-Telomere (T2T) and Human Pangenome Reference projects are now using long and accurate reads for population-scale assembly of multiple human genomes and for diagnosing rare diseases that remained below the radar of short-read technologies [132].

These breakthroughs in genome sequencing were mainly achieved using HiFi reads[89]. However, assembly of HiFi reads is far from being straightforward: the complete assembly of a human genome was generated using a semi-manual effort of a large consortium rather than by an automated approach [85]. Because such time-consuming efforts are neither sustainable nor scalable in the era of population-scale sequencing, there is a need for an accurate (nearly error-free) tool for complete genome assembly.

We argue that this challenge requires an algorithm for constructing large de Bruijn graphs [126] — that is, de Bruijn graphs for large genomes and large k-mer sizes exceeding 1,000 nucleotides. Indeed, similarly to assembling short and accurate reads, the de Bruijn graph approach has the potential to improve assemblies of any type of accurate reads. However, although it represents the algorithmic engine of nearly all short-read assemblers [33], [133], the problem of constructing large de Bruijn graphs remains open, and the existing HiFi assemblers HiCanu [75] and hifiasm [87] are based on the alternative string graph approach [134].

Because HiFi reads are even more accurate than Illumina reads, the de Bruijn graph approach is expected to work well for their assembly. Application of this approach to long HiFi reads requires either constructing the de Bruijn graph with a large k-mer size or, alternatively, using the de Bruijn graph with a small k-mer-size for follow-up repeat resolution by threading long reads through this graph. However, it remains unclear how to address three open algorithmic problems in assembling HiFi reads: (1) constructing large de Bruijn graphs, (2) error-correcting

HiFi reads so that they become nearly error-free and thus amenable to applying the de Bruijn graph approach and (3) using the entire read-length for resolving repeats that are longer than the k-mer size.

The existing genome assemblers are not designed for constructing large de Bruijn graphs because their time/memory requirements become prohibitive when the k-mer size becomes large—for example, simply storing all 5,001-mers of the human genome requires ˜4 TB. For example, the SPAdes assembler [33] faces time/memory bottlenecks assembling mammalian genomes with the k-mer size exceeding 500. To reduce the memory, some assembly algorithms avoid explicitly storing all k-mers by constructing a perfect hash map5 or the Burrows–Wheeler transform of all reads [135]. However, even with these improvements, the runtime (proportional to the k-mer size) remains large (Supplementary Note 1 in [107]).

The repeat graph approach [136] and the sparse de Bruijn graph approach [137] construct coarse versions of the de Bruijn graph with smaller time/memory requirements. Recently, [86] modified the Flye assembler for constructing the repeat graph of HiFi reads, and [138] showed how to assemble HiFi reads into a sparse de Bruijn graph. However, these graphs represent coarse versions of the de Bruijn graph, thus limiting their capabilities in assembling the highly repetitive regions such as centromeres (Supplementary Note 2 in [107]).

Here we introduce LJA, which includes three modules addressing all three challenges in assembling HiFi reads: jumboDBG (constructing large de Bruijn graphs), mowerDBG (error-correcting reads) and multiplexDBG (using the entire read-length for resolving repeats). jumboDBG combines four algorithmic ideas: the Bloom filter [101], the rolling hash [109], the sparse de Bruijn graph [137] and the disjointig generation [3]. Although each of these ideas was used in previous bioinformatics studies, jumboDBG is the first approach that combines them. LJA launches jumboDBG to construct the de Bruijn graph, launches mowerDBG that uses this graph to correct nearly all errors in reads, launches jumboDBG again to generate a much simpler graph of the error-corrected reads and launches multiplexDBG to transform it into the multiplex de Bruijn graph with varying k-mer sizes to take advantage of the entire read-lengths. LJA also

includes the LJApolish module that expands the collapsed homopolymer runs in the resulting assembly.

Although we benchmarked LJA, hifiasm and HiCanu on various genomes, evaluating the quality of the resulting assemblies is challenging because neither the complete reference for these genomes nor an automated pipeline for a reference-grade assembly validation are available yet [92]. We thus focused on benchmarking these assemblers using the HiFi read-set (referred to as the T2T dataset) from a haploid human CHM13 cell line assembled by the T2T consortium [85]. This painstakingly validated assembly represents the only accurate telomere-to-telomere sequence of a large genome available today. LJA generated the most contiguous assembly of this dataset (including complete assemblies of six human chromosomes without any misassemblies and only ten misassemblies for the entire human genome), reducing the number of assembly errors five-fold as compared to hifiasm and HiCanu. The accuracy of genome assemblers becomes critical in the era of population-wide complete genome sequencing because semi-manual validation of complete genome assemblies18 is prohibitively time-consuming.

5.3 Results

5.3.1 Key algorithmic concepts used in the LJA pipeline

The goal of genome assembly is to reconstruct a genome from its error-prone fragments (reads). Given a string-set *Reads* and an integer k, the (uncompressed) de Bruijn graph *UDB(Reads,k)* is a directed graph where each vertex is a k-mer from *Reads*, and each $(k+1)$-mer $a_1a_2\dots a_ka_{k+1}$ in reads corresponds to an edge connecting vertices $a_1a_2\dots a_k$ and $a_2\dots a_ka_{k+1}$. The uncompressed de Bruijn graph of a genome *UDB(Genome,k)* is defined by considering each chromosome in *Genome* as a single 'read'. We refer to an error-free read-set *Reads* that contains all $(k+1)$-mers from a string-set *Genome* as a *k-complete* read-set and note that *UDB(Genome,k)=UDB(Reads,k)* for a k-complete read-set.

A vertex with the indegree N and the outdegree M is referred to as an *N-in-M-out* vertex.

A vertex is *non-branching* if it is a one-in-one-out vertex and a *junction* otherwise. We refer to the set of all junctions (k-mers) in the graph *UDB(Reads,k)* as *Junctions(Reads,k)*. A path between junctions is *non-branching* if all its intermediate vertices are non-branching. A set of k-mers from *Reads* forms a *junction-superset* if it contains all junctions in *UDB(Reads,k)*.

The compressed de Bruijn graph *DB(Reads,k)* is a memory-efficient version of the uncompressed graph *UDB(Reads,k)* where each non-branching path is compressed into an appropriately labeled single edge (see Supplementary Notes 2 and 3 in [107] for the precise definition and the summary of terms used in this paper). Because LJA uses compressed de Bruijn graphs, we refer to them simply as the 'de Bruijn graphs' or *DB-graphs*. The *coverage* of an edge in *UDB(Reads,k)* is number of times the label of this edge occurs in *Reads*. The *coverage* of an edge in *DB(Reads,k)* is the average coverage of all edges in the non-branching path that was compressed into this edge.

The challenge of constructing a large de Bruijn graph

Because the compressed DB-graph *DB(Genome,k)* does not require storing all k-mers, the total length of all its edge-labels is up to k times smaller than the total length of all edge-labels in *UDB(Genome,k)*. The traditional assembly approach constructs *UDB(Reads,k)* first and transforms it into *DB(Reads,k)*. Because this approach is impractical for large genomes and large k-mer sizes, jumboDBG constructs *DB(Reads,k)* without constructing *UDB(Reads,k)*.

Even though *DB(Reads,k)* is more memory-efficient than *UDB(Reads,k)*, its direct construction also requires prohibitively large time/memory. jumboDBG thus assembles reads into disjointigs, sequences that are spelled by arbitrary walks through the (unknown) graph *DB(Reads,k)*. Even in the case of error-free reads, a disjointig might represent a misassembled concatenate of segments from various regions of the genome rather than its contiguous substring [3]. Although switching from reads to misassembled disjointigs might appear reckless, it is an important step because a carefully chosen disjointig-set *Disjointigs* has a much smaller total disjointig-length than the total read-length while resulting in the same DB-graph

DB(Disjointigs,k) as *DB(Reads,k)*.

Even though constructing *DB(Disjointigs,k)* is an easier task than constructing *DB(Reads, k)*, it still faces the time/memory bottleneck. jumboDBG addresses it by using the Bloom filter, a compact data structure for storing sets. It stores all $(k + 1)$-mers from disjointigs in a Bloom filter formed by multiple independent *hash functions*, each mapping a $(k + 1)$-mer into a bit array. The Bloom filter reports a *true positive* for all $(k + 1)$-mers occurring in disjointigs but might also report a *false positive* for some $(k + 1)$-mers that do not occur in disjointigs (with a small controlled probability). However, it never 'forgets' any inserted $(k + 1)$-mer and thus never reports a *false negative*.

5.3.2 Outline of the LJA pipeline

Below we outline all steps of the LJA pipeline using the *T2T* dataset of HiFi reads that was semi-manually assembled by the T2T consortium into a sequence *T2TGenome* by integrating information generated by multiple sequencing technologies (CHM13 reference genome version 1.1). All datasets analyzed in this paper are described in Supplementary Note 4 [107].

Figure 5.1 illustrates the work of the jumboDBG module (steps 1–7 of the LJA pipeline). Figure 5.2 illustrates the entire LJA pipeline.

Step 0: Transforming all reads into homopolymer-collapsed reads

Because errors in the length of homopolymer runs represent the dominant source of errors in HiFi reads, LJA collapses each homopolymer run X...X in each read into a single nucleotide X, resulting in a homopolymer-collapsed (HPC) read. The entire LJA pipeline works with HPC reads, except for the last LJApolish module that expands each collapsed nucleotide X in the HPC assembly into a run X...X in such a way that its run-length coincides with the correct run-length in nearly all cases (the error rate in the run-lengths is below 0.00001). On average, uncollapsed reads in the T2T dataset have ˜2,000 errors per megabase of the total read-length. Transforming them into HPC reads reduces the error rate to ˜620 errors per megabase in the HPC genome

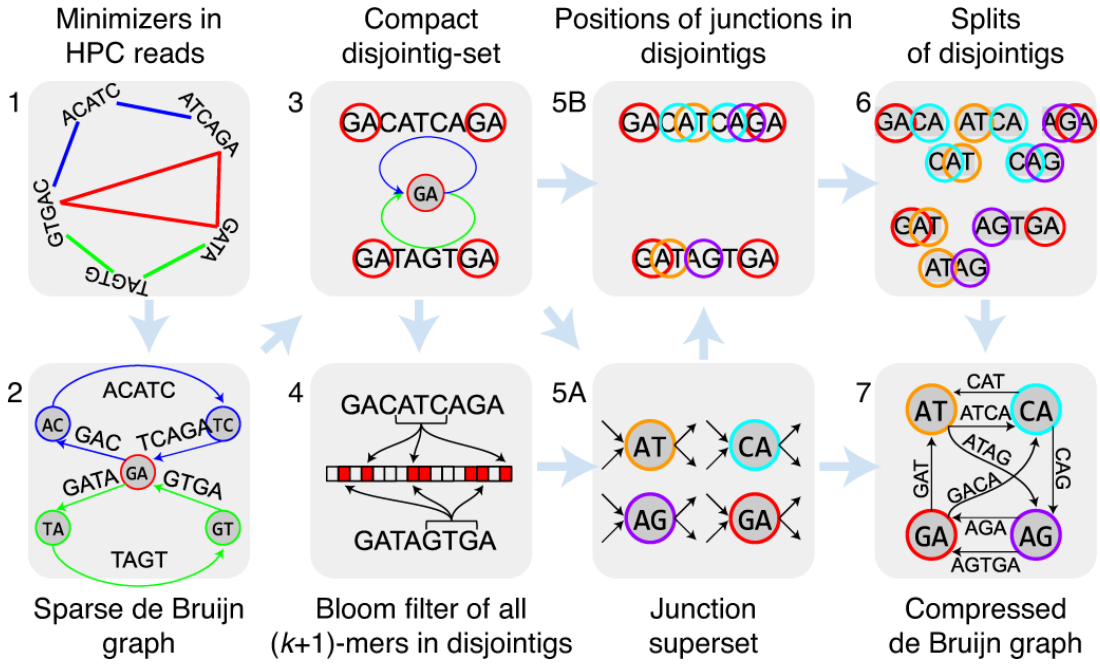


Figure 5.1. JumboDBG pipeline. (1) Generating the anchor-set *Anchors* = GA,AC,TC,TA,GT by finding all minimizers in *Reads*. For simplicity, the figure does not reflect that jumboDBG classifies all k-prefixes and k-suffixes of reads as minimizers. (2) Constructing a compact sparse de Bruijn graph *SDB(Reads,Anchors)*. (3) Constructing a compact disjointig-set *Disjointigs* as edge-labels in *SDB(Reads,Anchors)*. (4) Generating the Bloom filter for all $(k+1)$-mers in disjointigs. Each arrow directed from a $(k+1)$-mer to the Bloom filter illustrates its evaluation by one of the hash functions. (5) Using the Bloom filter to construct the junction-superset $Junctions^+$ and (5A) find positions of k-mers from $Junctions^+$ in disjointigs (5B). For simplicity, although the Bloom filter might generate some false-positive junctions (for example, TC), we only show the correct junctions in 5A. (6) Breaking disjointigs into splits. (7) Constructing the compressed de Bruijn graph *DB(Disjointigs,k)=DB(Reads,k)*.

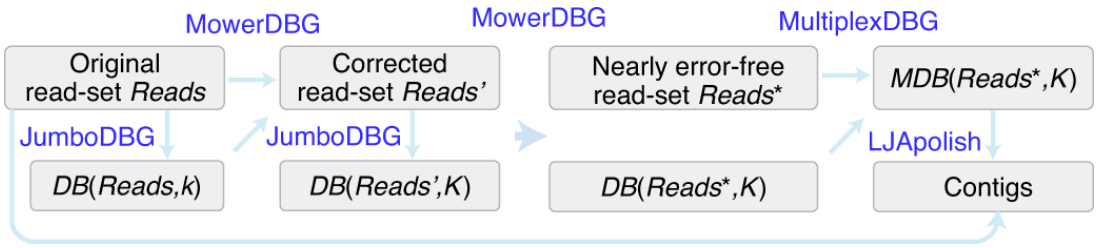


Figure 5.2. LJA pipeline. jumboDBG first constructs the de Bruijn graph *DB(Reads,k)* with a small k-mer size. mowerDBG uses this graph to correct errors in reads, resulting in an error-corrected read-set *Reads'*. Afterwards, jumboDBG constructs the de Bruijn graph *DB(Reads',K)* on the error-corrected read-set with a large K-mer size. mowerDBG uses this graph to correct even more errors in reads, resulting in an error-corrected read-set *Reads**. Because error correction in mowerDBG simultaneously modifies the graph *DB(Reads',K)* into the graph *DB(Reads***,K)*, there is no need to launch jumboDBG again for constructing *DB(Reads***,K)*. multiplexDBG complements the error-corrected read-set *Reads** by virtual reads and transforms *DB(Reads***,K)* into the multiplex de Bruijn graph *MDB(Reads***,K)*. LJApolish uses the set of original reads *Reads* to expand HPC contigs formed by non-branching paths in this graph.

(when comparing HPC reads to the HPC reference genome) and makes 38% of all HPC reads error-free.

Step 1: Generating the anchor-set by finding all minimizers in the HPC read-set Reads

Given a hash function defined on k-mers, a *minimizer* of a word is defined as a k-mer with a minimal hash in this word. A minimizer-set of a string is defined as the set of all minimizers over all its substrings of length *width* [139]. We modify the original concept of a minimizer of a linear string by adding its prefix and suffix k-mers to the set of its minimizers. A sensible choice of the parameter width (and a hash function) ensures that each read is densely covered by minimizers and that overlapping reads share minimizers, facilitating the assembly. jumboDBG generates the set of all minimizers in reads that we refer to as the *anchor-set Anchors*.

Step 2: Constructing a compact sparse de Bruijn graph.

Because the direct construction of *DB(Reads,k)* faces the time/memory bottleneck, jumboDBG first assembles reads into disjointigs. Although the Flye assembler [3] constructs disjointigs by searching for overlapping reads, it is unclear how to extend this construction to highly repetitive regions (for example, centromeres) that Flye does not adequately reconstruct.

Instead, jumboDBG constructs a sparse de Bruijn graph and generates disjointigs in this graph that are also disjointigs in the much larger (but unknown) de Bruijn graph.

Given a set of k-mers *Anchors* from a string-set *Reads*, we consider each pair a and a' of consecutive anchors in each read and generate a substring of the read (called a *split*) that starts at the first nucleotide of a and ends at the last nucleotide of a'. The resulting set of splits (after collapsing identical splits into a single one) is denoted *Splits(Reads,Anchors)*. The sparse de Bruijn graph *SDB(Reads,Anchors)* is defined as a graph with the vertex-set *Anchors* and the edge-set *Splits(Genome, Anchors)*. Each string in *Splits(Genome,Anchors)* represents the label of an edge connecting its k-prefix and k-suffix in *SDB(Reads,Anchors)*.

A sparse de Bruijn graph *SDB(Reads,Anchors)* is *compact* if all its vertices (anchors) represent junctions in the graph *DB(Reads,k)*. jumboDBG transforms the initially constructed graph *SDB(Reads,Anchors)* into a compact sparse de Bruijn graph *SDB(Reads,Anchors*)* to facilitate the construction of a compact disjointing-set (see below).

Step 3: Constructing a compact disjointig-set.

A disjointig-set is *complete* if its disjointigs contains all $(k + 1)$-mers from *Reads*. A disjointig in the sparse de Bruijn graph *SDB(Reads,Anchors)* is compact if its k-suffix and k-prefix are both junctions in *DB(Reads,k)*. A complete disjointig-set is *compact* if each disjointig in this set is compact. Because the set of all $(k + 1)$-mers in a complete disjointig-set coincides with the set of all $(k + 1)$-mers in reads, *DB(Reads,k)=DB(Disjointigs,k)* for a complete disjointig-set. Thus, the problem of constructing the compressed de Bruijn graph from reads is reduced to constructing the compressed de Bruijn graph of a complete disjointig-set. However, not every complete disjointig-set enables efficient construction of this graph. Below we show that a compact disjointig-set enables efficient construction of *DB(Disjointigs,k)*. jumboDBG constructs a compact disjointig-set as the set of edge-labels in the compact sparse de Bruijn graph *SDB(Reads,Anchors*)*.

Step 4: Generating the Bloom filter of all (k + 1)-mers in disjointigs

Even though we have reduced constructing the DB-graph *DB(Reads,k)* to constructing *DB(Disjointigs,k)*, even this simpler problem faces the time/memory bottleneck. To address it, jumboDBG constructs the Bloom filter [140], [141] for storing all (k + 1)-mers in disjointigs and uses the rolling hash to query them in $O(1)$ instead of *O(k)* time.

Step 5: Using the Bloom filter to construct the junction-superset

The Bloom filter enables rapid construction of small junction-superset $Junctions^+$ even though the DB-graph of disjointigs has not been constructed yet. To achieve this goal, jumboDBG uses the Bloom filter to compute the upper bound on the indegree and outdegree of each vertex (k-mer) in the unknown DB-graph of disjointigs by checking which of its 4 + 4 = 8 forward and backward extensions by a single nucleotide represent (k + 1)-mers present in the Bloom filter. A k-mer is called a *joint* if the upper bounds on either its indegree or outdegree exceed 1. Because each junction is a joint, the set of all joints forms a junction-superset.

Step 6: Using the junction-superset to break disjointigs into splits

jumboDBG also uses the Bloom filter to rapidly identify the positions of all k-mers from $Junctions^+$ in disjointigs and to break disjointigs into splits afterward.

Step 7: Using splits to construct *DB(Disjointigs,k)=DB(Reads,k)*

An *edge-subpartition* of an edge (v,w), in a graph 'substitutes' it with two edges by 'adding' a vertex u in the 'middle' of this edge. A *subpartition* of a graph is defined as a result of a series of edge-subpartitions. As described in the Methods, the string-set $Splits(Disjointigs,Junctions^+)$ represents edge-labels of a subpartition of the graph *DB(Disjointigs,k)*. jumboDBG compresses all one-in-one-out vertices in this graph to generate *DB(Disjointigs,k) = DB(Reads,k)*.

Step 8: Correcting errors in reads with mowerDBG

Supplementary Note 5 in [33] illustrates that jumboDBG generates highly contiguous assemblies of a k-complete read-set sampled from *T2TGenome* for large values of k—for example, k = 5,001. However, assembling real reads is challenging because the DB-graph *DB(T2T,k)* of the T2T read-set is much more complex than the DB-graph *DB(T2TGenome,k)*. mowerDBG uses the DB-graph *DB(Reads,k)* to correct errors in the read-set Reads. LJA performs two rounds of error correction and launches mowerDBG twice, with a small k-mer size in the first round, resulting in an error-corrected read-set *Reads'*, and a large K-mer size in the second round, resulting in a nearly error-free read-set $Reads^*$ (default values $k = 501$ and $K = 5,001$).

jumboDBG constructs the graph *DB(T2T,k)* with 33,230,906 edges in only 2.7 h using 54 Gb of memory. However, over 99% of edges in this graph are triggered by errors in reads: if reads in the T2T dataset were error-free, jumboDBG would construct the DB-graph of the error-free read-set *T2TErrorFree* with only 214,517 edges in 0.6 h using 33 Gb of memory. mowerDBG corrects most errors in reads, resulting in a much smaller DB-graph *DB(T2T',k)* with 297,176 edges on the error-corrected read-set T2T'. jumboDBG further constructs the DB-graph *DB(T2T',K)* using a larger K-mer size with 79,908 edges. Afterwards, mowerDBG performs the second round of error-correction in this graph, resulting in a nearly error-free read-set $Reads^*$ and a DB-graph $DB(T2T^*,K)$ with only 6,516 edges that approximates the graph *DB(T2TErrorFree,K)* with 4,956 edges. We note that, because the read-set *T2TErrorFree* was constructed based on the reference *T2TGenome*, which excluded the heterozygous regions present in the CHM13 cell line [85], $DB(T2T^*,K)$ might have some heterozygous edges missing in *DB(T2TErrorFree,K)*.

Step 9: Transforming the DB-graph $DB(Reads^*,K)$ into the multiplex de Bruijn graph (Fig. 5.4)

The choice of the k-mer size greatly affects the complexity of the DB-graph: gradually increasing k leads to a less tangled but more fragmented DB-graph. This tradeoff affects the contiguity of assembly, particularly in the case when the k-mers coverage by reads is non-uniform, let alone when the read-set misses some genomic k-mers. Ideally, we would like to vary the

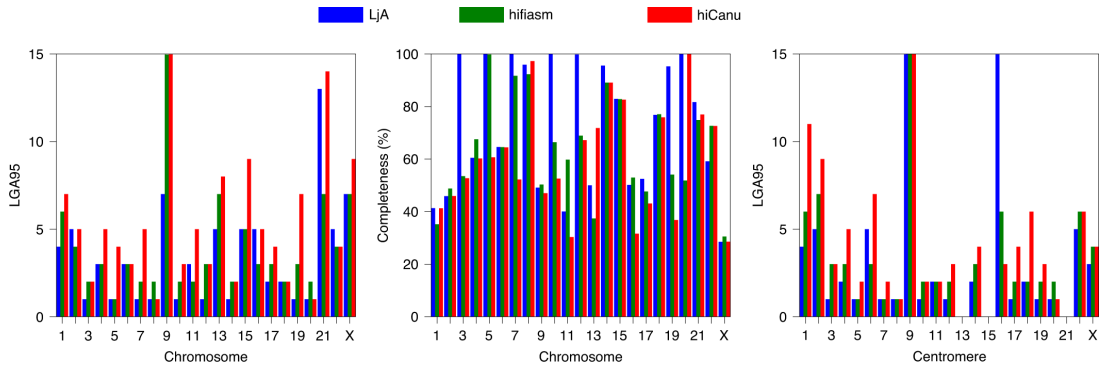


Figure 5.3. Chromosome-by chromosome LGA95 (left) and completeness (center) metrics as well as centromere-by-centromere LGA95 metric (right) for LJA (blue), hifiasm (green) and HiCanu (red) assemblies of the T2T read-set. LGA95 for each chromosome (centromere) is defined as the minimum number of aligned blocks needed to cover 95% of its length (aligned blocks are obtained by breaking contigs at misassembly breakpoints). LJA 100%-assembled six chromosomes and resulted in a better assembly (for example, assembly with lower LGA95) than hifiasm and HiCanu for most chromosomes. Although hifiasm resulted in lower LGA95 than LJA on chromosomes 2 (4 versus 5), 11 (2 versus 3), 16 (3 versus 5), 21 (7 versus 13) and 22 (4 versus 5), it made ten misassemblies in these five chromosomes compared to only one misassembly made by LJA. For example, for chromosome 21 with the biggest gap in the LGA95 values between hifiasm and LJA (7 versus 13), LJA resulted in a higher completeness (82%) than hifiasm (75%). As another example, LJA resulted in a much larger LGA95 = 22 on centromere 16 than hifiasm (6) and HiCanu (3) but made no errors (both hifiasm and HiCanu had three misassemblies). LGA95 values for centromeres 13, 15 and 21 (for all assemblers) are undefined because QUAST-LG reports multiple gaps in these centromeres. Long and highly repetitive rDDNA arrays are the only regions in the human genome that were not assembled by the T2T consortium. These regions are represented as simulated rDDNA models rather than correct rDDNA sequences in T2TGenome. Because centromeres 13, 15 and 21 include rDDNA models, QUAST-LG reports multiple gaps in their coverage by contigs that sum up to more than 5% of the lengths of these centromeres.

k-mer size, reducing it in low-coverage regions (to avoid fragmentation) and increasing it in high-coverage regions (to improve repeat resolution). The iterative de Bruijn graph approach [33], [32] is a step toward addressing this goal by incorporating information about the de Bruijn graphs for a range of k-mer sizes $k_1 < k_2 < \ldots < k_t$ into the de Bruijn graph. However, this approach still constructs a graph with a fixed k_t-mer size.

multiplexDBG transforms the DB-graph *DB(Reads*,K)* into the multiplex de Bruijn graph *MDB(Reads*,K)* with vertices labeled by strings of length varying from K to K^+, where K^+ is larger than K (default value K^+ = 40,001). It transforms *DB(T2T*,5001)* with 6,516 edges

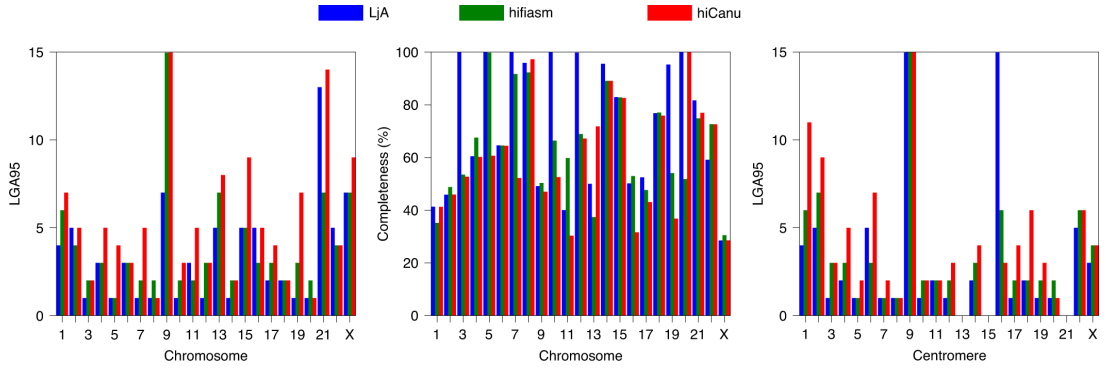


Figure 5.4. Constructing a multiplex de Bruijn graph. **a**, Iterative construction of the compressed de Bruijn graph. Transformation of *DB(Reads,2)* into *DB(Reads,3)* for the read-set Reads = **AACCGG,TTCCGG**. The first read defines a transition between edges **(AA,CC)** and **(CC,GG)**, whereas the second read defines a transition between edges **(TT,CC)** and **(CC,GG)**. The resulting transition-set defines a transition-graph that coincides with *DB(Reads,3)*. The read-paths that define the transition-set are depicted as red curves traversing *DB(Reads,2)*. multiplexDBG 'tears apart' edges of the *DB(Reads,2)* and increases the k-mer length within the vertices of the resulting isolated edges. Afterwards, it introduces new red edges that correspond to connections induced by transitions, resulting in a path-graph. Compressing non-branching paths in the path-graph (ignoring the edge colors) results in *DB(Reads,3)*. Because vertex '**CC**' is simple, graphs *DB(Reads,2)* and *DB(Reads,3)* have the same topology. The shown transformation merely substitutes the k-mer label of this vertex by the $(k+1)$-mer *label(w)*symbol*$_{k+1}$*(out)*. It preserves the label of the outgoing edge from this vertex and adds a single symbol to the labels of incoming edges into this vertex. **b**, Transforming a complex vertex. The read-sets *Reads*$_1$ = **AAGGAG,TTGGCT** (above) and *Reads*$_2$ = **AAGGAG, TTGGCT, AAGGCT** (below) result in the same graphs *DB(Reads*$_1$*,2)* and *DB(Reads*$_2$*,2)* but different graphs *DB(Reads*$_1$*,3)* and *DB(Reads*$_2$*,3)*. Because vertex **GG** in the graph *DB(Reads*$_1$*,2)=DB(Reads*$_2$*,2)* is complex, the topology of the graphs *DB(Reads*$_1$*,3)* and *DB(Reads*$_2$*,3)* depends on the read-set. *DB(Reads*$_1$*,3)* consists of two connected components (top), whereas *DB(Reads*$_2$*,3)* is a single-component graph because it has an extra edge (labeled **AGGC**) introduced by the additional read **AAGGCT**. **c**, Limitation of the de Bruijn graph approach to genome assembly. Reckless resolution of unpaired complex vertices might disconnect the genome traversal. For a read-set *Reads* = **AAGGAG, AAGGCT, TTGG**, because the read-path of **TTGG** ends inside the complex vertex **GG**, the vertex **TGG** (with red outline) represents a dead-end. **d**, Multiplex de Bruijn graph transformation. The multiplex de Bruijn graph for the same read-set avoids reckless resolution of complex vertices by freezing unpaired complex vertices.

into *MDB(T2T*,5001)* with only 1,432 edges and generates HPC contigs. Note that labels of some vertices of this graph are longer than all reads in the T2T dataset because multiplexDBG adds virtual reads to the read-set (Supplementary Note 6 in [107]).

Step 10: Expanding HPC contigs

LJApolish expands HPC contigs (edge-labels in the multiplex de Bruijn graph) and results in an accurate final assembly with only ˜15 single-base errors per million nucleotides.

5.3.3 Evaluating genome assemblies

Given a string s in a string-set S, we define $L^+(s)$ as the total length of all strings in S with length at least $|s|$. The N50 metric for a contig-set S is defined as the length of the longest contig s in S with $L^+(s) \leq 0.5 \cdot |S|$. See [64] on similar NG50, NGA50, NGA75 or LGA95 metrics.

We used the standard benchmarking metrics [64] as well as the additional *completeness metric* aimed at high-quality assemblies. We denote the length of a string s as $|s|$ and the total length of all strings in a string-set S as $|S|$. A contig is *correctly assembled* if it has no misassemblies. *Completeness* of a chromosome assembly is defined as the length of the longest correctly assembled contig from this chromosome divided by the chromosome length (in percentages). A chromosome is N%-assembled if its completeness is at least N%.

5.3.4 Benchmarking LJA on the T2T read-set

We benchmarked assemblies of the T2T read-set dataset against *T2TGenome* — the only complete and carefully validated large genome reference available today[92]. Table 5.1 illustrates that LJA produced the most contiguous assembly (NG50 = 97 Mb) with six 100%-assembled and nine 95%-assembled chromosomes (compared to NG50 = 90 Mb and one 95%-assembled chromosome for hifiasm). We classify a contig with length larger than or equal to NG50 (NG75) as NG50-long (NG75-long) contig. It turned out that 1 (4) out of all 12 NG50-long contigs assembled by LJA (hifiasm) are misassembled, resulting in a reduced NGA50 = 81 Mb for

hifiasm. LJA (hifiasm) misassembled 1 out of its 24 (9 out of its 23) NG75-long contigs. Figure 5.3 demonstrates that LJA substantially improves on both hifiasm and HiCanu with respect to chromosome-by-chromosome and centromere-by-centromere assemblies.

LJA substantially reduced the number of misassemblies (10) as compared to hifiasm (58) and HiCanu (47). Although LJA produced more contigs than hifiasm (665 versus 448), this increased number does not indicate an inferior assembly but, rather, reflects the fact that LJA assembled an extra 13 Mb of the genome as compared to hifiasm (99.7% versus 99.3% genome fraction). In fact, hifiasm produced more contigs longer than 50 Kb than LJA (250 versus 194).

LJA and hifiasm made an order of magnitude fewer mismatches and indels than HiCanu. Analysis of these errors should take into account that an assembler covering a larger fraction of a genome might have additional errors in highly repetitive regions that are not covered by other assemblers. Although LJA made slightly more errors than hifiasm (34,293 versus 33,732), it made a smaller number of errors in regions assembled by both LJA and hifiasm (2,330 LJA errors were made in highly repetitive regions that were not assembled by hifiasm).

Table 5.1 also benchmarks the 'ideal' assembler on a k-complete read-set that outputs contigs as the edge-labels of *DB(T2TGenome,20,000)*. It turned out that LJA assembly is similar in quality to the theoretically optimal assembly of error-free reads of length 20,000.

Supplementary Note 5 in [107] provides information about benchmarking individual LJA modules and illustrates that the runtime/memory of jumboDBG is largely defined by the size of the constructed DB-graph rather than the k-mer size.

5.3.5 Assembling a diploid human genome

Genome assemblers often collapse two heterozygous alleles (typically represented as bulges in the DB-graph) into a single copy to increase the contiguity of the consensus assembly, a mosaic of segments from maternal and paternal chromosomes. Diploid assemblers try to prevent such collapsing by (1) constructing a phased assembly graph that accurately represents the heterozygous alleles and (2) using the entire read-lengths and complementary technologies to

Table 5.1. The assemblies were generated with hifiasm version 0.15.5-r352 and HiCanu version version 2.3 and benchmarked using QUAST-LG version 5.0.2 with *T2TGenome* as the reference. Because authors of the QUAST-LG[64] recommend using HPC contigs for identifying misassemblies (Supplementary Note 5 in [107]), misassemblies were identified by a separate run of QUAST-LG with HPC contigs against the HPC reference. Analysis of misassemblies reported by QUAST-LG in the 30 longest contigs (for each assembly tool) using the Icarus tool [142] confirmed that they all represent structural errors (a large insertion, deletion or relocation) rather than alignment artifacts. LJA 100%-assembled chromosomes 3, 5, 7, 10, 12 and 20; HiCanu 100%-assembled chromosome 20; and hifiasm 100%-assembled chromosome 5. Because all assemblies might have small variations at the chromosome ends, 99.9%-assembled chromosomes are counted as 100%-assembled chromosomes. 'Ideal' refers to the theoretically optimal assembly of a *k*-complete read-set obtained by generating contigs as edge-labels of the graph *DB(T2TGenome,20,000)*. NGA50 and NGA75 metrics are reported based on contigs broken at positions defined by misassemblies in HPC contigs. Note that, even though very few HPC reads in the T2T dataset are longer than 20,000 bp, the LJA assembly of this read-set might improve on the ideal assembly *DB(T2TGenome,20,000)* because it uses information about coverage for loop resolution (Supplementary Note 8 in [107]). The reference length is 3,054,832,041. The number of LJA contigs (665) is smaller than the number of edges in the constructed multiplex de Bruijn graph (1,432) because removing overlaps between contigs in the final LJA output results in many short contigs (LJA removes contigs that become shorter than 5 kb).

	IdealAssembler(20,000)	**LJA**	**hifiasm**	**hiCanu**
total length (Mbp)	3,055	**3,050**	3,043	3,331
#contigs / #contigs longer than 50 kb	**401/401**	**665/194**	**448/250**	1,447/427
#misassemblies*	0	**10**	58	47
#local misassemblies*	0	**46**	54	120
#100%-assembled / #95%-assembled chromosomes	5/5	**6/9**	1/1	1/2
#100%-assembled / #95%-assembled centromeres	5/5	**9/9**	3/3	2/2
genome fraction (%)	100	**99.74**	99.28	99.60
duplication ratio	1	**1.001**	1.003	1.094
# mismatches / #indels per 1 Mbp	0/0	**6.9/7.8**	**7.7/5.7**	23.3/94.9
longest alignment (Mbp)	160.6	**201.1**	181.6	142.3
total aligned length (Mbp)	3055	**3047**	3038	3327
NG50 (Mbp)	93.6	96.7	90.2	69.7
NGA50 (Mbp)	93.6	96.7	75.1	69.7
NGA75 (Mbp)	59.2	44.0	36.4	30.5

increase the contiguity of paternal/maternal contigs in the phased assembly graph and generate a haplotype-resolved assembly [87], [143]. Constructing an accurate and contiguous phased assembly graph is a critical step in both consensus and diploid assembly and the focus of the current efforts of the T2T project as it scales up from a single haploid to multiple diploid genomes. Indeed, a fragmented phased graph makes it difficult to generate consensus and haplotype-resolved assemblies, not to mention that errors in the graph likely trigger errors in these assemblies.

LJA generates a phased DB-graph of a diploid genome that represents an excellent starting point for generating these assemblies. We analyzed the phased LJA and hifiasm assemblies using the HG002 read-set from a diploid human genome. LJA generated an assembly with N50 = 383 kb and total length 5.8 Gb, whereas hifiasm generated an assembly with N50 = 310 kb and total length 6.7 Gb (before heterozygosity collapsing) that is ˜2.2 times larger than the human genome length. Although LJA generated a more contiguous phased assembly than hifiasm, it is unclear how to evaluate the accuracy of these assemblies in the absence of a validated complete reference genome for each haplome.

Although the LJA graph of the diploid read-set is much larger than the graph of the haploid T2T read-set (42,298 versus 1,440 edges), it has a rather simple 'bulged' structure, making it well-suited for the follow-up consensus and haplotype-resolution steps. Each bulge represents differing segments of paternal/maternal alleles (average length ˜150 kb) alternating with identical segments of paternal/maternal alleles (average length ˜30 kb).

5.3.6 Assembling mouse, maize and fruit fly genomes

We benchmarked LJA, hifiasm and HiCanu on the MOUSE, MAIZE and FLY HiFi read-sets from the inbred mouse, maize and fly species. It is unclear how to compare the reference genomes with HiFi assemblies for these datasets because the quality of these assemblies might exceed the quality of the references. For example, although Table 5.2 illustrates that LJA and hifiasm improved on HiCanu with respect to NG50 metric, this metric does not account for

Table 5.2. Benchmarking LJA, hifiasm and HiCanu on MOUSE, MAIZE and FLY read-sets. Mouse, maize and fruit fly samples represent the C57BL/6J strain of *Mus musculus*[144], the B73 strain of *Zea mays* [144] and Drosophila ananassae [145], respectively. We used estimates of the lengths of the mouse, maize and fly genomes (2.7 Gb, 2.3 Gb and 0.22 Gb, respectively) for NG50 calculation. All assemblers were run with default parameters recommended for diploid genome assembly.

Assembler	MOUSE			MAIZE			FLY		
	#contigs	total length (Gb)	NG50 (Mb)	#contigs	total length (Gb)	NG50 (Mb)	#contigs	total length (Gb)	NG50 (Mb)
LJA	1,282	2.71	24	1,310	2.15	26	313	0.22	9.5
hifiasm	658	2.61	21	1,136	2.18	35	933	0.24	10.0
HiCanu	3,334	2.67	15	1,992	2.17	23	6,439	0.32	9.2

errors in assemblies and references. For example, hifiasm assembled the MAIZE dataset with the highest NG50 but made more misassemblies (6,081 versus 5,594 for LJA), resulting in a rather low NGA50 (1.4 Mb for both hifiasm and LJA). The large number of misassemblies for all inbred datasets suggests that many of them might be triggered by errors in the reference genome or differences between maternal/paternal alleles (even after inbreeding). Supplementary Note 7 in [107] illustrates that existing HiFi assemblers generate rather different results, implying that some of them make many assembly errors. In the absence of an automated validation pipeline, it remains unclear how to detect these errors in the era of complete genome sequencing.

5.4 Discussion

The development of assembly algorithms started from applications of the overlap/string graph approach. Even though this approach becomes slow and error-prone with respect to detecting overlaps in the highly repetitive regions, the alternative DB-graph approach [146], [147] was often viewed as a theoretical concept rather than a practical method.

Even after it turned into the most popular method for short-read assembly, the development of algorithms for assembling long error-prone reads again started from the overlap/string graph approach [76], [1], [148] because the DB-graph approach was viewed as inapplicable to error-prone reads [149]. Indeed, because long k-mers from the genome typically do not even occur in error-prone reads, it seemed unlikely that the DB-graph approach might assemble such

reads. However, the development of Flye[3] and wtdbg2 [7] demonstrated, once again, that the DB-graph-based long-read assemblers result in accurate and order(s) of magnitude faster algorithms than the overlap/string graph approach.

Because the DB-graph approach was initially designed for assembling accurate reads, it would seem natural to use it for assembling long and accurate reads. However, the history repeated itself and the first HiFi assemblers again relied on the overlap/string graph approach [75], [87]. We described an alternative approach for assembling HiFi reads, illustrating that the 'contest' between the overlap/string graph and the de Bruijn graph approach continues. Benchmarking on the T2T dataset demonstrated that LJA improves on the state-of-the-art HiFi assemblers with respect to both contiguity and accuracy. Although it is unclear how to conduct benchmarking without validated complete reference genomes, LJA results on the HG002 dataset illustrate that it generates highly contiguous phased assemblies. Although this paper focuses on phased assemblies, it has immediate implications for the downstream applications because phased assemblies represent a stepping stone for both consensus and haplotype-resolved assemblies. For example, bulge-collapsing and tip removal in the phased LJA assembly of the HG002 read-set results in a contiguous consensus assembly with N50 = 54 Mb. We are now developing the diploidLJA tool for haplotype-resolved assembly and the nanoLJA tool for combining HiFi and Oxford Nanopore reads to improve the contiguity of assemblies.

5.5 Methods

This section is organized as follows. Before constructing the compressed de Bruijn graph *DB(Reads,k)*, we describe a simpler yet still open problem of constructing the compressed de Bruijn graph *DB(Genome,k)* of a large circular string Genome for a large k-mer size. This problem [141] serves as a stepping stone for constructing a much larger graph *DB(Reads,k)*. After describing the algorithm for solving this problem, we describe complications that arise in the case of constructing the compressed de Bruijn graph from a genome with linear chromosomes.

Afterwards, we describe how to modify the algorithm for constructing *DB(Genome,k)* into the algorithm for constructing *DB(Reads,k)*. Because this transformation faces the time/memory bottleneck, we describe how jumboDBG first assembles reads into disjointigs. Afterwards, we describe steps 8 (error-correcting reads), 9 (constructing multiplex de Bruijn graph) and 10 (expanding HPC assembly) of the LJA pipeline. Supplementary Note 9 in [107] describes LJA parameters.

Because the algorithm for constructing the de Bruijn graph of a circular genome does not require construction of disjointigs (step 3), we number its steps 4–7 as 4G–7G to be consistent with the previously described steps of jumboDB:

- **Step 4G:** Generating the Bloom filter for all $(k+1)$-mers in *Genome*;
- **Step 5G:** Using the Bloom filter to construct a junction-superset $Junctions^+$ of *Genome*;
- **Step 6G:** Using the set $Junctions^+$ to break *Genome* into splits;
- **Step 7G:** Using splits to construct *DB(Genome,k)*.

5.5.1 Sparse de Bruijn graphs

Given a set of k-mers *Anchors* from a string-set *Genome*, the sparse de Bruijn graph *SDB(Genome,Anchors)* is a graph with the vertex-set *Anchors* and the edge-set *Splits(Genome, Anchors)* (each split in *Splits(Genome,Anchors)* represents a label of an edge connecting its k-prefix with its k-suffix). Two vertices in this graph might be connected by multiple edges with different labels corresponding to different splits with the same k-prefixes and k-suffixes. A straightforward algorithm for constructing *SDB(Genome,Anchors)* takes $O(|Genome| \cdot |Anchors| \cdot k)$ time.

A string-set *Genome* corresponds to a path-set that traverses each edge in the de Bruijn *DB(Genome,k)* at least once and spells *Genome*. We refer to this path-set as the *genome traversal*.

When the set of anchors is equal to the set of junctions *Junctions* = *Junctions(Genome,k)*, each vertex of *DB(Genome,k)* is an anchor, and each edge corresponds to two consecutive

anchors in the genome traversal. Therefore, the sparse de Bruijn graph *SDB(Genome,Junctions)* coincides with *DB(Genome,k)*. Moreover, if *Junctions*$^+$ is a *superset* of all junctions, that contains all junctions as well as some *false junctions* (that is, non-branching k-mers from *Genome*), *SDB(Genome,Junctions*$^+$*)* is a subpartition of *DB(Genome,k)*.

If the junction-set *Junctions* = *Junctions(Genome,k)* was known, construction of *DB(Genome, k)* would be a simple task because it coincides with *SDB(Genome,Junctions)*. Moreover, even if the junction-set is unknown but a junction-superset *Junctions*$^+$ (with a small number of false junctions) is known, one can construct *DB(Genome,k)* by constructing *SDB(Genome, Junctions*$^+$*)* and compressing all its non-branching paths.

Step 4G: Generating the Bloom filter for all (k + 1)-mers in *Genome*

In the case of assembling reads, jumboDBG stores all (k + 1)-mers from disjointigs in a Bloom filter *Bloom(Disjointigs,k,BloomNumber,BloomSize)* formed by *BloomNumber* different independent *hash functions*, each mapping a (k + 1)-mer into a bit array of size *BloomSize*. In the case of a circular genome, it constructs the Bloom filter in the same way by assuming that this genome forms a single disjointig. Storing all (k + 1)-mers in a Bloom filter allows one to quickly query whether an arbitrary (k + 1)-mer occurs in disjointigs and thus speed up the de Bruijn graph construction[140]. Given hash functions $h_1, h_2, \ldots, h_{BloomNumber}$ and a k-mer a, one can quickly check whether all bits $h_1(a), h_2(a), \ldots, h_{BloomNumber}(a)$ of the Bloom filter are equal to 1, an indication that the k-mer a might have been stored in the Bloom filter. Supplementary Note 9 in [107] describes how jumboDBG sets the parameters of the Bloom filter.

Step 5G: Using the Bloom filter to construct the junction-superset *Junctions*$^+$ of *Genome*

To generate a junction-superset *Junctions*$^+$ with a small number of false junctions, jumboDBG uses the Bloom filter to compute the upper bound on the indegree and outdegree of each vertex in *UDB(Genome,k)* as described in the Results.

A vertex in a graph is *complex* if both its indegree and outdegree exceed 1 and *simple* otherwise. A junction is a *dead-end* if it has no incoming or no outgoing edges and a *crossroad*

otherwise. In the case of a genome with linear chromosomes, the Bloom filter might overestimate the indegree and/or outdegree of some dead-end junctions—for example, to misclassify a zero-in-one-out junction as a simple vertex. However, all crossroad junctions will be correctly identified, thus generating a junction-superset (in the case of a circular genome that does not have dead-end junctions), constructing a subpartition of *DB(Genome,k)* and further transforming it into *DB(Genome,k)*.

Steps 6G and 7G: Using the set *Junctions*$^+$ to break *Genome* into splits and using splits to construct *DB(Genome,k)*

To construct *SDB(Genome,Junctions*$^+$*)*, one can generate a Bloom filter for computing a junction-superset *Junctions*$^+$ and check which k-mers from *Genome* coincides with a k-mer from *Junctions*$^+$. Both these tasks require computing hashes of each k-mer, a procedure that usually takes $O(k)$ time and becomes slow when k is large. For example, constructing a hashmap of *Junctions*$^+$ results in a prohibitively slow algorithm for constructing *SDB(Genome,Junctions*$^+$*)* with $O(|Genome| \cdot k)$ runtime.

To compute the hash function in $O(1)$ rather than *O(k)* time, jumboDBG uses a 128-bit polynomial rolling hash of k-mers from the genome to rapidly check whether two k-mers (one from *Genome* and one from *Junctions*$^+$) are equal and to reduce the time to construct the compressed de Bruijn graph to $O(|Genome|)$. Similarly, to speed up the construction of *Junctions*$^+$, instead of storing k-mers, jumboDBG stores their rolling hashes in the Bloom filter, thus reducing the runtime from $O(|Genome| \cdot k)$ to $O(|Genome|)$. Therefore, it constructs the compressed de Bruijn graph of a circular genome in $O(|Genome|)$ time that does not depend on the k-mer size.

5.5.2 Constructing the compressed de Bruijn graph from reads

The described algorithm for constructing the de Bruijn graph of a circular genome can be applied to any string-set *Genome*, resulting in a graph that we refer to as *DB***(Genome,k)*. However, although *DB***(Genome,k)=DB(Genome,k)* for a genome formed by circular chromo-

somes, it is not the case for a genome with linear chromosomes (or a genome represented by a k-complete error-free read-set).

We say that a string-set *Genome* bridges an edge of the compressed de Bruijn graph *DB(Genome,k)* if the label of this edge represents a substring of *Genome*. A genome is called *bridging* (with respect to a given k-mer size) if it bridges all edges of *DB(Genome,k)* and *non-bridging* otherwise. For example, a genome formed by 'chromosomes' *Genome* = {ATGC,GCACC} is non-bridging because *DB(Genome,2)* consists of a single edge with label ATGCACC that does not represent a substring of *Genome*.

Although DB^{*}*(Genome,k)=DB(Genome,k)* in the case of a bridging genome, *DB*(Genome, k)* does not necessarily coincide with *DB(Genome,k)* for a non-bridging genome—for example, a genome with linear chromosomes that does not bridge all edges of *DB(Genome,k)*. However, after extending the junction-superset by k-prefixes and k-suffixes of all linear chromosomes, the same algorithm will construct the graph that represents a subpartition of *DB(Genome,k)*. Although this subpartition can be further transformed into *DB(Genome,k)* by compressing all non-branching paths, the resulting algorithm becomes slow when the number of linear chromosomes is large, resulting in a prohibitively large junction-superset. This increase becomes problematic when one constructs the compressed de Bruijn graph *DB(Reads,k)* because each read represents a linear 'mini-chromosome'. Even more problematic is the accompanying increase in the number of calls to the hash functions that scales proportionally to the coverage of the genome by reads. An additional difficulty is that, in the absence of the genome, it is unclear how to select the appropriate size of the Bloom filter that keeps the false-positive rate low: selecting it to be proportional to the total read-length (as described in Supplementary Note 9 in [107]) results in the prohibitively large memory. Even if the genome size is known, it is unclear how to select *BloomSize* because the number of different k-mers in reads affects the false-positive rate.

jumboDBG addresses these problems by assembling reads into compact disjointigs that form a bridging genome for the graph *DB(Reads,k)* and constructing the compressed de Bruijn graph *DB(Disjointigs,k)=DB(Reads,k)* from the resulting disjointig-set *Disjointigs* instead of the

read-set *Reads*. LJA sets the parameter *BloomSize* to be proportional to the total disjointig-length (that is typically an order of magnitude smaller than the total read-length), thus greatly reducing the memory footprint.

Step 3: Constructing a compact disjointig-set from a read-set

We defined the concepts of complete and compact disjointig-sets in the Results. Similarly to a disjointig of a read-set, a disjointig of a genome is defined as a string spelled by an arbitrary path in *DB(Genome,k)*. If a disjointig-set *Disjointigs* is complete, then *DB(Genome, k) = DB(Disjointigs, k)*. However, the graph $DB^*(Disjointigs,k)$ constructed by jumboDBG might differ from *DB(Genome,k)* because it does not include edges of *DB(Genome,k)* that are not bridged by *Disjointigs*. However, if a disjointig-set *Disjointigs* is compact, it forms a bridging genome, implying that $DB^*(Disjointigs,k)=DB(Genome,k)$.

jumboDBG constructs a compact disjointig-set as a set of edge-labels in the compact sparse de Bruijn graph $SDB(Reads,Anchors^*)$. Traditionally, the anchor-set *Anchors* for constructing *SDB(Reads,Anchors)* is constructed as a set of all minimizers across all reads. However, if the *k*-prefix and/or the *k*-suffix of a read are not anchors, they might be missing in *SDB(Reads,Anchors)* because only segments between anchors are added to this graph. As described in the Results, jumboDBG modifies the concept of a minimizer of a linear string by adding its *k*-prefix and *k*-suffix to the set of its minimizers, resulting in the set *Anchors = Anchors(Reads,width,k)*. jumboDBG constructs the sparse de Bruijn graph *SDB(Reads,Anchors)*, transforms it into a compact sparse de Bruijn graph $SDB(Reads,Anchors^*)$ as described in Supplementary Note 10 in [107] and generates a compact disjointig-set as labels of all non-branching paths in this graph.

Step 8. Correcting errors in reads and constructing the graph $DB(Reads^*,K)$ on the error-corrected read-set $Reads^*$ using a larger K-mer size.

Because an error in a single position of a read triggers an error in each *k*-mer covering this position, and because a typical HiFi read has one error per 500 nucleotides on average, the

fraction of correct k-mers (among all k-mers in reads) becomes rather low when k exceeds 500, resulting in a complex de Bruijn graph of reads that does not adequately represent the genome. Ideally, we would like to construct the de Bruijn graph using as large a k as possible—for example, $k = 15,000$, slightly below the typical read-length in the T2T dataset. However, this approach results in a highly fragmented de Bruijn graph because reads in this dataset do not span a large fraction of genomic 15,000-mers. Although reducing k to, say, 5,000 addresses this complication (nearly all genomic 5,000-mers are spanned by reads), most 5,000-mers in reads are erroneous, preventing their assembly.

LJA attempts to minimize the effect of (1) errors in reads and (2) incomplete coverage of genomic k-mers by constructing and error-correcting the de Bruijn graphs with a small k-mer size and a large K-mer size with default values $k = 501$ and $K = 5,001$. As described in the Results, this two-round error-correction results in a nearly error-free read-set $Reads^*$ (Fig. 5.2).

mowerDBG uses the de Bruijn graph of HPC reads for detecting errors in these reads. Because ˜69% of 501-mers in HPC reads are correct, the correct 501-mers have high coverage, in contrast to low-coverage erroneous 501-mers that form bulges and tips (with the exception of k-mers that contain some tandem dinucleotide repeats discussed in Supplementary Note 11 in [107]). Afterwards, mowerDBG uses the *path-rerouting* and *bulge-collapsing* error-correction approaches to simultaneously correct reads and the graph.

Even though the previous paragraph might create an impression that errors in HiFi reads can be corrected by simply applying error-correcting approaches developed for short reads (for example, from SPAdes assembler), correction of HiFi reads faces unique challenges that we outline below. First, our goal is to correct reads (rather than to correct the de Bruijn graph as in SPAdes) because we need to rescue correct k-mers for the second round of error correction with a large K-mer size. Second, we need to perform nearly perfect error correction even in highly repetitive regions (for example, centromeres) that short-read assemblers do not even try to assemble. Third, in difference from short-read assembly, the target k-mer size (501) is a small fraction of a typical read-length (15,000). As a result, analyzing a bulge in a highly repetitive

region requires not only analysis of the graph structure (like in SPAdes) but also analysis of all read-paths traversing this bulge.

To address these complications, mowerDBG complements path-rerouting and bulge-collapsing by additional steps referred to as correcting dimers and correcting pseudo-correct reads. Supplementary Note 11 in [107] describes the mowerDBG algorithm.

Step 9: Transforming the de Bruijn graph into the multiplex de Bruijn graph

Below we describe a graph transformation algorithm for transforming the graph *DB(Reads, k)* into *DB(Reads,k$^+$)* for $k^+ > k$ by iteratively increasing the k-mer size by 1 at each iteration. Although launching jumboDBG to construct $DB(Reads, k)$, followed by these transformations, takes more time than simply launching jumboDBG to construct $DB(Reads, k^+)$, we use it as a stepping stone toward the multiplex de Bruijn graph construction.

Below we consider graphs, where each edge is labeled by a string and each vertex w is assigned an integer $vertexSize(w) \geq k$. We limit attention to graphs where suffixes of length $vertexSize(w)$ for all incoming edges into w coincide with prefixes of length $vertexSize(w)$ for all outgoing edges from w. We refer to the string of length $vertexSize(w)$ that represents these prefixes/suffixes as the label of the vertex w. We consider graphs with specified edge-labels (vertex-labels can be inferred from these edge-labels) and assume that different vertices have different vertex-labels. We will start by analyzing graphs with the same vertex size for all vertices and will later transition to the multiplex de Bruijn graphs that have vertices of varying sizes.

Transition-graph

Let *Transitions* be an arbitrary set of pairs of consecutive edges (v,w) and (w,u) in an edge-labeled graph G. We define the *transition-graph G(Transitions)* as follows. Every edge e in G corresponds to two vertices e_{start} and e_{end} in *G(Transitions)* that are connected by a blue edge that inherits the label of the edge e in G (Fig. 5.4, a). We set $vertexSize(e_{start}) = vertexSize(e_{end}) = k+1$ (vertex-labels are uniquely defined by the (k + 1)-suffixes/prefixes of the incoming/outgoing edges in each vertex). If an edge e in G is labeled by a (k + 1)-mer,

the corresponding blue edge in *G(Transitions)* is collapsed into a single vertex $e_{start} = e_{end}$. In addition to blue edges, each pair of edges $in = (v,w)$ and $out = (w,u)$ in *Transitions* adds a red transition edge between in_{end} and out_{start} to the transition graph. The label of this edge is defined as a $(k+2)$-mer formed by the concatenate $symbol_{-(k+1)}(in) \cdot label(w) \cdot symbol_{(k+1)}(out)$, where $symbol_i(e)$ stands for the i-th symbol of *label(e)*, and $symbol_{-i}(e)$ stands for the i-th symbol from the end of *label(e)*.

Path-graph

We say that a path *traverses* a vertex w in a graph if it both enters and exits this vertex. Given a path-set *Paths* in a graph, we denote the set of all paths containing an edge (v,w) as *Paths(v,w)* and the set of all paths traversing a vertex w as *Paths(w)*. We define the set *Transitions(Paths)* as the set of all pairs of consecutive edges in all paths from *Paths*. A path-graph *G(Paths)* of a path-set *Paths* is defined as the transition-graph *G(Transitions(Paths))*.

Let *Paths* be the set of all read-paths in the compressed de Bruijn graph *G = DB(Reads,k)*. A straightforward approach to constructing the graph *G(Paths)*, which recomputes labels from scratch at each iteration, nearly doubles the path lengths at each iteration and thus faces the time/memory bottleneck. multiplexDBG avoids this time/memory bottleneck by modifying rather than recomputing the edge labels from scratch, as described in Supplementary Note 13 in [107].

Multiplex de Bruijn graph transformation

Given a path-set *Paths* in a graph G, we call edges (v,w) and (w,u) in *G paired* if the transition-set *Transitions(Paths)* contain this pair of edges. A vertex w in G is *paired* if each edge incident to w is paired with at least one other edge incident to w and *unpaired* otherwise.

The important property of *DB(Genome,k)* is that there exists a genome traversal of this graph. Given a genome traversal of the graph *DB(Reads,k)*, we want to preserve it in *DB(Reads,k + 1)* after the graph transformation. However, it is not necessarily the case because the transformation of *DB(Reads,k)* into *DB(Reads,k + 1)* might create dead-ends (each unpaired

vertex in *DB(Reads,k)* results in a dead-end in *DB(Reads,k + 1)*), thus 'losing' the genome traversal that existed in *DB(Reads,k)* (Fig. 5.4, c). Below we describe a single iteration of the algorithm for transforming *DB(Reads,k)* into the multiplex de Bruijn graph *MDB(Reads,k)* that avoids creating dead-ends whenever possible by introducing vertices of sizes $(k+1)$ in this graph.

multiplexDBG transforms each paired vertex of *DB(Reads,k)* using the graph transformation algorithm and 'freezes' each unpaired vertex by preserving its *k*-mer label and the local topology. It also freezes some vertices adjacent to the already frozen vertices even if these vertices are simple. Specifically, if a frozen vertex *u* is connected with a non-frozen vertex *v* by an edge of length *VertexSize(v)* + 1, multiplexDBG freezes *v* (Fig. 5.4, d). The motivation for freezing *v* is that, if we did not freeze it, we would need to remove the edge connecting *u* and *v* in *MDB(Reads,k)*, disrupting the topology of the graph. multiplexDBG continues the graph transformations for all paired vertices (while freezing unpaired vertices) with gradually increasing *k*-mer sizes from *k* to K^+, resulting in the multiplex de Bruijn graph *MDB(Reads,k)* with *k*-mer varying in sizes from *k* to K^+. Supplementary Note 14 in [107] illustrates that multiplex transformations might be overly optimistic (by transforming vertices that should have been frozen) and overly pessimistic (by freezing vertices that should have been transformed).

Step 10: Expanding HPC contigs

Although LJA enables an accurate LJA assembly of HPC reads into HPC contigs (edge-labels in the multiplex de Bruijn graph), these contigs have to be expanded (de-collapsed) using information about homopolymer runs in the original reads. Supplementary Note 15 in [107] describes how LJApolish expands HPC contigs.

5.6 Data availability

All described datasets are publicly available through the corresponding repositories. All HiFi data were obtained from the National Center for Biotechnology Information Se-

quence Read Archive (SRA). The SRA access codes for all datasets are specified in Supplementary Note 2 'Information about datasets' in [107]. All assemblies generated by LJA are available at https://zenodo.org/record/5552696#.YV3MkVNBxH4. The CHM13 reference (version 0.9) generated by the T2T consortium (referred to as *T2TGenome*) can be found at https://s3.amazonaws.com/nanopore-human-wgs/chm13/assemblies/chm13.draft_v0.9.fasta.gz.

5.7 Code availability

The LJA code uses the open-source libraries spoa (version 4.0.5) and ksw2 (version 4e0a1cc) and is available at https://github.com/AntonBankevich/LJA. All software tools used in the analysis and their versions and parameters are specified in the text of the paper and in Supplementary Note 9 'LJA parameters' in [107].

5.8 Acknowledgements

A. Bankevich, A. Bzikadze and P.A.P. were supported by National Science Foundation EAGER award 2032783. D.A. was supported by Saint Petersburg State University (grant ID PURE 73023672). We thank A. Korobeynikov and A. Mikheenko for useful suggestions on assembling HiFi reads using SPAdes and benchmarking.

All authors contributed to developing the LJA algorithms and writing the paper. A. Bankevich (jumboDBG and mowerDBG), A. Bzikadze (multiplexDBG) and D.A. (LJApolish) implemented the LJA algorithm. A. Bankevich benchmarked LJA and other assembly tools. A. Bankevich and P.A.P. directed the work.

Chapter 5, in full, is a reprint of the material as it appears in Bankevich, A., Bzikadze, A. V., Kolmogorov, M., Antipov, D., & Pevzner, P. A. (2022). Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature biotechnology*, 1-7. The dissertation author is one of the two lead developers of the LJA algorithm and one of the two lead authors of this paper.

Chapter 6

TandemAligner: a new parameter-free framework for fast sequence alignment

6.1 Abstract

The recent advances in "complete genomics" revealed the previously inaccessible genomic regions (such as centromeres) and enabled analysis of their associations with diseases. However, analysis of variations in centromeres, immunoglobulin loci, and other extra-long tandem repeats (ETRs) faces an algorithmic bottleneck since there are currently no tools for accurate sequence comparison of ETRs. Counterintuitively, the classical alignment approaches, such as the Smith-Waterman algorithm, that work well for most sequences, fail to construct biologically adequate alignments of ETRs. This limitation was overlooked in previous studies since the ETR sequences across multiple genomes only became available in the last year. We present TandemAligner — the first parameter-free sequence alignment algorithm that introduces a sequence-dependent alignment scoring that automatically changes for any pair of compared sequences. We apply TandemAligner to various human centromeres and primate immunoglobulin loci, arrive at the first accurate estimate of the mutation rates in human centromeres, and quantify the extremely high rate of large insertions/duplications in centromeres. This extremely high rate (that the standard alignment algorithms fail to uncover) suggests that centromeres represent the most rapidly evolving regions of the human genome with respect to their structural organization.

6.2 Introduction

The Telomere-to-Telomere Consortium (T2T) recently assembled the first complete sequence of a human genome [85] and the efforts are currently underway to generate hundreds of complete haplotype-resolved human genomes [150], [151]. This progress opens a possibility to address the long-standing questions about the variation and structure of many biomedically important and complex regions of the genome such as segmental duplications [152], and long tandem repeats [153]. Since tandem repeats rapidly accumulate changes in the copy numbers, they greatly differ across the human population and their expansion is known to be the causal factor of various diseases [154], [155]. For example, alpha satellite arrays that host centromeres are formed by extra-long tandem repeats (ETRs) representing some of the most rapidly evolving regions of the human genome [114], [110], [118]. Comparison of ETRs across the human population is a prerequisite for understanding their evolution and association with cancer and infertility [19], [20], [10], [21], [11], [9], [22], [13], [15], [18], [122], [110]. However, alignment of ETRs across various human genomes remains an open algorithmic challenge. As a result, since constructing the pancentromere represents a bottleneck for the ongoing effort to generate the human pangenome graph, centromeres and other ETRs have been excluded from the pangenome graph recently constructed by the Human Pangenome Reference (HPR) consortium [151]. Comparing the immunoglobulin loci across vertebrate species represents yet another bottleneck since the standard alignment approach fails to adequately align these biomedically important ETRs [156].

A common approach to variant calling is based on aligning reads that originated from the query genome against the target sequence [124] to genotype single nucleotide variants [157] and short tandem repeats [158], [155], [159]. Emergence of long-read sequencing technologies enabled further genotyping of structural variants that are located outside ETRs [160] but genotyping ETRs remains an unsolved problem even with long-reads [151]. Thus, de novo genome assembly using long reads is currently the only approach that generates accurate ETR sequences

[35], [75], [87], [107], [138].

Emergence of "complete genomics" [85] and "complete metagenomics" [90] is shifting the focus of read mapping from variant calling to validation of newly generated assemblies [41]. Given assembled ETRs, the difficult problem of finding variations in ETRs by read mapping is substituted by a seemingly simpler problem of aligning ETRs using the standard sequence comparison dynamic approaches such as the Smith-Waterman algorithm [161]. However, we show that it is an inadequate computational model for analyzing ETRs since the highest-scoring pairwise alignment fails to reveal the evolutionary events that made two ETRs different (for any choice of scoring parameters). The HPR Consortium recently arrived at a similar conclusion and stated that *"more work is needed to determine how best to align and represent these large repeat arrays within pangenomes, particularly as T2T assembly becomes commonplace... new methods may need to be developed to fully understand and characterize this component of the human pangenome."* [151].

We describe the TandemAligner algorithm that addresses the ETR comparison problem. A substring of a string S is classified as *rare* if its *count* in S does not exceed a small threshold, and *frequent*, otherwise. ETRs are rich with frequent substrings, e.g., most 20-mers in the human centromere X occur over a thousand times in this centromere [34], [85], [110]. Thus, deciding whether a match between two occurrences of a frequent substring in two ETRs is challenging as the total number of such matches is often measured in millions.

The standard alignment algorithms apply the same scoring parameters (match/mismatch/indel penalty, affine gap penalties, etc.) to all pairs of sequences. Although they construct a biologically adequate alignment for most sequences (wrt their evolution), below we show that some sequences (e.g., ETRs) are less amenable to this approach for any choice of alignment parameters. The inherent limitation of the standard alignment approach is that less significant matches of frequent k-mers (that may have millions of matches between two ETRs) have the same contribution to the score as the more significant matches of rare k-mers (that may have a single match between two ETRs). This limitation was overlooked in the previous studies since

the ETR sequences across multiple genomes only became available in the last year.

TandemAligner introduces a novel parameter-free and sequence-dependent alignment scoring that automatically changes for any pair of compared sequences and that does not rely on the multiple scoring parameters used in the standard alignment. The sequence-dependent alignment prioritizes matches of rare substrings since they are more likely to be relevant to the evolutionary relationship between two sequences. Analysis of the constructed rare-alignments reveals new biological phenomena such as the extremely high rate of large insertion/deletion in centromeres. Since the T2T Consortium did not have bioinformatics tools for deriving the detailed history of indels in centromeres, its analysis of the first two assembled human centromeres resulted in a conclusion that they are highly concordant apart from the three regions with recent large indels (Figure 5D in [110]). We show that these centromeres differ from each other by over 300 large deletions and duplications of length at least 2 kb, including six extra-long indels varying in length from 23 to 76 kb. This extremely high rate of large duplications and deletions (that the standard alignment algorithms fail to uncover) suggests that centromeres represent the most rapidly evolving regions of the human genome with respect to their structural organization.

Analysis of rare-alignments also leads to the first accurate estimate of the single-nucleotide mutation rates in human centromeres. Previous studies came to the conclusion that the rate of single-nucleotide mutations is greatly elevated in centromeres [162], [163]. However, in the absence of bioinformatics tools for accurate ETR alignments, it is nearly impossible to identify orthologous repeat copies, a prerequisite for estimating single-nucleotide mutation rates. Rare-alignments revealed that, contrary to existing assumption, the rate of single-nucleotide mutations (and small indels) in centromeres does not exceed the average mutation rate of the human genome.

We demonstrate that TandemAligner not just improves on the standard alignment approach with respect to comparing ETRs (human centromeres and primate immunoglobulin loci) — instead it often generates completely different alignments. Although TandemAligner was

developed for comparing highly-repetitive sequences, we show that it results in fast and accurate comparison of any sequences with percent identity exceeding 70% and even slightly outperforms the standard alignment approach (with respect to accuracy) in the case of highly similar sequences with percent identity exceeding 90%. We further discuss theoretical properties of the alignment problem and suggest that the scoring scheme should be selected with respect to the underlying evolutionary model that is presumed to have produced the aligned sequences. Although the standard scoring scheme for sequence alignment often performs well in practice, it makes some implicit assumptions that do not hold for ETRs. While the new rare-alignment scheme is not necessarily induced by any natural evolutionary model, rare-alignments of centromeres and immunoglobulin loci appear to be closer to the "real alignment" than the traditional alignments.

6.3 Results

6.3.1 Outline of the TandemAligner algorithm

The codebase of TandemAligner is available at https://github.com/seryrzu/tandem_aligner. Figure 6.1 illustrates some (but not all) steps of TandemAligner.

A substring of a string S is called *unique* if it appears exactly once in S. TandemAligner uses the Longest Common Prefix array [164] to find the shortest unique substring starting at each position in the string S. A shortest unique substring P of a string S is called an *anchor* if all proper substrings of P are non-unique in S. Given strings S and T, TandemAligner efficiently computes the set of all anchors shared by S and T (this set is usually small) and uses sparse dynamic programming to rapidly construct an optimal path through these anchors in the standard grid-like sequence alignment graph. It further analyzes the resulting alignment-path *Alignment* formed by diagonal edges representing anchors and horizontal/vertical indel edges. We refer to a segment of *Alignment* consisting of a deletion-run immediately followed by an insertion-run (or vice versa) as an *indel-pair*. Each indel-pair corresponds to a pair of substrings s and t in S and T, respectively. TandemAligner recursively constructs an alignment of strings s and t, and

substitutes the indel-pair in *Alignment* by the resulting (small) alignment-path (this recursive step is not illustrated in Figure Algorithm). The process terminates when there are either no indel-pairs left or the set of anchors constructed on substrings corresponding to each remaining indel-pair is empty.

Sequence architecture of human centromeres. Alpha satellite repeats are the building blocks of centromeric repeats and form roughly 3% of the human genome [110]. Consecutive alpha satellite repeats are arranged into higher-order repeat (HOR) units that are repeated hundreds or thousands of times in each centromere. Individual alpha satellites within a higher-order unit show low (50–90%) sequence identity to each other while HORs within a single centromere show high (95–100%) sequence identity (Figure 6.2). Organization and nucleotide sequence of HORs is specific for a particular chromosome.

6.3.2 Datasets

We extracted the assembled satellite centromere on chromosome X (cenX) from the public release v2.0 by the Telomere-to-Telomere (T2T) consortium of the complete assembly for the effectively haploid CHM13 cell line (GCA_009914755.4), [34], [84], [85]. Specifically, we use the following coordinates: chrX:57,817,899-60,927,196. We also extracted cenX from the recently assembled HG002 human sample (CP086568.2, coordinates: chrX:57,860,000-61,000,000). We refer to these two centromeres as $cenX_1$ and $cenX_2$. $cenX_1$ of total length 3,109,297 bp is formed by 1533 HORs and $cenX_2$ of total length 3,140,000 bp is formed by 1539 HORs.

Similar lengths of $cenX_1$ and $cenX_2$ may create a wrong impression that the lengths of centromeres is conserved across the human population. It is known that each human centromere widely varies in length across different human genomes [110] and it is just a coincidence that the first two assembled human centromeres have similar lengths.

We also extracted the region of the heavy immunoglobulin locus of the human genome (referred to as $IGHD_H$; NC_000014.9:105,865,737-105,964,717) and orangutan genome (referred

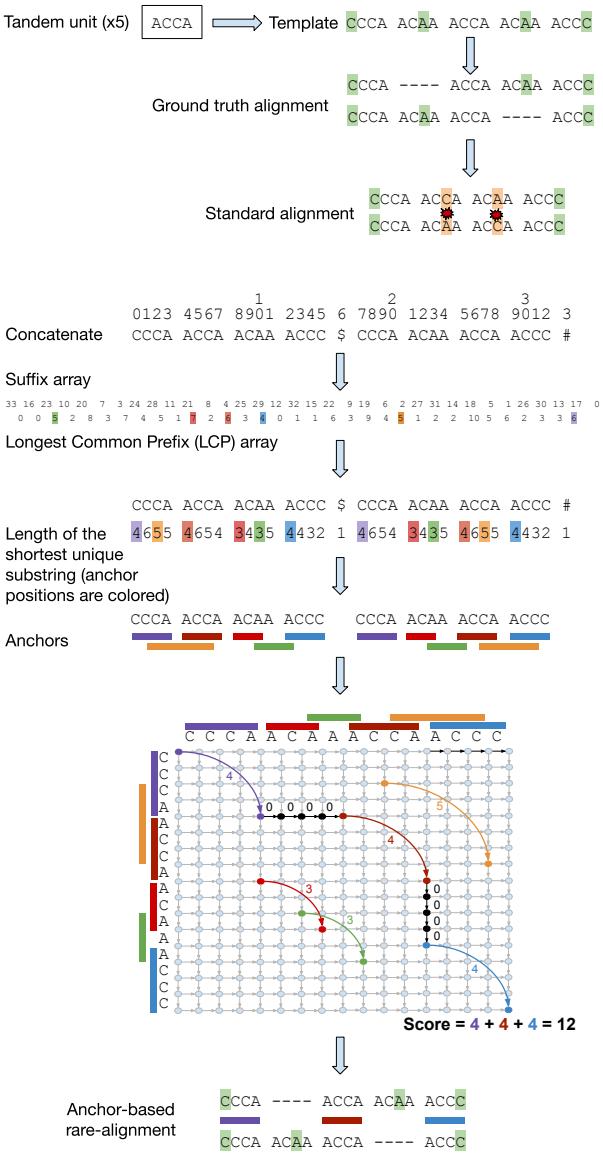


Figure 6.1. TandemAligner pipeline. Aligning ETRs S=CCCAACCAACAAACCC and T=CCCAACAAACCAACCC using TandemAligner. These four-unit ETRs evolved from a five-unit ancestral ETR CCCAACAAACCAACAAACCC by deletion of its second unit in S and its fourth unit in T. The standard alignment generates high-scoring but incorrect alignment between S and T. TandemAligner constructs the suffix array and the Longest Common Prefix (LCP) array of the concatenate $S\$T\#$ (with delimiters "$" and "#") and uses these arrays to rapidly identify shortest unique substrings and anchors shared by S and T. Afterward, it constructs a (sparse) anchor graph and finds an optimal alignment-path in this graph using sparse dynamic programming. Finally, it recursively applies the same procedure to all substrings of S and T that form a pair of consecutive insertion-deletion or deletion-insertion edges in the constructed alignment-path (this step is not shown).

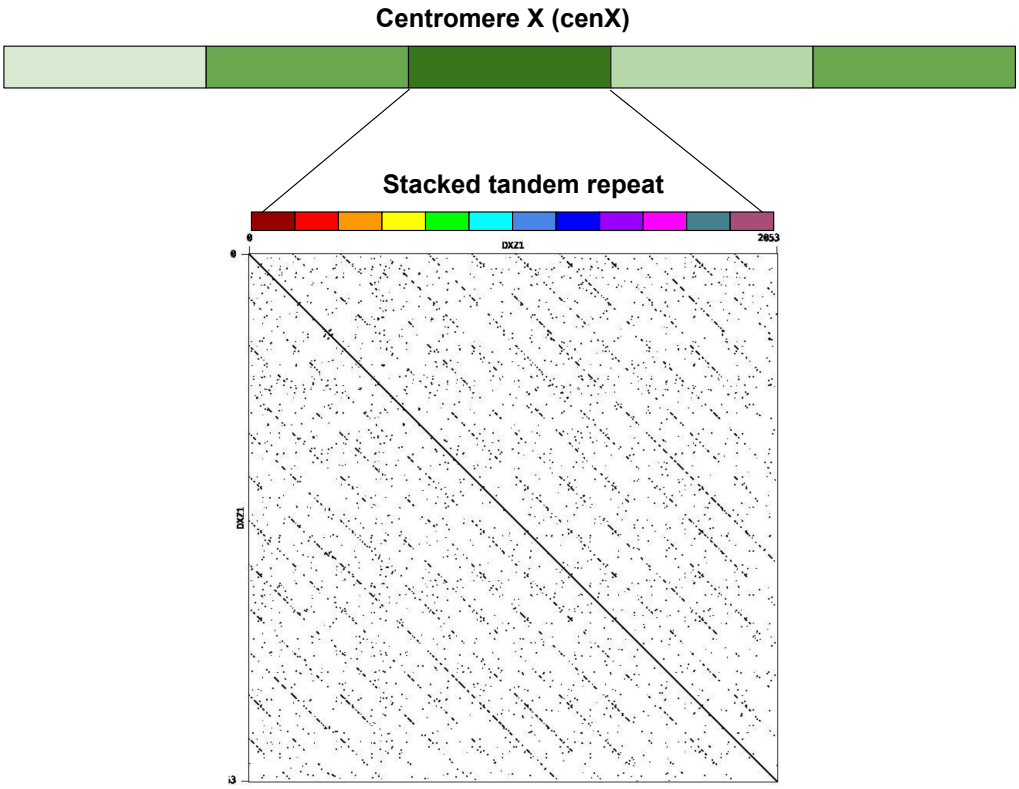


Figure 6.2. The architecture of centromere on Chromosome X. The centromere of human chromosome X (cenX) consists of ˜18100 monomers of length ˜171 bp each based on the cenX assembly in [35](the T2T assembly [85] represents a minor change to this assembly). These monomers are organized into ˜1500 units. Five units in the Figure are colored by five shades of green illustrating unit variations. Each unit is a stacked tandem repeat formed by various monomers. The vast majority of units in cenX correspond to the canonical HOR which is formed by twelve monomers (shown by twelve different colors). The figure on top represents the dot plot of the nucleotide sequence of the canonical HOR that reveals twelve monomers. While the canonical units are 95–100% similar, monomers in cenX are only 65–88% similar. In addition to the canonical 12-monomer units, cenX has a small number of partial HORs with varying numbers of monomers.

to as $IGHD_O$; NC_036917.1:87,805,787-87,899,312) that contains D genes in the IGHD loci.

6.3.3 The key limitation of the standard sequence alignment

The orange path in Figure 6.3, left, shows the highest-scoring alignment path between centromeres $cenX_1$ and $cenX_2$ constructed by the standard dynamic programming algorithm for sequence comparison. This alignment suggests that these two centromeres are very similar with 98% percent identity (28,421 bp are insertions or deletions, 37,975 bp are mismatches, and 3,057,465 bp are matches). However, since centromeres widely vary in length across the human population [110], this is likely a wrong conclusion that is affected by selecting two centromeres that coincidentally happened to be similar in length.

Below we demonstrate that the orange alignment path in Figure 6.3, left, does not reveal the true sequence of events on the evolutionary path between these centromeres. The blue path in Figure 6.3, left shows the rare-alignment path between $cenX_1$ and $cenX_2$ constructed by TandemAlignment. This path illustrates a very different and complex evolutionary scenario with only 1,954,622 matched positions and 2,335,879 insertions and deletions — surprisingly, the orange and blue paths in Figure 6.3, left, do not coincide at any of their matching positions!

To illustrate limitations of the standard alignment using a simpler example, we generated the HOR decomposition of $cenX_1$ [118] and extracted ten HOR-blocks at coordinates $cenX_1$:1,222,223-1,242,795 resulting in a sequence of length 20,572 bp referred to as a *Template*. We remove the third (eighth) HOR block at coordinates *Template*:4,112-6,172 (14,405-16,461), and refer to the resulting sequence as $Template_{-3}$ ($Template_{-8}$). The evolutionary correct alignment of $Template_{-3}$ against $Template_{-8}$ should align HORs 1, 2, 4, 5, 6, 7, 9, and 10, delete HOR 3, and insert HOR 8.

Since sequences $Template_{-3}$ and $Template_{-8}$ are very similar (edit distance is 53 with only six gap-symbols) (Figure 6.3, right), the standard alignment fails to reveal the eight related HORs that they share, illustrating that it does not adequately represent the correct evolutionary scenario of ETRs. Below we propose an alternative approach that leads to the correct alignment

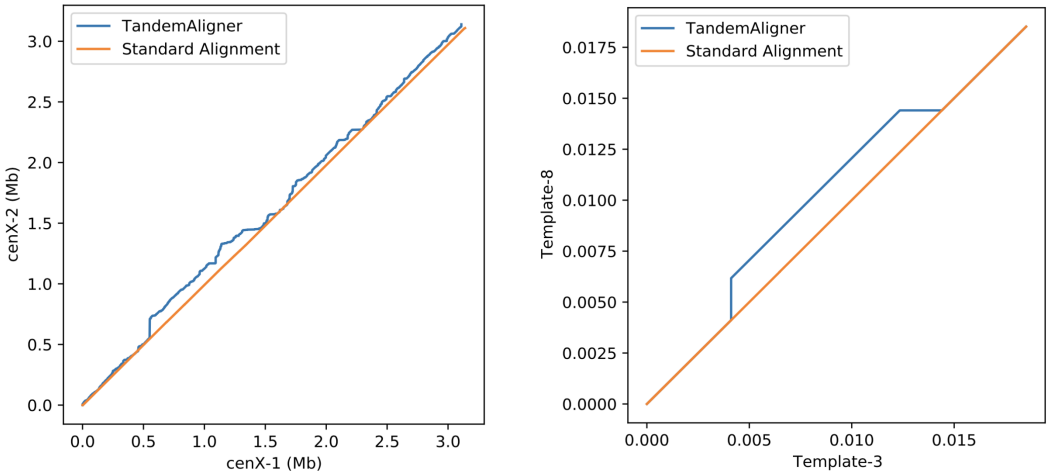


Figure 6.3. The highest-scoring alignment path between centromeres $cenX_1$ and $cenX_2$ (left) and strings $Template_{-3}$ and $Template_{-8}$ (right) constructed by the standard alignment algorithm (orange path) and by TandemAligner (blue path). The blue path is not seen at positions where it coincides with the orange path. The standard alignment was constructed using the edlib tool [165] with parameters match score 0, mismatch score -1, indel score -1 (changing parameters does not significantly change the alignment). The alignment of $Template_{-3}$ and $Template_{-8}$ is represented by the CIGAR string 4112M 1D 2354M 3I 2M 1I 7933M 1I 4111M. Other state-of-the-art sequence comparison tools result in similar alignments.

of $Template_{-3}$ and $Template_{-8}$. To motivate the TandemAligner algorithm, we first describe a simple parameter-dependent TandemAlignerk algorithm that uses rare k-mers for a fixed k-mer size. TandemAligner is a parameter-free algorithm that improves on TandemAlignerk by using rare k-mers of varying sizes and removing all parameters.

6.3.4 Sequence-dependent alignment scoring based on rare k-mers

A k-mer in a string S is called *rare* in S if its *count* in S does not exceed a threshold *MaxCount*. A k-mer occurring in strings S and T is called *rare* in S and T if it is rare in both S and T. Other k-mers occuring in either S or T are referred to as *frequent*. Below we describe a simple alignment algorithm (referred to as TandemAligner$_k$) based on sequence-dependent scoring that uses rare k-mers.

Given an alphabet A, we denote the alphabet of all k-mers from A as A^k. Given a string S in the alphabet A, we denote the sequence of its $|S|-k+1$ k-mers (written in the alphabet

A^k) as S^k. For example, for $S = ACGT$, $S^2 = AC, CG, GT$. TandemAlignerk transforms strings S and T into strings S^k and T^k and removes all frequent k-mers from these strings, resulting in strings S^{k*} and T^{k*} that consist of rare k-mers in S and T. Afterward, it constructs a *longest common subsequence (LCS)* between S^{k*} and T^{k*}. Equivalently, TandemAligner$_k$ finds an optimal alignment between S^k and T^k using the mismatch penalty equal to infinity and the indel penalty equal to zero. The premium for a match is k-mer-dependent and is defined as 1 (0) if the k-mer is rare (frequent) in S and T.

TandemAligner$_k$ constructs a k-mer-level alignment. Section "From a k-mer-level alignment to a nucleotide-level alignment" in Methods describes how to transform it into a regular nucleotide-level alignment.

6.3.5 Dot plots based on rare k-mers

The standard dot plots [166] have limited utility in ETRs since they are dominated by frequent k-mers that do not reveal the evolutionary relevant similarities between ETRs. Below we modify the concept of a dot plot to make it better suited for visualizing similarities between ETRs.

Given a pair of sequences S and T and a string-set X, a $DotPlot(S,T;X)$ is defined as a scatter plot such that each occurrence of every string P in X with starting coordinate i (j) in S (T) corresponds to a line connecting two points with coordinates (i, j) and $(i+|P|, j+|P|)$. In the case when the string-set X consists of all rare k-mers in S and T for a threshold *MaxCount*, we refer to the set $DotPlot(S,T;X)$ as $DotPlot_{k,MaxCount}(S,T)$. This definition of a dot plot differs from the standard definition as $DotPlot_{k,MaxCount}(S,T)$ only depicts rare k-mers shared by S and T, and reverse-complementary k-mers are treated as different k-mers.

Figure 6.4 present $DotPlot_{k,MaxCount}$*(Template_-3, Template_-8)* for various values of parameters and illustrates that TandemAligner$_k$ reveal the correct evolutionary scenario between Template$_{-3}$ and Template$_{-8}$ (except for the case $k = 10$ and *MaxCount*$= 10$that we will address later). Figure 6.5 DotPlotsCenXk shows dotplots DotPlotk,MaxCount($cenX_1$, $cenX_2$) and

the corresponding alignments constructed by TandemAlignerk (for various values of k and MaxCount) that reveal a complex evolutionary history of centromere X.

6.3.6 Limitations of scoring based on rare k-mers of fixed size

Even though TandemAlignerk is a fast sequence comparison algorithm, it is unclear how to select a parameter k. On one hand, reducing its value ensures that one detects all shorter rare matches that cannot be extended either downstream or upstream. On the other hand, increasing the value of k provides reassurance that the detected matches reflect the evolutionary relationship between sequences rather than a random coincident match. Simultaneously, a rare match that is longer than k might not contain any rare k-mers as its substrings and thus will not be detected. Even though, for example in Figure 6.4, such a dilemma is not a critical concern (except for the case $k = 10$ and $MaxCount$ = 10), it remains unclear if an optimal choice for parameter k exists for real ETRs. Moreover, various regions of ETRs likely have varying optimal values of parameters k.

The last example in Figure 6.4 ($k = 10$ and $MaxCount$ = 10) motivates development of an alignment approach that does not rely on specific parameters k and $MaxCount$. TandemAligner improves on TandemAligner$_{\text{k}}$ and reconstructs the correct evolutionary scenario between $Template_{-3}$, and $Template_{-8}$ by using an alternative parameter-free scoring approach.

6.3.7 Shortest rare substrings

A substring P of a string S is called *n-rare* if $count_P(S) = n$ and *unique* if $count_P(S) = 1$ ($count_P(S)$ refers to the number of occurrences of a substring P in a string S). For each n such that $1 \leq n \leq MaxCount$, we consider the shortest n-rare substring starting at position i and denote its length as $rare_i(S,n)$. In the case when there is no n-rare substring starting at position i, we assign $rare_i(S,n) = \infty$ (there exists a 1-rare substring starting at each position since we add a unique "$" symbol to mark the end of a string). Figure 6.6 illustrates that $rare_i(cenX_1,1)$ varies widely across the centromere. Positions located within recent duplications in cenX typically result in

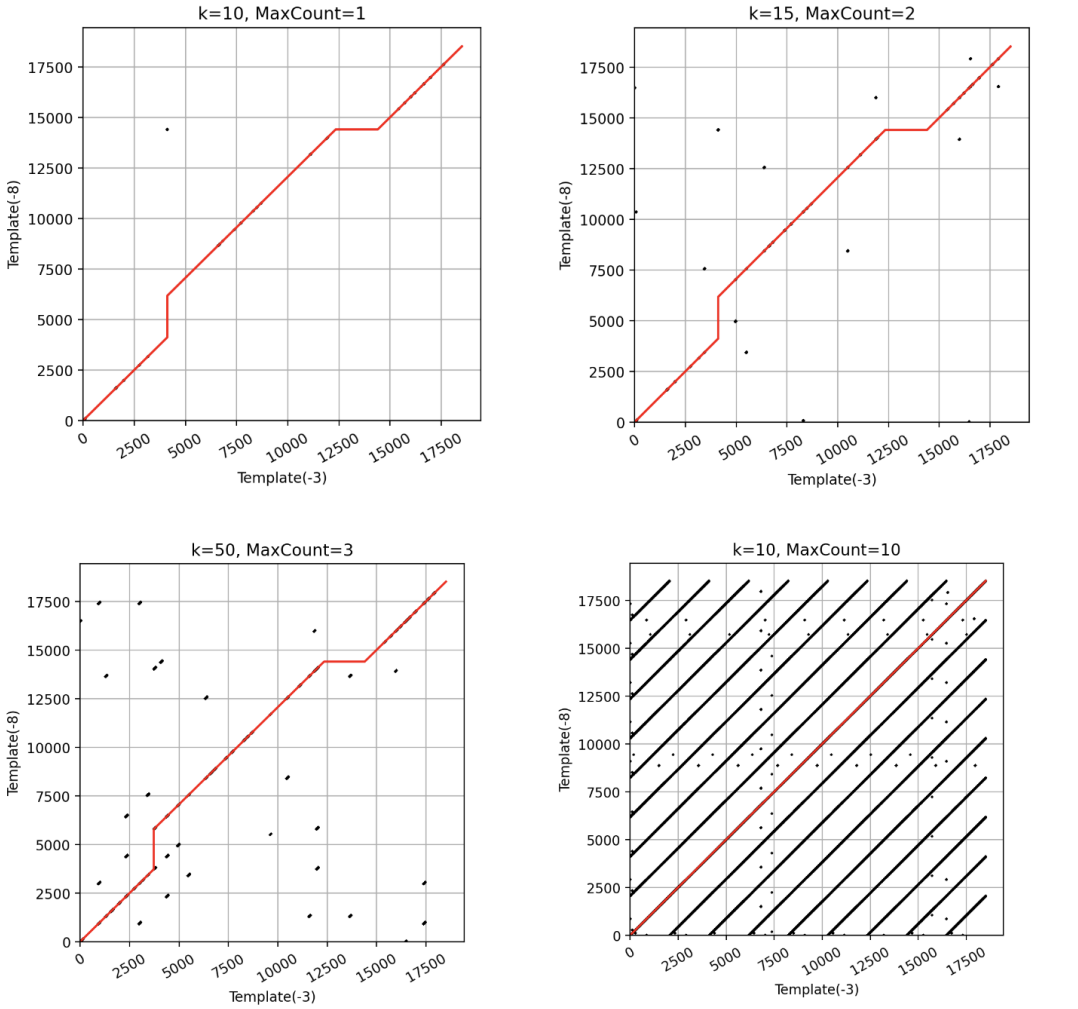


Figure 6.4. The red path corresponds to the optimal alignment constructed by TandemAligner$_k$. In three cases, rare-alignment identifies the correct evolutionary scenario and reveals two indels that correspond to the third (eighth) HOR of *Template*$_{-8}$ (*Template*$_{-3}$) that is missing in *Template*$_{-3}$ (*Template*$_{-8}$). The rare-alignment with parameters $k = 10$ and *MaxCount* = 10 does not reveal the correct evolutionary scenario.

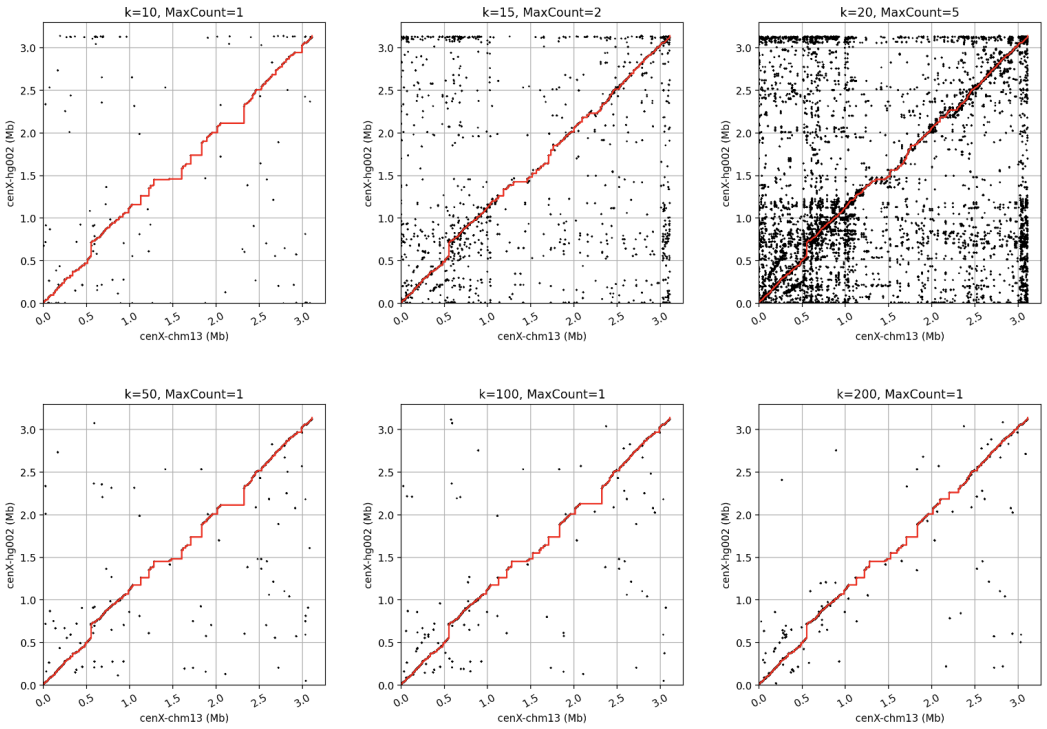


Figure 6.5. *DotPlot$_{k,MaxCount}$(cenX$_1$, cenX$_2$)* for various values of parameters *k* and *MaxCount*. The red path corresponds to the optimal alignment constructed by TandemAligner$_k$.

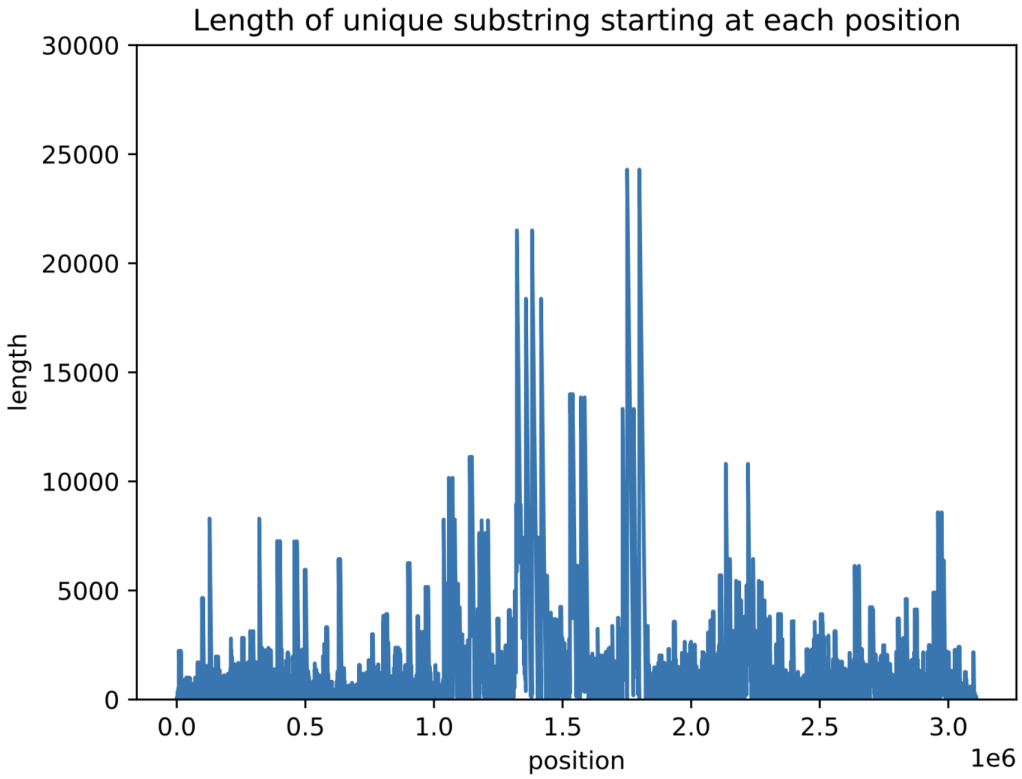


Figure 6.6. The array $rare_i(cenX_1, 1)$. Although the mean value of $rare_i(cenX_1, 1)$ is only 1,736, $rare_i(cenX_1, 1)$ exceeds 10,000 for 94,976 positions in $cenX_1$.

high values of $rarei(cenX_1, 1)$. Section "Algorithm for computing shortest rare substrings" in Methods describes an algorithm for computing $rare_i(S, n)$ for all $1 \leq i \leq |S|$ given a parameter n in $O(|S|)$ time.

6.3.8 Anchors

An n-rare substring P of S is called an *n-anchor* if no (proper) substring of P is an n-rare substring of S. Given a threshold n and array $rare_i(S, n)$ for all i, we can compute the set of the starting/ending positions $Anchor_n(S)$ of all n-anchors in $O(|S|)$ time. Since the complexity of constructing the array $rare_i(S, n)$ is linear, the resulting complexity of finding the set $Anchor_n(S)$ in a string S is also linear. Since the set of n-anchors is typically much smaller than the set of n-rare substring (e.g., $cenX_1$ and $cenX_2$ share only 9,708 1-anchors), TandemAligner uses an

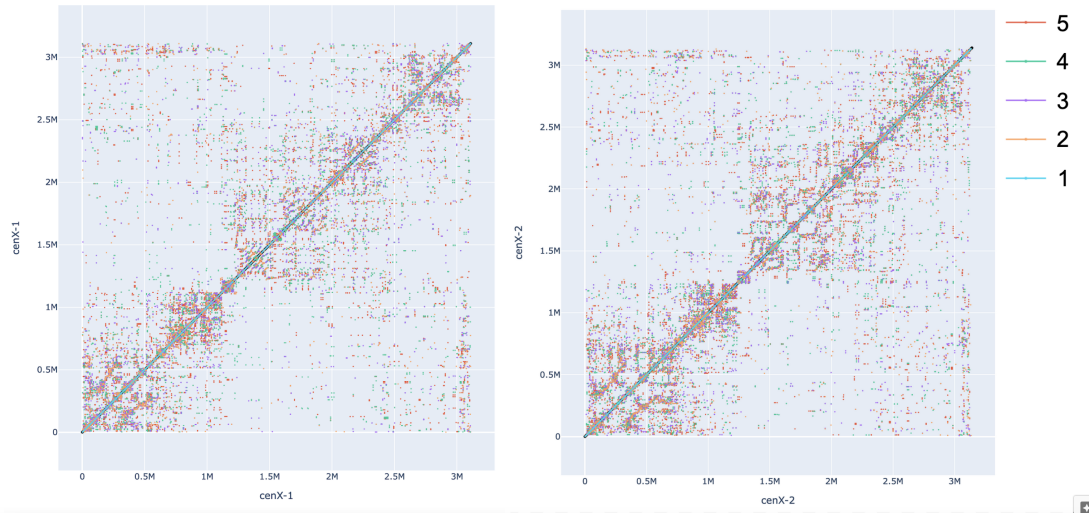


Figure 6.7. The superposition of n-rare dot plots $DotPlot_n(cenX_1)$ (left) and $DotPlot_n(cenX_2)$ (right) for $n = 2, 3, 4, 5$. Each of four dotplots is shown using an individual color.

anchor-based scoring.

Given a string S, we use the shorthand $DotPlot_n(S)$ for $DotPlot(S, S, Anchors_n(S))$. Figure 6.7 combines the dot plots $DotPlot_n(cenX_i)$ for $n = 2, 3, 4, 5$ and reveals the complex (and cryptic) evolutionary history of insertions/deletions in $cenX_1$ and $cenX_2$. Note that $DotPlot_n(cenX_1)$ and $DotPlot_n(cenX_1)$ share many anchors (reflecting duplications that happened before they diverged) but also a significant number of different anchors that reflect recent duplications in each of these centromeres.

Since two dot plots in Figure 6.7 look similar (with clearly visible rectangles that have high density of points), one may arrive to a conclusion that $cenX_1$ and $cenX_2$ have nearly identical architectures. Since the T2T Consortium did not have tools for deriving the detailed history of indels in centromere evolution it came to this conclusion by analyzing dot plots generated by the StainedGlass tool [152]. Specifically, it concluded that $cenX_1$ and $cenX_2$ are highly concordant apart from the three regions with recent insertions and deletions (Figure 5D in [110]). Below we show that $cenX_1$ and $cenX_2$ differ from each other by over 300 large duplications and deletions of entire HORs (canonical HOR unit in centromere X has length 2057 bp). Moreover, six of them exceed 20 kb in length and include 11, 12, 20, 22, 25, and even 37 HOR units.

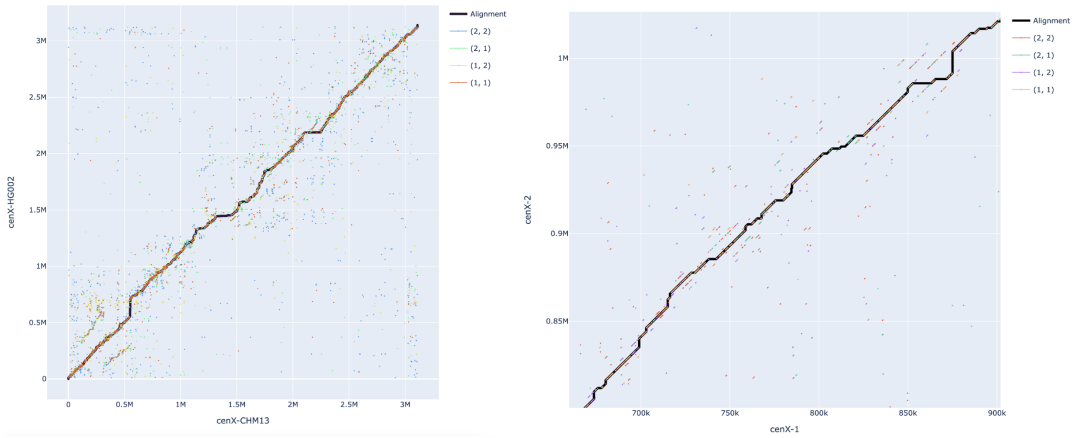


Figure 6.8. The anchor-based alignment of $cenX_1$ and $cenX_2$ and $DotPlot(cenX_1, cenX_2, Anchors_{n,m}(cenX_1, cenX_2))$ for $(n,m) = (1,1),(1,2),(2,1),(2,2)$. (Left) Anchor-based dot plot and alignment of the entire centromeres. (Right) Zoomed subrectangle shows the anchor-based dot plot of substrings of $cenX_1$ and $cenX_2$ of length approximately 220 kb and reveals that many lines in this dot plot aggregate into lines $j = i + t|HOR|$.

Given strings S and T, an (n,m)-*anchor* is an n-anchor in S and an m-anchor in T. We define $Anchors_{n,m}(S,T)$ as the set of all (n,m)-anchors in strings S and T. Finding the set $Anchors_{n,m}(S,T)$ for two strings S and T is analogous to finding the set of anchors for a single string and can be done in $O(|S|+|T|)$ time. Figure 6.8, left, combines $DotPlot(cenX_1, cenX_2, Anchors_{n,m}(cenX_1, cenX_2))$ for four different values of n and m.

6.3.9 Anchor-based alignment graph

TandemAligner modifies the standard alignment graph (see section "Standard alignment graph" in Methods) by removing/adding some edges and modifying the scoring approach. First, it removes all diagonal edges from the graph and assigns weight 0 to the remaining edges. Given a (positive) integer parameter *MaxCount*, we consider *AllAnchors = AllAnchors(S, T, MaxCount)* — the set of all occurrences of (n,m)-anchors in S and T for $1 \le n,m \le MaxCount$. For an (n,m)-anchor P, such that $P = S[i : i+|P|] = T[j : j+|P|]$, we add an edge connecting vertices (i,j) and $(i+|P|, j+|P|)$ and assign it the weight equal to $|P|/(n \cdot m)$. We denote $K = \sum_{P \in AllAnchors} count_P(S) \cdot count_P(T)$ — the number of diagonal edges in this graph.

The standard dynamic programming algorithm finds the heaviest path in this graph in $O(|S| \cdot |T|)$ time. However, since all vertical and horizontal edges have weight 0, one can find this path in $O(K^2)$ time by conducting the sparse dynamic programming computation only for the starting/ending vertex of each diagonal edge [167]. Although TandemAligner uses this simple approach, more advanced algorithms solve this problem in $O(K \log K)$ time, resulting in further speed-up [168]. In the case of centromere comparison, K is usually small, e.g., for rare-alignment of $cenX_1$ against $cenX_2$ with $MaxCount = 1$, $K = 9708$. The next subsection explains why such a low value of K is sufficient for constructing accurate alignments.

Since the set *AllAnchors* can be computed in $O(MaxCount^2 * (|S| + |T|))$ time, the resulting running time of the anchor-based alignment algorithm with a single parameter *MaxCount* is $O(MaxCount^2 * (|S| + |T|) + K^2)$. Figure 6.8, left, shows an anchor-based alignment of $cenX_1$ and $cenX_2$.

6.3.10 Recursive anchor-based rare-alignment

The only trade-off of the anchor-based alignment is the choice of parameter *MaxCount*. On one hand, a lower value of the parameter *MaxCount* leads to more sparse alignment with fewer matched bases. On another hand, higher values of the parameter *MaxCount* increase the number of (n,m)-anchors and introduce a computational burden since the number of diagonal edges in the anchor-based alignment graph increases.

We call a sequence of consecutive matches (insertions, deletions) in a rare-alignment as a match-run (insertion-run, deletion-run). For example, the anchor-based alignment of $cenX_1$ against $cenX_2$ with *MaxCount* = 1(5), has 1,447,947 (1,790,434) matched bases forming 865 (1223) match-runs, 1,692,054 (1,349,567) insertions forming 811 (1141) insertion-runs, and 1,661,351 (1,318,864) deletions forming 808 (1137) deletion-runs. The number of shared matching positions between these two alignments is 1,399,978.

To address the trade-off between small and large values of parameter *MaxCount*, TandemAligner constructs a parameter-free rare-alignment by introducing a recursive strategy described

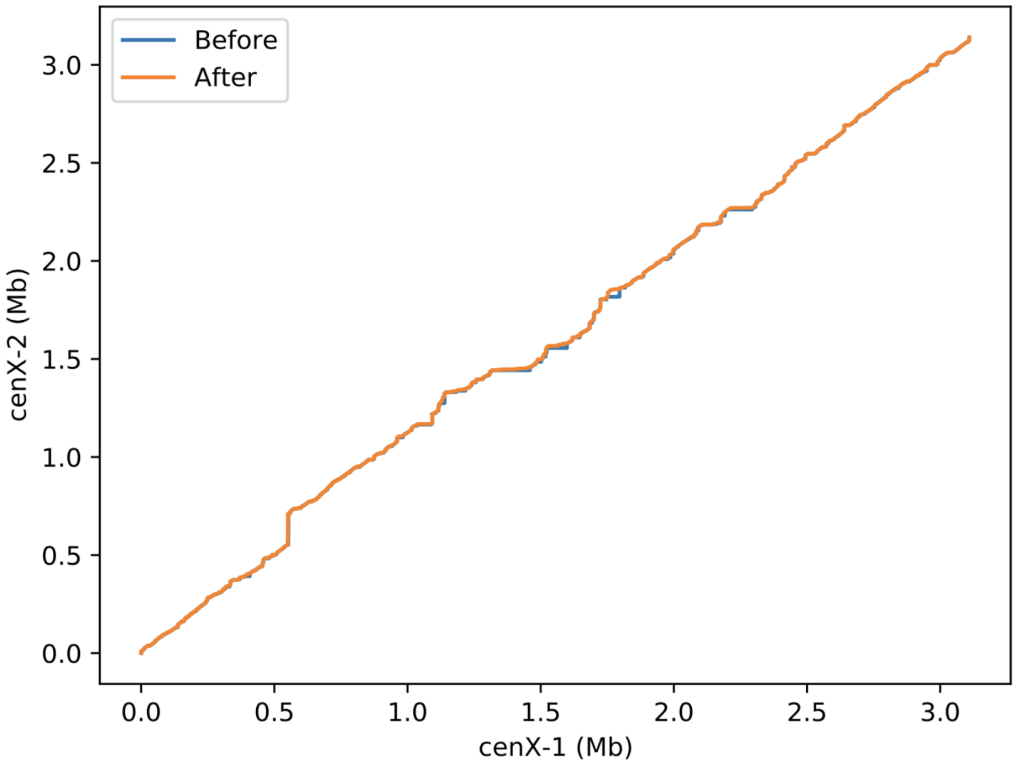


Figure 6.9. Anchor-based alignment path before applying the recursive procedure (blue) and rare-alignment after applying the recursive procedure (orange). The anchor-based alignment is constructed with parameter *MaxCount* = 1. Although the changes appear to be small for a naked eye, the recursive procedure results in many changes that are difficult to see due to the small scale of the Figure.

in Methods. The rare-alignment of $cenX_1$ against $cenX_2$ has 1,983,431 (2744) matched bases (match-runs), 1,156,570 (2479) insertion bases (insertion-runs), and 1,125,867 (2467) deletion bases (deletion-runs). This rare-alignment accounts for 535,484 increase in the number of matched bases compared to the non-recursive anchor-based alignment (Figure 6.9).

The constructed recursive rare-alignment does not include any mismatches since it represents a single-nucleotide mismatch as a single-nucleotide insertion (deletion) edge followed by a single-nucleotide deletion (insertion) edge. Section "Refining rare-alignments" in Methods describes how TandemAligner transforms some of such pairs of indel edges into mismatches.

Below we analyze the rare-alignment of human centromeres and immunoglobulin loci. Further benchmarking is provided in Supplementary Notes "Benchmarking standard alignments

and rare-alignments on simulated centromeres", "Benchmarking standard alignments and rare-alignments on non-repetitive strings" in [169].

6.3.11 TandemAligner reveals the very high rate of large deletions and duplications in centromeres

Figure 6.9, left shows the distribution of lengths of insertion-runs and deletion-runs in the rare-alignment of $cenX_1$ and $cenX_2$. Notably the prominent peaks in this distribution correspond to the length of a single canonical HOR in cenX (2057 bp) or to the lengths of multiple canonical HOR. The fact that TandemAligner automatically derived the HOR length in cenX without any prior knowledge adds confidence that it adequately represents evolution of centromeres.

For an indel *IND*, we compute *multiplicity(IND)* as the closest integer to $|IND|/|HOR|$, where $|IND|$ is the length of the indel and $|HOR|$ is the length of the canonical HOR (2057 bp for centromere X). We further define $offset(IND) = |IND| - multiplicity(IND) \cdot |HOR|$ and classify an indel *IND* as a *HOR-indel* if $offset(IND)/|HOR|$ does not exceed a threshold (the default value 0.05). The rare-alignment of $cenX_1$ and $cenX_2$ includes 175 (59, 28, 63) HOR-indels of multiplicity 1 (2, 3, more than 3) suggesting that HOR-indels formed by a single HOR dominate centromere evolution with 54% (18%, 9%, 19%) HOR-indels of multiplicity 1 (2, 3, more than 3). It will be interesting to see whether these numbers are stable for other centromeres across the human population when their sequences become available. Figure 6.9, right shows distribution of HOR-indels in the rare-alignment of $cenX_1$ and $cenX_2$ and reveals their high density over the entire length of the centromere. This rare-alignment includes 164 HOR-insertions and 161 HOR-deletions of total multiplicity 845 (total multiplicity of HOR-insertions and HOR-deletions is 457 and 388, respectively). The high total multiplicity of HOR-deletions in $cenX_1$ implies that only approximately (1533-457)/1533% = 70% of HORs in $cenX_1$ form orthologs with HORs in $cenX_2$ and thus limits the region for estimating the mutation rate to these HORs.

With about twenty thousands structural variations (SVs) per a single human genome,

[151] the rate of SVs in the human genome is estimated as roughly one SV per 150 kb on average. With 325 HOR-indels identified by TandemAligner, the rate of SVs in human centromeres may be as large as one SV per 10 kb, an order of magnitude increase. We note that [151] classified an indel as an SV if its length exceeds 50 bp while we consider large indels (with lengths exceeding 2 kb) that form less than 10% of indels identified in [151]. Thus, the rate of large SVs in human centromeres exceeds the rate of large SV in the rest of the human genome by two orders of magnitude. Interestingly, while the "live" centromeres have an extremely high indel rate, the adjacent monomeric "dead" layers formed by blocks of monomeric alpha-satellites flanking the centromeres [110] have very few indels (see Supplementary Note "Extending rare-alignments into monomeric 'dead' layers" in [169]).

6.3.12 TandemAligner reveals the low rate of single-nucleotide substitutions and small indels in centromeres

Although estimating the rate of single-nucleotide substitutions (and small indels) is a straightforward task for most genomic regions, it should be done with caution in ETRs. Constructing an accurate alignment of ETR (and limiting attention to regions of this alignment that do not include large indels) is a prerequisite for an accurate estimate of the single-nucleotide mutation rates in ETRs.

For example, the mutation rate between sequences $Template_{-3}$ and $Template_{-8}$ is zero since each of them was generated from the same sequence *Template* by a large deletion. However, the standard alignment (orange path in Figure 6.3, right) suggests that these sequences have 53 mutations, resulting in an inflated estimate of the mutation rate (53/18,500=0.0029) that exceeds the average mutation rate in the human genome. Similarly, the standard alignment of $cenX_1$ against $cenX_2$ (orange path in Figure 6.3, left) suggests that these sequences have an extremely high mutation rate (nearly 1% for single-nucleotide substitutions and over 1.2% for short indels) that exceeds the average mutation rate in the human genome by an order of magnitude. However, such high mutation rates (consistent with previous papers aimed at analyzing mutation rates in

centromeres [162], [163] are merely artifacts of the incorrect alignments.

Moreover, even with a correct alignment, estimation of the mutation rates should be done with caution after limiting attention to the regions that do not include large indels. For example, after removing regions corresponding to two large indels in the rare-alignment between *Template_3* and *Template_8* (blue path in Figure 6.3, right), we are left with identical regions that result in the correct 0% estimate of the mutation rate between *Template_3* and *Template_8*.

We call an indel *short* if its length does not exceed a threshold *ShortIndel*, and *long*, otherwise (default value *ShortIndel* = 5). The rare-alignment of $cenX_1$ and $cenX_2$ contains 268 short indels (of total length 473) and 520 long indels (of total length 2,277,742). Removing all long indels from the alignment-path of $cenX_1$ and $cenX_2$ breaks it into a set of shorter paths (with total length L^- =1,986,015) that contains $M = 2107$ mismatches and 268 short indels of total length $I = 473$. We use this path-set to estimate the rate of mismatches and (short) indels as $M/L^- \approx 1/1000$ bp and $I/L^- \approx 2/10,000$, respectively. Thus, analysis of rare-alignments suggests that previous studies may have come to the incorrect conclusion that the rate of single-nucleotide mutations is greatly elevated in centromeres [162], [163].

[84] recently came to a conclusion that the single-nucleotide mutation rate is elevated in centromeres by analyzing cen8 (centromere on chromosome 8) in human and primate genomes. In the absence of accurate algorithms for aligning HOR arrays, they arrived at this conclusion by analyzing easier-to-align blocks of monomeric alpha-satellites flanking cen8. Supplementary Note "Extending rare-alignments into monomeric 'dead' layers" describes a similar analysis using blocks of monomeric alpha-satellites flanking cenX (that we refer to as paracenX) but arrives to a conclusion that the single-nucleotide mutation rate is low not only in HOR arrays but also in paracenX. Since the HPR Consortium is now completing assembly and validation of the first compete diploid-resolved human genome (HG002), it will soon be possible to check whether centromeres in human autosomes have vastly different mutation rates as compared to centromeres in sex chromosomes.

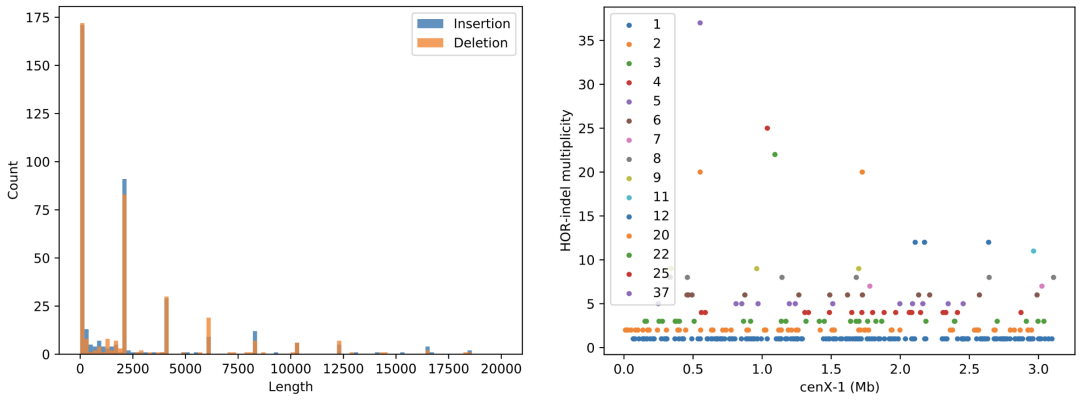


Figure 6.10. The histogram of lengths of insertion-runs and deletion-runs in the rare-alignment of $cenX_1$ and $cenX_2$ (left) and distribution of HOR-indels in this alignment along the entire length of $cenX_1$ (right). The peaks in the histogram correspond to either short (< 100 bp) runs or to likely insertions / deletions of a single or multiple canonical HOR units of cenX. The length of the canonical HOR in cenX is 2057 bp [34], while the peaks in the histogram correspond to 2057 bp (a single HOR unit), 4114 bp (two HOR units), 6171 bp (three HOR units), 8225 bp (approximately 4 HOR units), etc. The width of each bar is 500bp. Even though the histogram's x-axis is cut at 20 kb, there are long indels of lengths 22,628 bp (11 HOR units), 24,676 bp (12 HOR units), 41,133 bp (20 HOR units), 45,225 (22 HOR units), 51,393 bp (25 HOR units), and 76,098 bp (37 HOR units). (Right) Each color corresponds to an indel of a particular length, e.g., blue (orange, green) color corresponds to HOR-indel of multiplicity 1 (2, 3), etc.

6.3.13 TandemAligner reveals orthologous D genes in primate immunoglobulin loci

Comparative analysis of the immunoglobulin genes across multiple species is a prerequisite for evolutionary studies of the adaptive immune system. Below we focus on D genes in the immunoglobulin IGHD locus that play a key role in diversifying antibody repertoires and that are notoriously difficult to compare between species and predict in newly sequenced genomes [156]. Moreover, since D genes are located within ETRs, identifying pairs of orthologous D genes even between close species (such as primates) is challenging.

This difficulty is further compounded by the fact that assembly of the highly-repetitive immunoglobulin loci has been challenging [170], [171]. However, even though there were very few genomes available for comparative immunogenomics studies until recently, the situation has changed with the advent of contiguous long-read assemblies generated by the Vertebrate Genomes Project (VGP) [172]. Nevertheless, the existing methods for finding orthologous genes [173] are not well-suited for D genes since these methods are based on similarity search that often fails to detect similarities between highly diverged and short D genes (most are shorter than 50 bp) located within ETRs. As a result, the IGHD loci in hundreds vertebrate genomes recently assembled by the VGP consortium remain unannotated [156].

The IGHD locus in mammalian genomes includes a long tandem repeat [174]. For example, the human $IGHD_H$ locus contains a tandem repeat formed by four ˜10 kb long units while the orangutan $IGHD_O$ locus contains a tandem repeat formed by five ˜10 kb long units. Finding orthologs between these units (and thus analyzing evolution of D genes) is challenging.

The evolutionary correct alignment of $IGHD_H$ and $IGHD_O$ loci should match the first i units, followed by a unit-insertion in the orangutan genome (a unit deletion in the human genome), followed by matches of the remaining 4-i units (i is unknown). In difference from edlib, TandemAligner reveals the additional unit of the tandem repeat in the $IGHD_O$ locus and suggests that this additional unit emerged in the end of this locus ($i = 4$) in the common ancestor

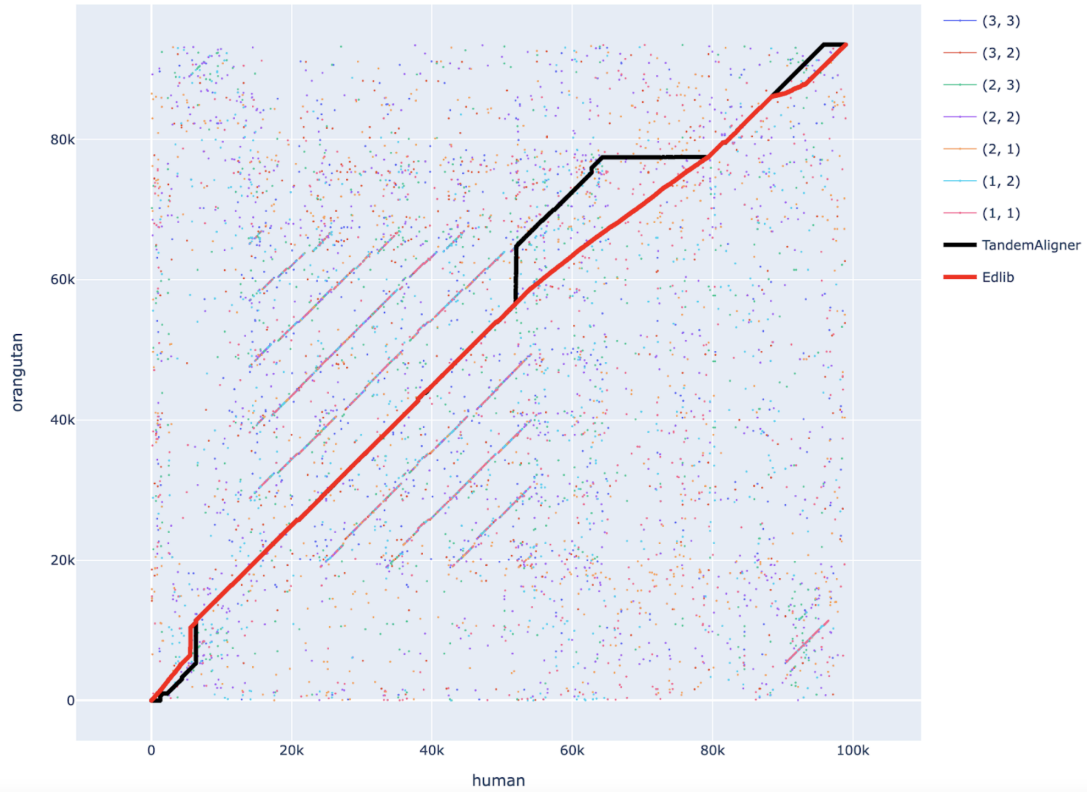


Figure 6.11. The rare-alignment (black) and standard alignment (red) of $IGHD_H$ and $IGHD_O$ loci and $DotPlot(IGHD_H, IGHD_O, Anchors_{n,m}(IGHD_H, IGHD_O))$ for (n,m) = (1,1), (1,2), (2,1), (2,2), (2, 3), (3, 2), (3, 3).

of humans and orangutans, thus establishing orthology between human and orangutan D genes (Figure 6.11).

6.3.14 Running time of TandemAligner

It takes TandemAligner 25 seconds to compute a rare-alignment and generate the CIGAR-string for $cenX_1$ and $cenX_2$. For comparison, it takes a state-of-the-art fast alignment algorithm edlib [165] 58 seconds on the same machine. Both tools use a single CPU thread.

6.4 Methods

6.4.1 From a k-mer-level alignment to a nucleotide-level alignment

A k-mer-level alignment between strings S^k and T^k induces a regular nucleotide-level alignment. Each rare k-mer in the alignment of strings S^k and T^k induces k matches (i, j), $(i+1, j+1), \ldots, (i+k-1, j+k-1)$ between individual positions in strings S and T. Since matches of overlapping k-mers may induce multiple matches of a single position in the string S (T), we need to remove some of the induced matches to ensure that a single position in one string is matched to a single position in another string. We say that an induced match (i, j) precedes induced matches $(i, j+\delta)$ and $(i+\delta, j)$ if $\delta > 0$ and iteratively remove all matches that are preceded by other matches. The remaining matches (along with resulting unmatched symbols that form indels) form a nucleotide-level alignment between strings S and T.

6.4.2 Algorithm for computing shortest rare substrings

Below we describe an algorithm for computing the array $rare_i(S,n)$ for all $1 \leq i \leq |S|$ given a parameter n in $O(|S|)$ time.

Let $S(i)$ be the suffix corresponding to the i-th element of the suffix array of the string S [164]. We define $LCP(k,l)$ as the Longest Common Prefix of $S(k)$ and $S(l)$. TandemAligner first computes the standard Longest Common Prefix (LCP) array $LCP(i-1,i)$ for consecutive elements $i-1$ and i in the suffix array and uses it to construct the array $LCP(k,k+n-1)$. It computes each element of this array in $O(1)$ time by iterating through the standard LCP array and using a deque with a minimum over the standard LCP array.

A segment $[k,l]$ is called n-rare if $LCP(k,l)$ is larger than both $LCP(k-1,l)$ and $LCP(k,l+1)$ and $l-k+1 = n$. Given an index i, we find the n-rare segment $[k,k+n-1]$ containing $S(i)$. Finally, we set $rare_i(S,n) = 1+\max(LCP(k-1,l), LCP(k,l+1))$. If no such rare segment exists, suffix $S(i)$ itself is frequent. In this case, we set $rare_i(S,n) = \infty$. Since we can compute $rare_i(S,n)$ for all i during a single iteration through the $LCP(k,k+n-1)$ array (and the

elements of this array can be efficiently computed from the standard LCP array), the complexity of calculating $rare_i(S,n)$ for all i is $O(|S|)$. Since the LCP array can be constructed from the suffix array in $O(|S|)$ time [175], the complexity of building $rare_i(S,n)$ for a string S is $O(|S|)$. In practice, we use a fast suffix array construction algorithm with complexity $O(|S|\log|S|)$ [176], [177] rather than $O(|S|)$ which is sufficient for practical purposes [178], [179], [180].

6.4.3 Standard alignment graph

The standard weighted directed acyclic graph that is used for the alignment of strings S and T presents a grid with $|S|+1$ rows and $|T|+1$ columns. For $1 \leq i \leq |S|$ and $1 \leq j \leq |T|$, the vertex with coordinate (i, j) is connected via a vertical, diagonal, and a horizontal edge to vertices $(i+1, j), (i+1, j+1), (i, j+1)$, respectively. Vertex $(|S|+1, j)$ in the last "row" of the grid is connected to the vertex $(|S|+1, j+1)$ via a horizontal edge for $1 \leq j \leq |T|$, and vertex $(i, |T|+1)$ in the last "column" of the grid is connected to $(i+1, |T|+1)$ via a vertical edge for $1 \leq i \leq |S|$. The diagonal edge connecting vertices (i, j) and $(i+1, j+1)$ corresponds to a *match* of characters $S[i]$ and $T[j]$ in case they are equal or to a *mismatch*, otherwise. The vertical edge connecting vertices (i, j) and $(i+1, j)$ corresponds to an *insertion* of character $S[i]$ between $T[j]$ and $T[j+1]$. Horizontal edge connecting vertices (i, j) and $(i, j+1)$ corresponds to a *deletion* of the character $T[j]$ between $S[i]$ and $S[i+1]$. The simplest scoring strategy in the standard alignment approach assigns the match (mismatch) weight to all diagonal edges that define matches (mismatches), and indel weight to all vertical and horizontal edges.

6.4.4 Recursive algorithm for constructing rare-alignment

Given strings S and T, TandemAligner constructs the anchor-based alignment *Alignment* using the set *Anchors(S, T, 1)* or, if this set is empty, for the minimal value of *Count* with non-empty set *Anchors(S, T, Count)*. We refer to a segment of *Alignment* consisting of a deletion-run immediately followed by an insertion-run (or vice versa) as an *indel-pair*. Each indel-pair corresponds to a pair of substrings s and t in S and T, respectively. TandemAligner recursively

constructs an alignment on strings s and t, and substitutes the indel-pair in *Alignment* by the resulting (small) alignment-path. The process terminates when there are either no indel-pairs left or the set of anchors constructed on substrings corresponding to each remaining indel-pair is empty for any choice of *Count*. In practice, we limit the value of *Count* by a large constant *MaxCount* (default value = 50) since matches of substrings of higher counts are likely to be spurious. However, this parameter is simply a practical heuristic that is not necessary for the theoretical description of the algorithm.

6.4.5 Refining rare-alignments

The recursive rare-alignment does not include any mismatches since it represents a single-nucleotide mismatch as an insertion (deletion) edge followed by a deletion (insertion) edge. Given a rare-alignment *Alignment*, TandemAligner searches for square indel-pairs that contain an equal number of insertions and deletions and represent likely runs of mismatches. For a given square indel-pair, we refer to this number as the *length* of this block. Recursive anchor-based rare-alignment of $cenX_1$ and $cenX_2$ contains 2051 (24, 4) square indel-pairs of length 1 (2, 3). For each square indel-pair in *Alignment*, TandemAligner substitutes it by a diagonal series of matches/mismatches in *Alignment*. Such a procedure applied to the recursive anchor-based rare-alignment of $cenX_1$ and $cenX_2$ introduces only 4 additional matches. The number of insertions (insertion-runs) reduced by 2111 (2079) to 1,154,459 (400). The number of deletions (deletion-runs) reduced to 1,123,756 (388). The number of introduced mismatches (mismatch-runs) is 2107 (2083).

6.5 Discussion

The ongoing effort to construct the human pangenome promises to change the way we analyze genomic variations and infer their associations with diseases. The construction of the pangenome graph is based on generating alignments between complete human genomes that are now being assembled by the HPR Consortium [150]. Although these alignments have already

been constructed for the vast majority of regions of the human genome, [151] it turned out that the standard alignment approach fails to adequately align the most repetitive (and biomedically important) regions such as centromeres. The emerging consensus is that aligning such regions requires a new framework for sequence comparison rather than simply tweaking parameters of the standard alignment approach. As stated in [151]: *"Our near-term goal is ... to refine the pangenome alignment methods (so that telomere-to-telomere alignment is possible capturing more complex regions of the genome)."*

We described a new parameter-free framework for sequence comparison that reveals the evolutionary history of highly-repetitive regions such as centromeres and immunoglobulin loci. Admittedly, since only two human centromeres have been assembled, carefully validated, and publicly released so far, our benchmarking remains limited to these centromeres and various simulated examples. However, even this limited benchmarking shed light on the evolution of human centromeres (and the extremely high rate of duplications/deletions of single/multiple HORs) that the standard alignment approach failed to uncover.

Although TandemAligner is already fast, we plan to further speed it up using the fast sparse dynamic programming algorithm by [168]. Although the rare-alignment framework represents the first step toward aligning centromeres and other ETRs, many questions about human ETRs remain unanswered. For example, TandemAligner reveals extremely high-rate of large insertion-runs in human centromeres but does not answer the question which region of the ancestral centromeres contributed to each of these insertion-runs. Zooming on Figure 6.8, right, reveals that many lines in this dot-plot aggregate into lines $j = i + t|HOR|$ that are parallel to the main diagonal ($|HOR|$ refers to the length of HOR in cenX and t is an integer). Since such aggregates are often triggered by HOR-insertions, their analysis should provide the initial clues for the common patterns of duplications in centromeres. Our next goal is to integrate analysis of duplications in the TandemAligner framework. Another bottleneck in applications of rare-alignments is that, in contrast to the analysis of statistical significance of standard alignments [181], [182], it remains unclear how to estimate the statistical significance of rare-alignments (see

Supplementary Notes “Summary of differences between standard alignment and rare-alignment”, “Evolution and alignment from probabilistic perspective” in [169]).

6.6 Data availability

Alignment of $cenX_1$ and $cenX_2$ generated by TandemAligner is located at Zenodo: https://zenodo.org/record/7058133.

6.7 Acknowledgements

We are grateful to Ivan Alexadrov, Anton Bankevich, Alexander Bzikadze, Tatiana Dvorkina, Evan Eichler, Glennis Logsdon, Olga Kunyavskaya, and Cynthia Wu for helpful discussions and suggestions.

Chapter 6, in full, has been submitted for a journal publication and appears as a *bioRxiv* pre-print: Bzikadze, A. V., & Pevzner, P. A. (2022). TandemAligner: a new parameter-free framework for fast sequence alignment. *bioRxiv*. The dissertation author is the primary developer of the TandemAligner algorithm and the first author of this paper.

Bibliography

[1] Chen-Shan Chin, Paul Peluso, Fritz J Sedlazeck, Maria Nattestad, Gregory T Concepcion, Alicia Clum, Christopher Dunn, Ronan O'Malley, Rosa Figueroa-Balderas, Abraham Morales-Cruz, et al. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature methods*, 13(12):1050–1054, 2016.

[2] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[3] Mikhail Kolmogorov, Jeffrey Yuan, Yu Lin, and Pavel A Pevzner. Assembly of long, error-prone reads using repeat graphs. *Nature biotechnology*, 37(5):540–546, 2019.

[4] Govinda M Kamath, Ilan Shomorony, Fei Xia, Thomas A Courtade, and N Tse David. Hinge: long-read assembly achieves optimal repeat resolution. *Genome research*, 27(5):747–756, 2017.

[5] Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R Miller, Nicholas H Bergman, and Adam M Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5):722–736, 2017.

[6] Sergej Nowoshilow, Siegfried Schloissnig, Ji-Feng Fei, Andreas Dahl, Andy WC Pang, Martin Pippel, Sylke Winkler, Alex R Hastie, George Young, Juliana G Roscito, et al. The axolotl genome and the evolution of key tissue formation regulators. *Nature*, 554(7690):50–55, 2018.

[7] Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *Nature methods*, 17(2):155–158, 2020.

[8] Mitchell R Vollger, Philip C Dishuck, Melanie Sorensen, AnneMarie E Welch, Vy Dang, Max L Dougherty, Tina A Graves-Lindsay, Richard K Wilson, Mark JP Chaisson, and Evan E Eichler. Long-read sequence and assembly of segmental duplications. *Nature methods*, 16(1):88–94, 2019.

[9] So I Nagaoka, Terry J Hassold, and Patricia A Hunt. Human aneuploidy: mechanisms and new insights into an age-old problem. *Nature Reviews Genetics*, 13(7):493–504, 2012.

[10] NI Enukashvily, R Donev, IS-R Waisertreiger, and OI Podgornaya. Human chromosome 1 satellite 3 dna is decondensed, demethylated and transcribed in senescent cells and in a431 epithelial carcinoma cells. *Cytogenetic and genome research*, 118(1):42–54, 2007.

[11] David T Ting, Doron Lipson, Suchismita Paul, Brian W Brannigan, Sara Akhavanfard, Erik J Coffman, Gianmarco Contino, Vikram Deshpande, A John Iafrate, Stan Letovsky, et al. Aberrant overexpression of satellite repeats in pancreatic and other epithelial cancers. *science*, 331(6017):593–596, 2011.

[12] Daniela Ferreira, Susana Meles, Ana Escudeiro, Ana Mendes-da Silva, Filomena Adega, and Raquel Chaves. Satellite non-coding rnas: the emerging players in cells, cellular pathways and cancer. *Chromosome Research*, 23(3):479–493, 2015.

[13] Simona Giunta and Hironori Funabiki. Integrity of the human centromere dna repeats is protected by cenp-a, cenp-c, and cenp-t. *Proceedings of the National Academy of Sciences*, 114(8):1928–1933, 2017.

[14] Elizabeth M Black and Simona Giunta. Repetitive fragile sites: centromere satellite dna as a source of genome instability in human diseases. *Genes*, 9(12):615, 2018.

[15] Ksenia Smurova and Peter De Wulf. Centromere and pericentromere transcription: roles and regulation... in sickness and in health. *Frontiers in Genetics*, 9:674, 2018.

[16] V Barra and D Fachinetti. The dark side of centromeres: types, causes and consequences of structural abnormalities implicating centromeric dna. *Nature Communications*, 9(1):1–17, 2018.

[17] Quan Zhu, Nien Hoong, Aaron Aslanian, Toshiro Hara, Christopher Benner, Sven Heinz, Karen H Miga, Eugene Ke, Sachin Verma, Jan Soroczynski, et al. Heterochromatin-encoded satellite rnas induce breast cancer. *Molecular cell*, 70(5):842–853, 2018.

[18] Karen H Miga. Centromeric satellite dnas: hidden sequence variation in the human population. *Genes*, 10(5):352, 2019.

[19] Mary G Schueler, Anne W Higgins, M Katharine Rudd, Karen Gustashaw, and Huntington F Willard. Genomic and genetic definition of a functional human centromere. *Science*, 294(5540):109–115, 2001.

[20] Can Alkan, Mario Ventura, Nicoletta Archidiacono, Mariano Rocchi, S Cenk Sahinalp, and Evan E Eichler. Organization and evolution of primate centromeric dna from whole-genome shotgun sequence data. *PLoS computational biology*, 3(9):e181, 2007.

[21] Valery A Shepelev, Alexander A Alexandrov, Yuri B Yurov, and Ivan A Alexandrov. The evolutionary origin of man can be traced in the layers of defunct ancestral alpha satellites flanking the active centromeres of human chromosomes. *PLoS genetics*, 5(9):e1000641, 2009.

[22] Daniël P Melters, Keith R Bradnam, Hugh A Young, Natalie Telis, Michael R May, J Graham Ruby, Robert Sebra, Paul Peluso, John Eid, David Rank, et al. Comparative analysis of tandem repeats from hundreds of species reveals unique insights into centromere evolution. *Genome biology*, 14(1):1–20, 2013.

[23] Sarah Sander Lower, Michael P McGurk, Andrew G Clark, and Daniel A Barbash. Satellite dna evolution: old ideas, new approaches. *Current opinion in genetics & development*, 49:70–78, 2018.

[24] A Cellamare, CR Catacchio, C Alkan, G Giannuzzi, F Antonacci, MF Cardone, G Della Valle, M Malig, M Rocchi, EE Eichler, et al. New insights into centromere organization and evolution from the white-cheeked gibbon and marmoset. *Molecular biology and evolution*, 26(8):1889–1900, 2009.

[25] Sasha A Langley, Karen H Miga, Gary H Karpen, and Charles H Langley. Haplotypes spanning centromeric regions reveal persistence of large blocks of archaic dna. *Elife*, 8:e42989, 2019.

[26] Miten Jain, Hugh E Olsen, Daniel J Turner, David Stoddart, Kira V Bulazel, Benedict Paten, David Haussler, Huntington F Willard, Mark Akeson, and Karen H Miga. Linear assembly of a human centromere on the y chromosome. *Nature biotechnology*, 36(4):321–323, 2018.

[27] Karen E Hayden, Erin D Strome, Stephanie L Merrett, Hye-Ran Lee, M Katharine Rudd, and Huntington F Willard. Sequences associated with centromere competency in the human genome. *Molecular and cellular biology*, 33(4):763–772, 2013.

[28] Volkan Sevim, Ali Bashir, Chen-Shan Chin, and Karen H Miga. Alpha-centauri: assessing novel centromeric repeat sequence variation with long read sequencing. *Bioinformatics*, 32(13):1921–1924, 2016.

[29] Dirk Schindelhauer and Tobias Schwarz. Evidence for a fast, intrachromosomal conversion mechanism from mapping of nucleotide variants within a homogeneous α-satellite dna array. *Genome research*, 12(12):1815–1826, 2002.

[30] Melanie M Mahtani and Huntington F Willard. Physical and genetic mapping of the human x chromosome centromere: repression of recombination. *Genome research*, 8(2):100–110, 1998.

[31] Karen H Miga, Yulia Newton, Miten Jain, Nicolas Altemose, Huntington F Willard, and W James Kent. Centromere reference models for human chromosomes x and y satellite arrays. *Genome research*, 24(4):697–707, 2014.

[32] Yu Peng, Henry CM Leung, Siu-Ming Yiu, and Francis YL Chin. Idba-ud: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, 28(11):1420–1428, 2012.

[33] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.

[34] Karen H Miga, Sergey Koren, Arang Rhie, Mitchell R Vollger, Ariel Gershman, Andrey Bzikadze, Shelise Brooks, Edmund Howe, David Porubsky, Glennis A Logsdon, et al. Telomere-to-telomere assembly of a complete human x chromosome. *Nature*, 585(7823):79–84, 2020.

[35] Andrey V Bzikadze and Pavel A Pevzner. Automated assembly of centromeres from ultra-long error-prone reads. *Nature Biotechnology*, 38(11):1309–1316, 2020.

[36] C Yang, RL Warren, J Chu, et al. Supporting data for "nanosim: nanopore sequence read simulator based on statistical characterization.". *GigaScience Database2017*, 2017.

[37] Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

[38] Alkes L Price, Eleazar Eskin, and Pavel A Pevzner. Whole-genome analysis of alu repeat elements reveals complex evolutionary history. *Genome research*, 14(11):2245–2252, 2004.

[39] E. Andrew Bennett, Heiko Keller, Ryan E. Mills, Steffen Schmidt, John V. Moran, Oliver Weichenrieder, and Scott E. Devine. Active alu retrotransposons in the human genome. *Genome Research*, 18(12):1875–1883, 2008.

[40] Uri Keich and Pavel A. Pevzner. Finding motifs in the twilight zone. In *Proceedings of the Sixth Annual International Conference on Computational Biology*, RECOMB '02, page 195–204, New York, NY, USA, 2002. Association for Computing Machinery.

[41] Alla Mikheenko, Andrey V Bzikadze, Alexey Gurevich, Karen H Miga, and Pavel A Pevzner. Tandemtools: mapping long reads and assessing/improving assembly quality in extra-long tandem repeats. *Bioinformatics*, 36(Supplement_1):i75–i83, 2020.

[42] LI Uralsky, VA Shepelev, AA Alexandrov, YB Yurov, Evgeny I Rogaev, and IA Alexandrov. Classification and monomer-by-monomer annotation dataset of suprachromosomal family 1 alpha satellite higher-order repeats in hg38 human genome assembly. *Data in brief*, 24:103708, 2019.

[43] Jorja G Henikoff, Jitendra Thakur, Sivakanthan Kasinathan, and Steven Henikoff. A unique chromatin complex occupies young α-satellite arrays of human centromeres. *Science advances*, 1(1):e1400234, 2015.

[44] John S Waye and Huntington F Willard. Chromosome-specific alpha satellite dna: nucleotide sequence analysis of the 2.0 kilobasepair repeat from the human x chromosome. *Nucleic acids research*, 13(8):2731–2743, 1985.

[45] Robert S Harris, Monika Cechova, and Kateryna D Makova. Noise-cancelling repeat finder: uncovering tandem repeats in error-prone long-read sequencing data. *Bioinformatics*, 35(22):4809–4811, 2019.

[46] Yu Lin, Jeffrey Yuan, Mikhail Kolmogorov, Max W Shen, Mark Chaisson, and Pavel A Pevzner. Assembly of long error-prone reads using de bruijn graphs. *Proceedings of the National Academy of Sciences*, 113(52):E8396–E8405, 2016.

[47] Tatiana Dvorkina, Andrey V Bzikadze, and Pavel A Pevzner. The string decomposition problem and its applications to centromere analysis and assembly. *Bioinformatics*, 36(Supplement_1):i93–i101, 2020.

[48] Sharon J Durfy and Huntington F Willard. Concerted evolution of primate alpha satellite dna: evidence for an ancestral sequence shared by gorilla and human x chromosome alpha satellite. *Journal of molecular biology*, 216(3):555–566, 1990.

[49] George P Smith. Evolution of repeated dna sequences by unequal crossover: Dna whose sequence is not maintained by selection will develop periodicities as a result of random crossover. *Science*, 191(4227):528–535, 1976.

[50] Albino Bacolla, Jacquelynn E Larson, Jack R Collins, Jian Li, Aleksandar Milosavljevic, Peter D Stenson, David N Cooper, and Robert D Wells. Abundance and length of simple repeats in vertebrate genomes are determined by their structural properties. *Genome research*, 18(10):1545–1553, 2008.

[51] Jorge J Yunis and Walid G Yasmineh. Heterochromatin, satellite dna, and cell function: Structural dna of eucaryotes may support and protect genes and aid in speciation. *Science*, 174(4015):1200–1209, 1971.

[52] Karen N McFarland, Jilin Liu, Ivette Landrian, Ronald Godiska, Savita Shanker, Fahong Yu, William G Farmerie, and Tetsuo Ashizawa. Smrt sequencing of long tandem nucleotide repeats in sca10 reveals unique insight of repeat expansion structure. *PloS one*, 10(8):e0135906, 2015.

[53] Janet HT Song, Craig B Lowe, and David M Kingsley. Characterization of a human-specific tandem repeat associated with bipolar disorder and schizophrenia. *The American Journal of Human Genetics*, 103(3):421–430, 2018.

[54] Melissa Gymrek, Thomas Willems, Audrey Guilmatre, Haoyang Zeng, Barak Markus, Stoyan Georgiev, Mark J Daly, Alkes L Price, Jonathan K Pritchard, Andrew J Sharp, et al. Abundant contribution of short tandem repeats to gene expression variation in humans. *Nature genetics*, 48(1):22–29, 2016.

[55] S Saini, I Mitra, N Mousavi, SF Fotsing, and M Gymrek. A reference haplotype panel for genome-wide imputation of short tandem repeats. nat. commun. 9: 4397, 2018.

[56] Thomas Willems, Melissa Gymrek, Gareth Highnam, David Mittelman, Yaniv Erlich, 1000 Genomes Project Consortium, et al. The landscape of human str variation. *Genome research*, 24(11):1894–1904, 2014.

[57] Laura Manuelidis and John C Wu. Homology between human and simian repeated dna. *Nature*, 276(5683):92–94, 1978.

[58] Huntington F Willard and John S Waye. Hierarchical order in chromosome-specific human alpha satellite dna. *Trends in Genetics*, 3:192–198, 1987.

[59] Huntington F Willard and John S Waye. Chromosome-specific subsets of human alpha satellite dna: analysis of sequence divergence within and between chromosomal subsets and evidence for an ancestral pentameric repeat. *Journal of mol. evolution*, 25(3):207–214, 1987.

[60] Dmitry Antipov, Anton Korobeynikov, Jeffrey S McLean, and Pavel A Pevzner. hybridspades: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, 32(7):1009–1015, 2016.

[61] Ryan R Wick, Louise M Judd, Claire L Gorrie, and Kathryn E Holt. Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLoS computational biology*, 13(6):e1005595, 2017.

[62] Aleksey V Zimin, Daniela Puiu, Ming-Cheng Luo, Tingting Zhu, Sergey Koren, Guillaume Marçais, James A Yorke, Jan Dvořák, and Steven L Salzberg. Hybrid assembly of the large and highly repetitive genome of aegilops tauschii, a progenitor of bread wheat, with the masurca mega-reads algorithm. *Genome research*, 27(5):787–792, 2017.

[63] Steven L Salzberg, Adam M Phillippy, Aleksey Zimin, Daniela Puiu, Tanja Magoc, Sergey Koren, Todd J Treangen, Michael C Schatz, Arthur L Delcher, Michael Roberts, et al. Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567, 2012.

[64] Alla Mikheenko, Andrey Prjibelski, Vladislav Saveliev, Dmitry Antipov, and Alexey Gurevich. Versatile genome assembly evaluation with quast-lg. *Bioinformatics*, 34(13):i142–i150, 2018.

[65] Alla Mikheenko, Vladislav Saveliev, and Alexey Gurevich. Metaquast: evaluation of metagenome assemblies. *Bioinformatics*, 32(7):1088–1090, 2016.

[66] Elena Bushmanova, Dmitry Antipov, Alla Lapidus, Vladimir Suvorov, and Andrey D Prjibelski. rnaquast: a quality assessment tool for de novo transcriptome assemblies. *Bioinformatics*, 32(14):2210–2212, 2016.

[67] Scott C Clark, Rob Egan, Peter I Frazier, and Zhong Wang. Ale: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome assemblies. *Bioinformatics*, 29(4):435–443, 2013.

[68] Mohammadreza Ghodsi, Christopher M Hill, Irina Astrovskaya, Henry Lin, Dan D Sommer, Sergey Koren, and Mihai Pop. De novo likelihood-based measures for comparing genome assemblies. *BMC research notes*, 6(1):1–18, 2013.

[69] Martin Hunt, Taisei Kikuchi, Mandy Sanders, Chris Newbold, Matthew Berriman, and Thomas D Otto. Reapr: a universal tool for genome assembly evaluation. *Genome biology*, 14(5):1–10, 2013.

[70] Felipe A Simão, Robert M Waterhouse, Panagiotis Ioannidis, Evgenia V Kriventseva, and Evgeny M Zdobnov. Busco: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, 31(19):3210–3212, 2015.

[71] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*, 2013.

[72] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[73] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009.

[74] Chirag Jain, Arang Rhie, Haowen Zhang, Claudia Chu, Brian P Walenz, Sergey Koren, and Adam M Phillippy. Weighted minimizer sampling improves long read mapping. *Bioinformatics*, 36(Supplement_1):i111–i118, 2020.

[75] Sergey Nurk, Brian P Walenz, Arang Rhie, Mitchell R Vollger, Glennis A Logsdon, Robert Grothe, Karen H Miga, Evan E Eichler, Adam M Phillippy, and Sergey Koren. Hicanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. *Genome research*, 30(9):1291–1305, 2020.

[76] Chen-Shan Chin, David H Alexander, Patrick Marks, Aaron A Klammer, James Drake, Cheryl Heiner, Alicia Clum, Alex Copeland, John Huddleston, Evan E Eichler, et al. Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nature methods*, 10(6):563–569, 2013.

[77] Nicholas J Loman, Joshua Quick, and Jared T Simpson. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature methods*, 12(8):733–735, 2015.

[78] Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 27(5):737–746, 2017.

[79] Thomas Haaf and Huntington F Willard. Orangutan α-satellite monomers are closely related to the human consensus sequence. *Mammalian genome*, 9(6):440–447, 1998.

[80] Sarah E Hall, Gregory Kettler, and Daphne Preuss. Centromere satellites from arabidopsis populations: maintenance of conserved and variable domains. *Genome research*, 13(2):195–205, 2003.

[81] MY Dennis, L Harshman, BJ Nelson, O Penn, S Cantsilieris, J Huddleston, F Antonacci, K Penewit, L Denman, A Raja, et al. The evolution and population diversity of human-specific segmental duplications. nat ecol evol, 1 (3), 69, 2017.

[82] US DOE Joint Genome Institute: Hawkins Trevor 4 Branscomb Elbert 4 Predki Paul 4 Richardson Paul 4 Wenning Sarah 4 Slezak Tom 4 Doggett Norman 4 Cheng Jan-Fang 4 Olsen Anne 4 Lucas Susan 4 Elkin Christopher 4 Uberbacher Edward 4 Frazier Marvin 4,

RIKEN Genomic Sciences Center: Sakaki Yoshiyuki 9 Fujiyama Asao 9 Hattori Masahira 9 Yada Tetsushi 9 Toyoda Atsushi 9 Itoh Takehiko 9 Kawagoe Chiharu 9 Watanabe Hidemi 9 Totoki Yasushi 9 Taylor Todd 9, Genoscope, CNRS UMR-8030: Weissenbach Jean 10 Heilig Roland 10 Saurin William 10 Artiguenave Francois 10 Brottier Philippe 10 Bruls Thomas 10 Pelletier Eric 10 Robert Catherine 10 Wincker Patrick 10, Institute of Molecular Biotechnology: Rosenthal André 12 Platzer Matthias 12 Nyakatura Gerald 12 Taudien Stefan 12 Rump Andreas 12 Department of Genome Analysis, GTC Sequencing Center: Smith Douglas R. 11 Doucette-Stamm Lynn 11 Rubenfield Marc 11 Weinstock Keith 11 Lee Hong Mei 11 Dubois JoAnn 11, Beijing Genomics Institute/Human Genome Center: Yang Huanming 13 Yu Jun 13 Wang Jian 13 Huang Guyang 14 Gu Jun 15, et al. Initial sequencing and analysis of the human genome. *nature*, 409(6822):860–921, 2001.

[83] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt, et al. The sequence of the human genome. *science*, 291(5507):1304–1351, 2001.

[84] Glennis A Logsdon, Mitchell R Vollger, PingHsun Hsieh, Yafei Mao, Mikhail A Liskovykh, Sergey Koren, Sergey Nurk, Ludovica Mercuri, Philip C Dishuck, Arang Rhie, et al. The structure, function and evolution of a complete human chromosome 8. *Nature*, 593(7857):101–107, 2021.

[85] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V Bzikadze, Alla Mikheenko, Mitchell R Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022.

[86] Mikhail Kolmogorov, Derek M Bickhart, Bahar Behsaz, Alexey Gurevich, Mikhail Rayko, Sung Bong Shin, Kristen Kuhn, Jeffrey Yuan, Evgeny Polevikov, Timothy PL Smith, et al. metaflye: scalable long-read metagenome assembly using repeat graphs. *Nature Methods*, 17(11):1103–1110, 2020.

[87] Haoyu Cheng, Gregory T Concepcion, Xiaowen Feng, Haowen Zhang, and Heng Li. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature methods*, 18(2):170–175, 2021.

[88] Kishwar Shafin, Trevor Pesout, Ryan Lorig-Roach, Marina Haukness, Hugh E Olsen, Colleen Bosworth, Joel Armstrong, Kristof Tigyi, Nicholas Maurer, Sergey Koren, et al. Nanopore sequencing and the shasta toolkit enable efficient de novo assembly of eleven human genomes. *Nature biotechnology*, 38(9):1044–1053, 2020.

[89] Aaron M Wenger, Paul Peluso, William J Rowell, Pi-Chuan Chang, Richard J Hall, Gregory T Concepcion, Jana Ebler, Arkarachai Fungtammasan, Alexey Kolesnikov, Nathan D Olson, et al. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature biotechnology*, 37(10):1155–1162, 2019.

[90] Derek M Bickhart, Mikhail Kolmogorov, Elizabeth Tseng, Daniel M Portik, Anton Korobeynikov, Ivan Tolstoganov, Gherman Uritskiy, Ivan Liachko, Shawn T Sullivan,

Sung Bong Shin, et al. Generating lineage-resolved, complete metagenome-assembled genomes from complex microbial communities. *Nature biotechnology*, 40(5):711–719, 2022.

[91] Andrey V Bzikadze, Alla Mikheenko, and Pavel P Pevzner. Fast and accurate mapping of long reads to complete genome assemblies with VerityMap. *Submitted*, 2022.

[92] Ann M Mc Cartney, Kishwar Shafin, Michael Alonge, Andrey V Bzikadze, Giulio Formenti, Arkarachai Fungtammasan, Kerstin Howe, Chirag Jain, Sergey Koren, Glennis A Logsdon, et al. Chasing perfection: validation and polishing strategies for telomere-to-telomere genome assemblies. *Nature Methods*, pages 1–9, 2022.

[93] Daniel Mapleson, Gonzalo Garcia Accinelli, George Kettleborough, Jonathan Wright, and Bernardo J Clavijo. Kat: a k-mer analysis toolkit to quality control ngs datasets and genome assemblies. *Bioinformatics*, 33(4):574–576, 2017.

[94] Arang Rhie, Brian P Walenz, Sergey Koren, and Adam M Phillippy. Merqury: reference-free quality, completeness, and phasing assessment for genome assemblies. *Genome biology*, 21(1):1–27, 2020.

[95] B Langmead, C Trapnell, M Pop, and SL Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. genome biol10: R25, 2009.

[96] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357–359, 2012.

[97] Chirag Jain, Arang Rhie, Nancy F Hansen, Sergey Koren, and Adam M Phillippy. Long-read mapping to repetitive reference sequences using winnowmap2. *Nature Methods*, pages 1–6, 2022.

[98] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011.

[99] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. Kmc 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017.

[100] Israt Nisa, Prashant Pandey, Marquita Ellis, Leonid Oliker, Aydın Buluç, and Katherine Yelick. Distributed-memory k-mer counting on gpus. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 527–536. IEEE, 2021.

[101] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[102] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[103] Heng Li. New strategies to improve minimap2 alignment accuracy. *Bioinformatics*, 37(23):4572–4574, 2021.

[104] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. Pbsim2: a simulator for long-read sequencers with a novel generative model of quality scores. *Bioinformatics*, 37(5):589–595, 2021.

[105] James T Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S Lander, Gad Getz, and Jill P Mesirov. Integrative genomics viewer. *Nature biotechnology*, 29(1):24–26, 2011.

[106] Mantas Sereika, Rasmus Hansen Kirkegaard, Søren Michael Karst, Thomas Yssing Michaelsen, Emil Aarre Sørensen, Rasmus Dam Wollenberg, and Mads Albertsen. Oxford nanopore r10. 4 long-read sequencing enables near-perfect bacterial genomes from pure cultures and metagenomes without short-read or reference polishing. *Biorxiv*, 2021.

[107] Anton Bankevich, Andrey V Bzikadze, Mikhail Kolmogorov, Dmitry Antipov, and Pavel A Pevzner. Multiplex de bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature biotechnology*, pages 1–7, 2022.

[108] Dan Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.

[109] Richard M Karp and Michael O Rabin. Efficient randomized pattern-matching algorithms. *IBM journal of research and development*, 31(2):249–260, 1987.

[110] Nicolas Altemose, Glennis A Logsdon, Andrey V Bzikadze, Pragya Sidhwani, Sasha A Langley, Gina V Caldas, Savannah J Hoyt, Lev Uralsky, Fedor D Ryabov, Colin J Shew, et al. Complete genomic and epigenetic maps of human centromeres. *Science*, 376(6588):eabl4178, 2022.

[111] William R. Rice. A game of thrones at human centromere i. multifarious structure necessitates a new molecular/evolutionary model. *bioRxiv*.

[112] Jitendra Thakur, Jenika Packiaraj, and Steven Henikoff. Sequence, chromatin and evolution of satellite dna. *International Journal of Molecular Sciences*, 22(9):4309, 2021.

[113] Harmit S Malik and Steven Henikoff. Major evolutionary transitions in centromere complexity. *Cell*, 138(6):1067–1082, 2009.

[114] Tatiana Dvorkina, Olga Kunyavskaya, Andrey V Bzikadze, Ivan Alexandrov, and Pavel A Pevzner. Centromerearchitect: inference and analysis of the architecture of centromeres. *Bioinformatics*, 37(Supplement_1):i196–i204, 2021.

[115] John S Waye and Huntington F Willard. Nucleotide sequence heterogeneity of alpha satellite repetitive dna: a survey of alphoid sequences from different human chromosomes. *Nucleic acids research*, 15(18):7549–7569, 1987.

[116] Ivan Alexandrov, Alexei Kazakov, Irina Tumeneva, Valery Shepelev, and Yuri Yurov. Alpha-satellite dna of primates: old and new families. *Chromosoma*, 110(4):253–266, 2001.

[117] Shannon M McNulty and Beth A Sullivan. Alpha satellite dna biology: finding function in the recesses of the genome. *Chromosome Research*, 26(3):115–138, 2018.

[118] Olga Kunyavskaya, Tatiana Dvorkina, Andrey V Bzikadze, Ivan A Alexandrov, and Pavel A Pevzner. Automated annotation of human centromeres with hormon. *Genome Research*, 2022.

[119] Hajime Suzuki and Masahiro Kasahara. Introducing difference recurrence relations for faster semi-global alignment of long sequences. *BMC bioinformatics*, 19(1):33–47, 2018.

[120] Vladimir Paar, Nenad Pavin, Marija Rosandić, Matko Glunčić, Ivan Basar, Robert Pezer, and Sonja Durajlija Žinić. Colorhor—novel graphical algorithm for fast scan of alpha satellite higher-order repeats and hor annotation for genbank sequence of human genome. *Bioinformatics*, 21(7):846–852, 2005.

[121] VA Shepelev, LI Uralsky, AA Alexandrov, YB Yurov, Evgeny I Rogaev, and IA Alexandrov. Annotation of suprachromosomal families reveals uncommon types of alpha satellite organization in pericentromeric regions of hg38 human genome assembly. *Genomics data*, 5:139–146, 2015.

[122] Karen H Miga and Ivan A Alexandrov. Variation and evolution of human centromeres: A field guide and perspective. *Annual Review of Genetics*, 55:583–602, 2021.

[123] Vladimir Paar, Ines Vlahović, Marija Rosandić, and Matko Glunčić. Global repeat map (grm): Advantageous method for discovery of largest higher-order repeats (hors) in neuroblastoma breakpoint family (nbpf) genes, in hornerin exon and in chromosome 21 centromere. In *Satellite DNAs in Physiology and Evolution*, pages 203–234. Springer, 2021.

[124] Lingzhan Xue, Yu Gao, Meiying Wu, Tian Tian, Haiping Fan, Yongji Huang, Zhen Huang, Dapeng Li, and Luohao Xu. Telomere-to-telomere assembly of a fish y chromosome reveals the origin of a young sex chromosome pair. *Genome biology*, 22(1):1–20, 2021.

[125] Jin Jun, Ion I Mandoiu, and Craig E Nelson. Identification of mammalian orthologs using local synteny. *BMC genomics*, 10(1):1–13, 2009.

[126] Phillip EC Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991, 2011.

[127] Kathleen Carine, Alain Jacquemin-Sablon, Elysa Waltzer, Jim Mascarello, and Immo E Scheffler. Molecular characterization of human minichromosomes with centromere from chromosome 1 in human-hamster hybrid cells. *Somatic cell and molecular genetics*, 15(5):445–460, 1989.

[128] N Mestrović, Miroslav Plohl, Brankica Mravinac, and D Ugarković. Evolution of satellite dnas from the genus palorus–experimental evidence for the" library" hypothesis. *Molecular biology and evolution*, 15(8):1062–1068, 1998.

[129] Steven Henikoff, Kami Ahmad, and Harmit S. Malik. The centromere paradox: Stable inheritance with rapidly evolving dna. *Science*, 293(5532):1098–1102, 2001.

[130] Yuta Suzuki, Eugene W. Myers, and Shinichi Morishita. Rapid and ongoing evolution of repetitive sequence structures in human centromeres. *Science Advances*, 6(50):eabd9230, 2020.

[131] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. 1988.

[132] Danny E Miller, Arvis Sulovari, Tianyun Wang, Hailey Loucks, Kendra Hoekzema, Katherine M Munson, Alexandra P Lewis, Edith P Almanza Fuerte, Catherine R Paschal, Tom Walsh, et al. Targeted long-read sequencing identifies missing disease-causing variation. *The American Journal of Human Genetics*, 108(8):1436–1449, 2021.

[133] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.

[134] Eugene W Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85, 2005.

[135] Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, 31(10):1674–1676, 2015.

[136] Pavel A Pevzner, Haixu Tang, and Glenn Tesler. De novo repeat classification and fragment assembly. In *Proceedings of the eighth annual international conference on Research in computational molecular biology*, pages 213–222, 2004.

[137] Chengxi Ye, Zhanshan Sam Ma, Charles H Cannon, Mihai Pop, and Douglas W Yu. Exploiting sparseness in de novo genome assembly. In *BMC bioinformatics*, volume 13, pages 1–8. BioMed Central, 2012.

[138] Mikko Rautiainen and Tobias Marschall. Mbg: Minimizer-based sparse de bruijn graph construction. *Bioinformatics*, 37(16):2476–2478, 2021.

[139] Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004.

[140] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology*, 8(1):1–9, 2013.

[141] Ilia Minkin, Son Pham, and Paul Medvedev. Twopaco: an efficient algorithm to build the compacted de bruijn graph from many complete genomes. *Bioinformatics*, 33(24):4024–4032, 2017.

[142] Alla Mikheenko, Gleb Valin, Andrey Prjibelski, Vladislav Saveliev, and Alexey Gurevich. Icarus: visualizer for de novo assembly evaluation. *Bioinformatics*, 32(21):3321–3323, 2016.

[143] Shilpa Garg, Arkarachai Fungtammasan, Andrew Carroll, Mike Chou, Anthony Schmitt, Xiang Zhou, Stephen Mac, Paul Peluso, Emily Hatas, Jay Ghurye, et al. Chromosome-scale, haplotype-resolved assembly of human genomes. *Nature biotechnology*, 39(3):309–312, 2021.

[144] Ting Hon, Kristin Mars, Greg Young, Yu-Chih Tsai, Joseph W Karalius, Jane M Landolin, Nicholas Maurer, David Kudrna, Michael A Hardigan, Cynthia C Steiner, et al. Highly accurate long-read hifi sequencing data for five complex genomes. *Scientific data*, 7(1):1–11, 2020.

[145] Eric S Tvedte, Mark Gasser, Benjamin C Sparklin, Jane Michalski, Carl E Hjelmen, J Spencer Johnston, Xuechu Zhao, Robin Bromley, Luke J Tallon, Lisa Sadzewicz, et al. Comparison of long-read sequencing technologies in interrogating bacteria and fly genomes. *G3*, 11(6):jkab083, 2021.

[146] Ramana M Idury and Michael S Waterman. A new algorithm for dna sequence assembly. *Journal of computational biology*, 2(2):291–306, 1995.

[147] Pavel A Pevzner, Haixu Tang, and Michael S Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the national academy of sciences*, 98(17):9748–9753, 2001.

[148] Sergey Koren, Michael C Schatz, Brian P Walenz, Jeffrey Martin, Jason T Howard, Ganeshkumar Ganapathy, Zhong Wang, David A Rasko, W Richard McCombie, Erich D Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature biotechnology*, 30(7):693–700, 2012.

[149] Richard J Roberts, Mauricio O Carneiro, and Michael C Schatz. The advantages of smrt sequencing. *Genome biology*, 14(6):1–4, 2013.

[150] Ting Wang, Lucinda Antonacci-Fulton, Kerstin Howe, Heather A Lawson, Julian K Lucas, Adam M Phillippy, Alice B Popejoy, Mobin Asri, Caryn Carson, Mark JP Chaisson, et al. The human pangenome project: a global resource to map genomic diversity. *Nature*, 604(7906):437–446, 2022.

[151] Wen-Wei Liao, Mobin Asri, Jana Ebler, Daniel Doerr, Marina Haukness, Glenn Hickey, Shuangjia Lu, Julian K Lucas, Jean Monlong, Haley J Abel, et al. A draft human pangenome reference. *bioRxiv*, 2022.

[152] Mitchell R Vollger, Xavi Guitart, Philip C Dishuck, Ludovica Mercuri, William T Harvey, Ariel Gershman, Mark Diekhans, Arvis Sulovari, Katherine M Munson, Alexandra P Lewis, et al. Segmental duplications and their variation in a complete human genome. *Science*, 376(6588):eabj6965, 2022.

[153] Savannah J Hoyt, Jessica M Storer, Gabrielle A Hartley, Patrick GS Grady, Ariel Gershman, Leonardo G de Lima, Charles Limouse, Reza Halabian, Luke Wojenski, Matias Rodriguez, et al. From telomere to telomere: The transcriptional and epigenetic state of human repeat elements. *Science*, 376(6588):eabk3112, 2022.

[154] Mehrdad Bakhtiari, Jonghun Park, Yuan-Chun Ding, Sharona Shleizer-Burko, Susan L Neuhausen, Bjarni V Halldórsson, Kári Stefánsson, Melissa Gymrek, and Vineet Bafna. Variable number tandem repeats mediate the expression of proximal genes. *Nature communications*, 12(1):1–12, 2021.

[155] Jonghun Park, Mehrdad Bakhtiari, Bernt Popp, Michael Wiesener, and Vineet Bafna. Detecting tandem repeat variants in coding regions using code-advntr. *Iscience*, 25(8):104785, 2022.

[156] Vikram Sirupurapu, Yana Safonova, and Pavel A Pevzner. Gene prediction in the immunoglobulin loci. *Genome Research*, 32(6):1152–1169, 2022.

[157] Geraldine A Van der Auwera and Brian D O'Connor. *Genomics in the cloud: using Docker, GATK, and WDL in Terra*. O'Reilly Media, 2020.

[158] Mehrdad Bakhtiari, Sharona Shleizer-Burko, Melissa Gymrek, Vikas Bansal, and Vineet Bafna. Targeted genotyping of variable number tandem repeats with advntr. *Genome research*, 28(11):1709–1719, 2018.

[159] Nima Mousavi, Sharona Shleizer-Burko, Richard Yanicky, and Melissa Gymrek. Profiling the genome-wide landscape of tandem repeat expansions. *Nucleic acids research*, 47(15):e90–e90, 2019.

[160] Mark JP Chaisson, Ashley D Sanders, Xuefang Zhao, Ankit Malhotra, David Porubsky, Tobias Rausch, Eugene J Gardner, Oscar L Rodriguez, Li Guo, Ryan L Collins, et al. Multi-platform discovery of haplotype-resolved structural variation in human genomes. *Nature communications*, 10(1):1–16, 2019.

[161] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[162] M Katharine Rudd, Gregory A Wray, and Huntington F Willard. The evolutionary dynamics of α-satellite. *Genome research*, 16(1):88–96, 2006.

[163] Mark D Pertile, Alison N Graham, KH Andy Choo, and Paul Kalitsis. Rapid evolution of mouse y centromere repeat dna belies recent sequence stability. *Genome research*, 19(12):2202–2213, 2009.

[164] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[165] Martin Šošić and Mile Šikić. Edlib: a c/c++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 33(9):1394–1395, 2017.

[166] Adrian J Gibbs and George A McIntyre. The diagram, a method for comparing sequences: Its use with amino acid and nucleotide sequences. *European journal of biochemistry*, 16(1):1–11, 1970.

[167] William R Pearson and David J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[168] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F Italiano. Sparse dynamic programming i: linear cost functions. *Journal of the ACM (JACM)*, 39(3):519–545, 1992.

[169] Andrey V Bzikadze and Pavel A Pevzner. Tandemaligner: a new parameter-free framework for fast sequence alignment. *bioRxiv*, 2022.

[170] Corey T Watson and Felix Breden. The immunoglobulin heavy chain locus: genetic variation, missing data, and implications for human disease. *Genes & Immunity*, 13(5):363–373, 2012.

[171] Oscar L Rodriguez, William S Gibson, Tom Parks, Matthew Emery, James Powell, Maya Strahl, Gintaras Deikus, Kathryn Auckland, Evan E Eichler, Wayne A Marasco, et al. A novel framework for characterizing genomic haplotype diversity in the human immunoglobulin heavy chain locus. *Frontiers in immunology*, 11:2136, 2020.

[172] Arang Rhie, Shane A McCarthy, Olivier Fedrigo, Joana Damas, Giulio Formenti, Sergey Koren, Marcela Uliano-Silva, William Chow, Arkarachai Fungtammasan, Juwan Kim, et al. Towards complete and error-free genome assemblies of all vertebrate species. *Nature*, 592(7856):737–746, 2021.

[173] Eugene V Koonin. Comparative genomics, minimal gene-sets and the last universal common ancestor. *Nature Reviews Microbiology*, 1(2):127–136, 2003.

[174] Yana Safonova and Pavel A Pevzner. V (dd) j recombination is an important and evolutionarily conserved mechanism for generating antibodies with unusually long cdr3s. *Genome research*, 30(11):1547–1558, 2020.

[175] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Annual Symposium on Combinatorial Pattern Matching*, pages 181–192. Springer, 2001.

[176] N Jesper Larsson and Kunihiko Sadakane. *Faster suffix sorting*. Citeseer, 1999.

[177] Stefan Burkhardt and Juha Kärkkäinen. Fast lightweight suffix array construction and checking. In *Annual Symposium on Combinatorial Pattern Matching*, pages 55–69. Springer, 2003.

[178] Juha Kärkkäinen and Peter Sanders. Simple linear work suffix array construction. In *International colloquium on automata, languages, and programming*, pages 943–955. Springer, 2003.

[179] Dong Kyue Kim, Jeong Seop Sim, Heejin Park, and Kunsoo Park. Linear-time construction of suffix arrays. In *Annual Symposium on Combinatorial Pattern Matching*, pages 186–199. Springer, 2003.

[180] Pang Ko and Srinivas Aluru. Space efficient linear time construction of suffix arrays. In *Annual Symposium on Combinatorial Pattern Matching*, pages 200–210. Springer, 2003.

[181] Richard Arratia and Michael S Waterman. A phase transition for the score in matching random sequences allowing deletions. *The Annals of Applied Probability*, pages 200–225, 1994.

[182] Michael S Waterman and Martin Vingron. Sequence comparison significance and poisson approximation. *Statistical Science*, pages 367–381, 1994.