Scalable methods for improving genome assemblies

by

Vladimir Nikolić

B.Sc., The University of Belgrade, 2018

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Bioinformatics)

The University of British Columbia
(Vancouver)

April 2021

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Scalable methods for improving genome assemblies

submitted by **Vladimir Nikolić** in partial fulfillment of the requirements for the degree of **Master of Science** in **Bioinformatics**.

Examining Committee:

Dr. Inanc Birol, Medical Genetics, UBC
Supervisor

Dr. Andrew Roth, Computer Science, UBC
Supervisory Committee Member

Dr. Faraz Hach, Urologic Sciences, UBC
Supervisory Committee Member

Abstract

De novo genome assembly is cornerstone to modern genomics studies. It is also a useful method for studying genomes with high variation, such as cancer genomes, as it is not biased by a reference. *De novo* short-read assemblers commonly use de Bruijn graphs, where nodes are sequences of equal length k, also known as k-mers. Edges in this graph are established between nodes that overlap by $k-1$ bases, followed by merging nodes along unambiguous walks in the graph. The selection of k is influenced by a few factors, and its fine tuning results in a trade-off between graph connectivity and sequence contiguity. Ideally, multiple k sizes should be used, so lower values can provide good connectivity in lesser covered regions and higher values can increase contiguity in well-covered regions. However, this approach has only been explored with small genomes, without addressing scalability issues with larger ones. Here we present RResolver, a scalable algorithm that takes a short-read de Bruijn graph assembly with a starting k as input and uses a k value closer to that of the read length to resolve repeats. RResolver builds a Bloom filter of sequencing reads which it uses to evaluate the assembly graph path support at branching points and removes the paths with insufficient support. RResolver runs efficiently, taking 3% of a typical ABySS human assembly pipeline run time on average with 48 threads and 40GB memory. Compared to a baseline assembly, RResolver improves scaffold contiguity (NGA50) by up to 16% and reduces misassemblies by up to 7%. RResolver adds a missing component to scalable de Bruijn graph genome assembly. By improving the initial and fundamental graph traversal outcome, all downstream ABySS algorithms greatly benefit by working with a more accurate and less complex representation of the genome.

Lay Summary

Current technologies that read DNA from a sample do so by providing only fragments of original chromosome sequences. Moreover, depending on the technology used, these fragments have a varying number of errors in them. For these reasons, there is a need for algorithms that will assemble these fragments together to, ideally, form a sequence for every chromosome of the sampled organism. However, this is far from an easy task and a number of algorithms is used together in order to perform as high quality assembly as possible within reasonable time. This work includes a new approach that helps improve the quality of these assemblies in the contiguity of sequences, number of correctly joined fragments, and the number of genes recovered. The tested genomes belong to a variety of species with very different genomes in terms of total length and complexity.

Preface

This work has been completed under the supervision of Dr. Inanc Birol at Canada's Michael Smith Genome Sciences Centre at BC Cancer. The research idea was conceived by Dr. Birol and I have designed, implemented, and benchmarked the algorithms presented. René L. Warren, Justin Chu, Johnathan Wong, Lauren Coombe, and Ka Ming Nip from the Bioinformatics Technology Lab at the Genome Sciences Centre have contributed to the design and troubleshooting of the algorithms. Lauren Coombe has also helped with dataset benchmarking. Shaun Jackman, a former Ph.D. student at the Genome Sciences Centre, has helped with the discussions and clarifications around the ABySS assembler design and concepts as well as for a number of suggestions on the algorithm design presented in the thesis. Kirstin Brown, from Knowledge Translation & Communication team at Genome Sciences Centre has helped proofread the research chapter. The colorblind-friendly color palette used in the thesis figures was provided by Martin Krzywinski, a staff scientist at the Genome Sciences Centre. This thesis uses the ubcdiss LaTeX template provided by Brian de Alwis. The content of this thesis is solely the responsibility of the author, and does not necessarily represent the official views of the funding organizations.

This work is planned for publication.

Table of Contents

List of Tables

List of Figures

Glossary

BAC Bacterial Artificial Chromosome

BUSCO Benchmarking sets of Universal Single-Copy Orthologs

CEG Core Eukaryotic Gene

CNV Copy Number Variation

DBG de Bruijn Graph

FPR False Positive Rate

OLC Overlap-Layout-Consensus

ORF Open Reading Frame

POG Personalized OncoGenomics

SNP Single Nucleotide Polymorphism

TE Transposable Element

TR Tandem Repeat

Acknowledgments

I would like to thank my supervisor Dr. Inanc Birol for entrusting me with this research after I expressed interest in the field. For allowing me to explore different ideas and concepts and satisfy my curiosity throughout my work. And more than being a supervisor, our lab socials have made my time during this period much more enjoyable.

Thank you to my supervisory committee members, Dr. Andrew Roth and Dr. Faraz Hach, for all the feedback and guidance.

Thank you to all the cool people in the Bioinformatics Technology Lab from whom I have learned immensely. Shaun Jackman, Lauren Coombe, and René Warren have been especially helpful.

Thank you to Jordan Sicherman and Mariia Radaeva for being great buddies during this time. Your friendship and all the laughter has helped me go through the frustrating moments of research.

This work has been supported by Genome BC and Genome Canada [281ANV]; the National Institutes of Health [2R01HG007182-04A1]; and the Natural Sciences and Engineering Research Council of Canada. The content of this thesis is solely the responsibility of the author, and does not necessarily represent the official views of the funding organizations.

Dedication

To my parents, Miroslav and Marina.

Chapter 1

Introduction

1.1 History of *de novo* genome assembly

De novo genome assembly problem deals with reconstruction of all chromosome and mitochondrial DNA sequences of a given organism without any reference genome sequence. The initial approaches in the 80s relied on the shotgun method and Sanger sequencing [56] wherein the target sequence had to be assembled from many randomly sampled subsequences, i.e., reads. A shotgun method relies on oversampling the target genome so that a number of reads cover every, or almost every, part of the sequence. This ensures sufficient overlap between reads and enough base pair redundancy to take into account erroneous base calls. The first shotgun assembly [33] reconstructed the DNA sequence of bacteriophage λ [57] using reads of approximately 200 base pairs (bp) and manual inspections of the pieces with the help of an algorithm to guide overlaps.

Sanger sequencing paved way for publication of a series of genomes of model organisms beginning in the 90s. From the *C. elegans* genome in 1995 through 1998 [70] followed by the first complete eukaryote genome sequence of *S. cerevisiae* in 1996 [26] and *D. melanogaster* in 2000 [6]. In 1990, The Human Genome Project was launched and in competition with J. C. Venter produced a first draft of human genome in 2000 and a complete genome in 2003 [1, 64].

In 1997, paired-end sequencing was introduced [71], complementing information from an

individual read with approximate distance from another read. With this approach, reads are taken from ends of a DNA fragment whose size ranges from less than reads' sizes combined, meaning they overlap, to multiple kilobases. Smaller fragments allow for inferring the sequence in-between reads in order to produce longer pseudo-reads [63]. Alternatively, they can be used in order to pair two assembled sequences together to form a more contiguous sequence called **contig**. For larger fragments in the kilobase range, the pairing can be done on contigs in order to create **scaffolds**, i.e., paths of contigs with gaps of known distance in-between.

At the beginning of 80s, fairly basic programs were used to find overlaps between sequences [62]. Soon, however, new conceptual approaches were being introduced and algorithms formalized. Two prominent approaches arose, Overlap-Layout-Consensus (OLC) and de Bruijn Graph (DBG), which are, to this day, being widely used. In general, the former finds overlaps between reads, taking into account sequencing errors, and arranges then together to form an assembly, using consensus when bases conflict. The latter works at the level of k-mers, i.e., substrings of reads of equal length, and relies on redundancy to work with correct k-mers, joining them together when they overlap almost completely. OLC was the strategy of the early methods [33] [51] [34] and was later modified in the string graph algorithms, introduced in 1995 [46] and formalized in 2005 [48] by E. W. Myers. String graphs have been later successfully utilized by the SGA assembler [60], published in 2011. Celera assembler has used the OLC approach successfully in the assembly of *D. melanogaster* and *H. sapiens* [47] [64]. DBG as an assembly concept was also introduced in 1995 [29] and has further explored by *Pevzner et al* in their Euler assembler [53].

1.2 Utility of sequence assembly

De novo genome assembly has a wide range of applications, such as gene annotation [66], phylogenetic inference [25], identifying polymorphisms [18] and structural variations [31]. *De novo* assembly specifically is used when either no reference genome is available, or to avoid the biases that may be introduced by using one. For example, a reference genome will not be

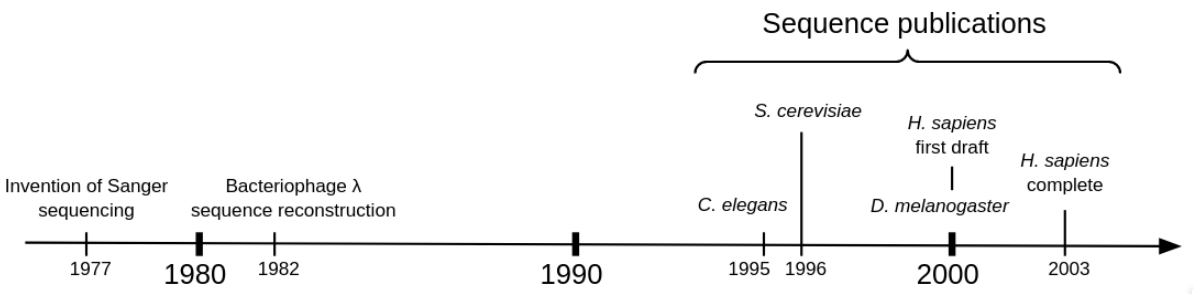

Figure 1.1: Timeline with early sequencing technology and landmark sequence publications.

available when sequencing and annotating the genome of a species for the first time. Another example is cancer studies, in which structural differences between the sequenced tumor and the reference are important.

Clinical applications such as Personalized OncoGenomics (POG) program [38] is certainly one way in which *de novo* assembly is highly beneficial, both genomic and transcriptomic. Structural variants inferred from the assemblies of healthy and tumor tissue can point to aberrations in the tumor genome which may guide therapeutic decisions. Translocations, inversions, and copy number variation are all known to be relevant for oncogenesis. Ideally, multiple sources of evidence would point to the same abnormality providing higher confidence to the clinicians. e.g., genomic and transcriptomic assembly might indicate Copy Number Variation (CNV) and high gene expression. Additionally, as studies and programs like POG collect data on the efficacy of treatment decisions and genome and transcriptome sequences they were based on, the utility of assemblies will increase.

When it comes to gene annotation of species without a reference, *de novo* assembly is indispensable. Often, hybrid assembly of short and long-reads provides the highest quality output [49]. There are multiple ways of assembling hybrid data. e.g., a short-read assembly followed by long read repeat resolution and scaffolding [49]. Alternatively, the relative high quality of short-reads compared to long can be used to correct the errors in long-reads, which are then assembled, as is done in Canu [37] and Falcon [15] assemblers. A high quality assembly obtained can then be used to shine light on gene function, annotate protein coding genes, and

evolution of gene families. Gene annotation today is far from perfect, however, as it suffers from inaccuracies caused by fragmented and erroneous draft assemblies. The process requires knowing which of the possible six Open Reading Frames (ORFs) translates a protein, but attempts at figuring that out are hampered by errors. Moreover, for eukaryotes in particular, gene annotation poses a difficult problem. Protein-coding genes are sparse, covering only a small percentage of the genome, e.g., <%3 for *H. sapiens* [5] and can have many kilobases long introns. Generally, an automated pipeline such as MAKER [12] is used to perform the annotation, combined with manual expert inspection. No matter how good the annotation process is, though, the quality is plagued by errors of the draft assemblies, the improvement of which directly benefits annotation [54].

1.3 Assembly quality evaluation

Given the complexity of the problem, genome assembly has seen many quality metrics developed over the years — contiguity, number of assembled bases, number of misassemblies and their types, number of complete and/or fragmented genes, and number of mismatches and/or indels are only some of the common ones. Along with them, more indirect approaches have been used, such as haplotype inconsistency [41] which looks at whether the assembly stays true to the ploidy of the genome.

Contiguity is perhaps the most straightforward family of metrics. One of the early ways in which it has been reported is the N50 value, which is the length of the contig which splits the set of contig lengths into two. More precisely, the sum of lengths of all contigs that are equal or larger must be equal or larger than the sum of lengths of smaller contigs [22]. As explained in first Assemblathon [22], however, this metric cannot be used to fairly compare contiguity of two different assemblies, as it does not take actual genome size into account but instead uses the sum of all contig lengths as the proxy. For this reason, the NG50 metric was introduced, which requires that the length of the contig which splits the two sets and length of larger contigs sum up to at least half of the genome size (hence the G in NG50), instead of comparing them

to smaller contigs. This means that the total sum of contig lengths must be at least half the genome size, or the metric does not make sense. NG50 still does not give us the full picture, though. It suffers from giving us an overly optimistic look into the correctness of the assembled contigs. This number can be inflated by simply joining as many contigs as possible, giving a highly contiguous, but erroneous output, so there needs to be a counterweight to mitigate this. As proposed in GAGE paper [55], contiguity metrics can be corrected by breaking contigs at misjoin locations or at indels 5bp or longer, and then calculating N50 or NG50. Similarly, Assemblathon used aligned contig paths in order make the correction, and normalized N50 has been proposed by *Mäkinen et al*[44]. Finally, QUAST paper [27] introduced the NGA50 (A for aligned) metric, which is commonly used today. NGA50, similarly to corrected contiguity in GAGE, breaks contigs at misassembled points. The broken contigs are then aligned to the reference and the unaligned portions are removed. The contigs outputted from this procedure are then used to calculate NG50, giving us the NGA50 metric that is comparable between assemblies and which counterweighs misassemblies. The caveat here is that a reference genome is needed to perform the alignments, which may not always be available.

Misassemblies can also be reported as a separate metric. They are defined as a misjoin of two sequences, i.e., the resulting sequence has structural difference from the target genome. There are different types of misassemblies, as shown in Figure 1.2 — relocation, translocation, and inversion. Depending on what the assembly is used for, they can have varying severity.

The process of identifying misassembly points may require a reference. In QUAST, contigs or scaffolds are aligned to the reference and if parts of one sequence align to distant (e.g., >1000bp) sections of the reference or if they heavily overlap, it is considered an (extensive) misassembly. It is important to note that misassemblies identified in this manner may also indicate genuine structural variants.

For *de novo* assemblies, using a reference is not an option, and so tools such as ALE [17] and REAPR [28] have been developed that do not need one and can identify misassemblies through other means. ALE reports uses Bayes' theorem in order to determine the likelihood

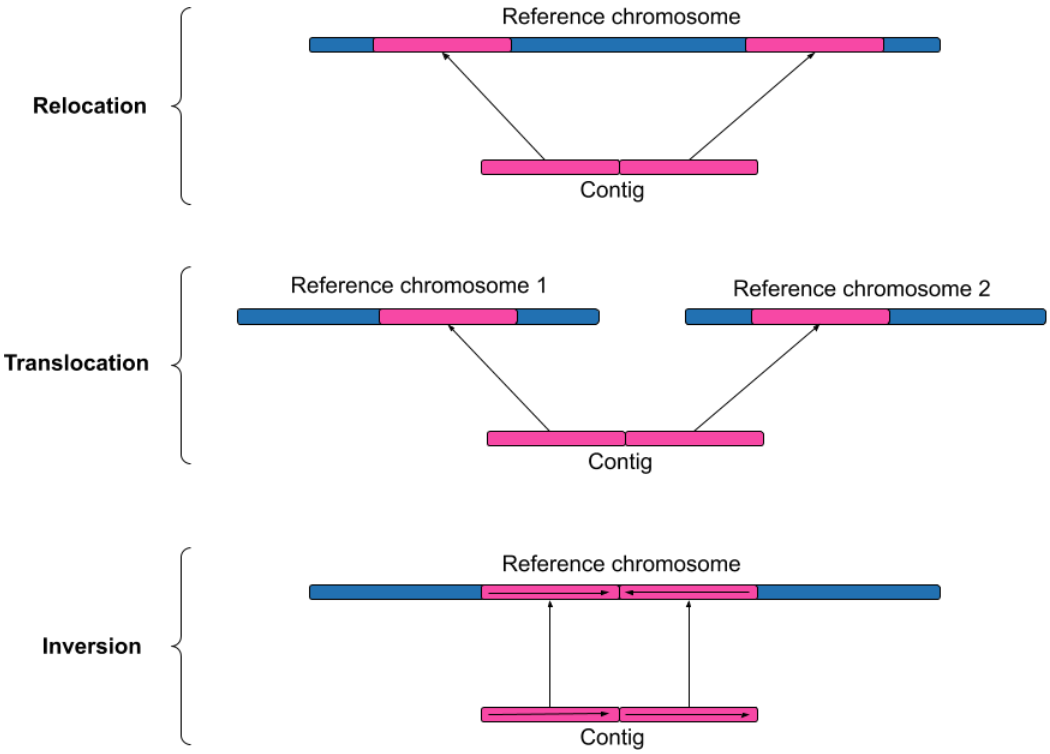

Figure 1.2: Visual explanation of three types of misassemblies: relocation, translocation, and inversion are shown. Blue sequence is the reference genome and pink is from assembled contigs, each containing a misassembly.

that the provided assembly was built from the input set of reads. Indirectly, this gives us a "score" for that assembly that, by itself, is not meaningful but can be used to compare to other assemblies to determine which one is of higher quality. Internally, ALE has four probability sub-scores per base, based on whether aligned read bases are congruent with assembly, how well insert lengths match expected distribution, whether depth agrees with expected, and whether k-mer frequency is as expected. With this information, it is possible to locate places in the assembly that are likely erroneous. Looking at insert lengths specifically can pinpoint where a misassembly has occurred. Similarly, REAPR also assigns a score to each base to identify misassemblies. For every base in the assembly, fragment coverage distribution is calculated, i.e., distribution of mapped paired end reads for which this base lies in between the pair. If this distribution does not match the theoretically expected, it indicates a possible misassembly.

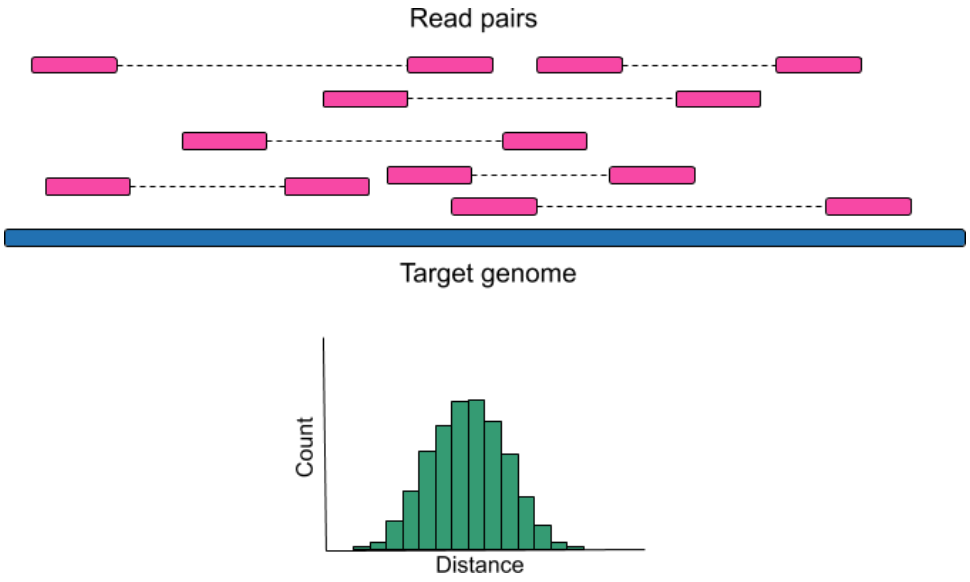

Figure 1.3: Distances between paired end reads follow a known distribution. In order to find misassemblies, they can be mapped to a reference and their distances inspected — if pairs in a region do not follow the expected distribution, a misassembly may have occurred. In the figure, blue sequence is the genome being assembled and pink are paired end reads. Often, a normal distribution is assumed with mean and variance inferred from the majority of read pair distances, as is done in ALE paper.

De novo assemblies may also be evaluated on a more pragmatic way, by looking at reconstructed genes in the draft assembly. As elaborated in Section 1.2, one of the primary goals of a *de novo* assembly can be gene annotation. *Parra et al*[50] introduced the concept of Core Eukaryotic Genes (CEGs) that are well conserved and present in low copy numbers across eukaryotes. Determining how many CEGs are present has been shown to be a good proxy for determining the total number of recovered genes. Following this concept, Benchmarking sets of Universal Single-Copy Orthologs (BUSCOs) [69] have been identified using OrthoDB to assess assembly quality and this is the premise of the software under the same name [58]. The chosen sets of representative genes had the requirement to be present in >90% of species with a single copy for each orthologous group, and so from an evolutionary standpoint, we can expect these genes to be present in newly sequenced genomes. The BUSCO software specifically outputs three metrics — complete, fragmented, and missing BUSCO genes, providing some granularity when assessing quality.

It has been established numerous times that draft assembly quality can vary enormously between assemblers, technologies, data quality, and the genomes being assembled [55] [22] [11]. GAGE paper explains that contiguity can increase at the order of 30x if the reads are error corrected, indicating the impact data quality can have on the output. Additionally, genome size and complexity has been emphasized as a major determinant of final assembly quality. Seeing as this is something inherent to the genomes being assembled, it hints at the importance of benchmarking novel assembly methodologies on a number of genomes, in order to fully understand how well they deal under different circumstances. Finally, it should be no surprise that, given the complexity of the problem, even when assembling the same genome with the same data, performance may vary between different assemblers and strategies significantly, as shown in both Assemblathon 1 and 2.

Seeing as data is of fundamental importance, it can be insightful to understand the properties of the data we are dealing with. *Simpson* [59] has introduced a method that assess read quality, fragment size distribution, and genome coverage. This is accomplished by looking at overlaps between reads themselves to determine sequencing error rate and classifying branch types in DBG to identify errors, variants, and repeats. Additionally, using tools such as FastQC [7] can provide read error profile and a number of statistics that can help understand what quality we can reasonably expect. Figure 1.4 shows an example of what information FastQC can provide — per base Phred quality scores [24]. Phred quality score Q is assigned to every base and is defined as:

$$Q = -10\log_{10} P \tag{1.1}$$

Where P is the probability of an erroneous base call. The maximum score shown in Figure 1.4 is 40, which gives us the probability of 1 in 10,000 that the base call was wrong. The trend of quality dropping towards the end is expected in short-reads [21] and since most bases have >=30 quality, this can be considered good quality data. Deviations from this pattern may indicate that the data is faulty and require a closer look into the sequencing procedure as well as lowering the possible quality of assembly.

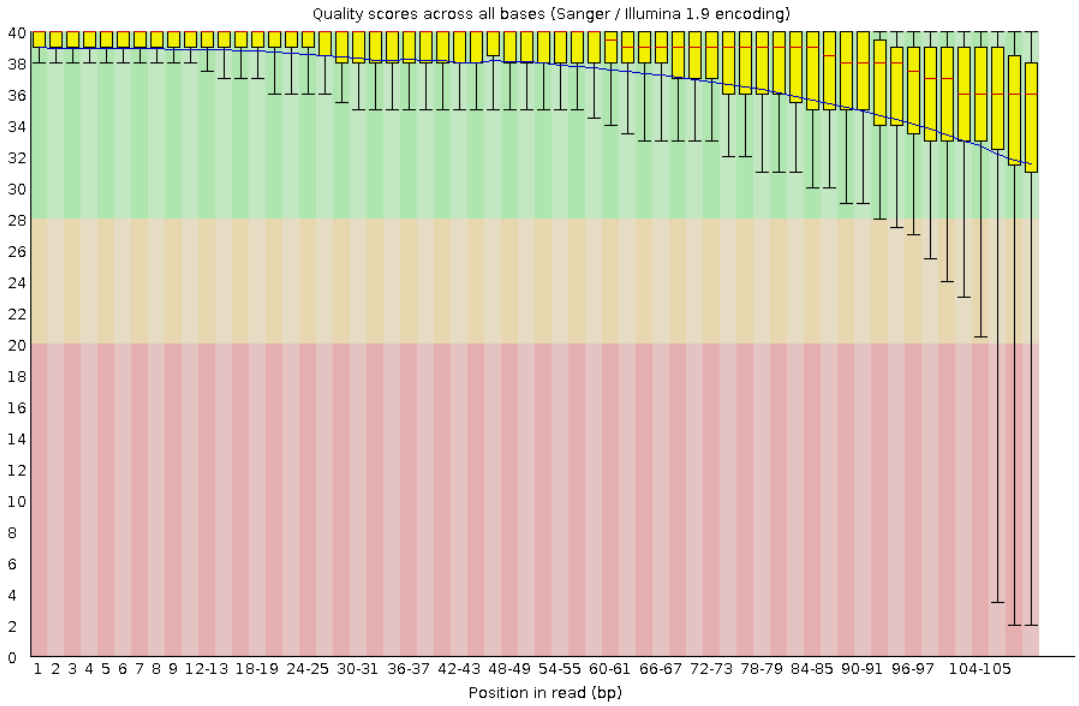

Figure 1.4: One of the outputs of the FastQC software, showing 110bp *C. elegans* read base quality with Phred score on the Y axis.

Other than metrics described so far that look at the properties of the assembly output alone, some experiments utilize independent validation from external sequencing. For example, to assess the accuracy of the panda genome assembly, nine Bacterial Artificial Chromosomes (BACs) were sequenced using Sanger technology, assembled, and aligned to the assembly [40]. These sequences were not part of the assembly and were only used for quality check. There are other technologies, such as optical maps, that have been used for this purpose. *Church et al*[16] were the first to use them in their assessment and disambiguation of repetitive regions of the mouse genome assembly.

1.4 de Bruijns Graphs

The work presented here improves upon DBG assemblies and so the concept is more closely explained. DBGs are directed graphs defined on an alphabet S and node size k, where all of the nodes are composed of strings of size k containing the characters from the alphabet. For

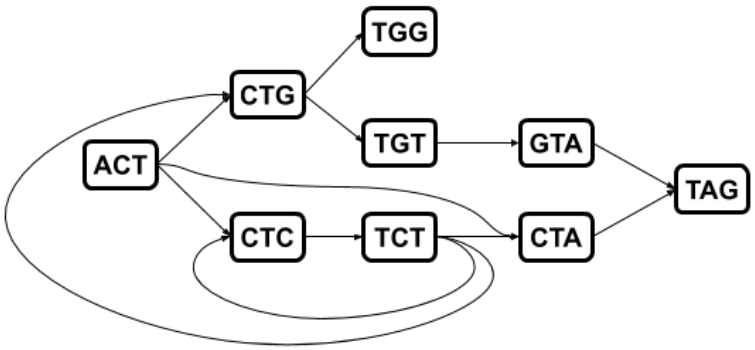

Figure 1.5: A simple DBG on alphabet $S = \{A, C, T, G\}$ and node size $k = 3$.

every pair of nodes x and y there is a directed edge going from x to y if the $k-1$ suffix of x is the $k-1$ prefix of y, i.e., they overlap by $k-1$ symbols. In graph theory, a DBG has a node for every permutation of S symbols. In the genome assembly problem, however, a variant is used wherein the nodes are all read substrings of size k, known as k-mers, and the valid symbols are $S = \{A, C, T, G\}$. An alternative to this is sometimes used that represents k-mers as the edges and $k-1$ overlaps as the nodes, as is done in the EULER assembler [53]. In this setup, nodes represent branching points.

Many short-read *de novo* assemblers make use of a DBG based approach [2, 8, 14, 30, 43, 73]. The assembly process usually starts out by splitting all reads into k-mers and storing them in a hash table or a Bloom filter [14, 30, 61]. This provides the benefit of being able to query node adjacency in constant time as opposed to searching for overlaps.

1.4.1 Bloom filters

A Bloom filter [10] is a probabilistic data structure that has the operations of a set: insertion of an element and querying for the presence of an element. The set is typically implemented as a bit vector, initialized with all zeroes. The elements must be hashable, i.e., there must exist a hash function that takes the elements as its input. On insertion, the element is hashed into a predetermined number of hash values, h, which represent indices in the bit vector that get turned into ones. Essentially, the content of the element is compressed into only h bits, making Bloom filters very memory efficient. To query for the presence of an element, like for

insertion it is hashed into h values and those are used as indices into the bit vector to check for the presence of one bits. Because the bit vector is limited in size, some of the bit indices from different elements may overlap. This can produce a query false positive if the queried element bit indices happen to land on indices of other previously inserted elements, even if the queried element has never been inserted. The chance of a query false positive, i.e., the False Positive Rate (FPR) of a Bloom filter is estimated as [10]:

$$FPR = (1 - (1 - \frac{1}{b})^{hn})^h \approx (1 - e^{-(hn/b)})^h \tag{1.2}$$

Where b is the number of bits in the bit vector, h the number of hash values per element, and n the number of distinct elements. The chance of bit index overlap between elements and thus false positives is increased with the reduction in size of the bit vector, as seen in the FPR formula. In this way, Bloom filters allow the user to make the trade-off between memory usage and produced false positives. It is worth noting that false negatives are not possible in a Bloom filter designed this way, as once an element is inserted, its one bit indices stay unchanged. Any subsequent query for the presence of this element will return true. Since Bloom filters are highly memory efficient, they have been widely used with memory intensive genomic data [14, 30, 63, 67]. In fact, they have enabled various bioinformatics tools such as assemblers and scaffolders to work with large genomes and are also used in this work for scalability.

1.4.2 Repeats

One of the main confounders of genome assembly are repetitive sequences. If the same DNA sequence is repeated at a single locus, potentially many times, it is known as a Tandem Repeat (TR). Otherwise, if the same sequence appears at different loci across the genome, as Transposable Elements (TE) do, it is an interspersed repeat. In the context of DBG based assembly, a repeat that is at least $k-1$ bases long will create a false edge, seeing as any sequence overlap of that length creates an edge. While constructing a DBG, it is impossible to disambiguate repeats that are $k-1$ bases or longer, and this task is left to the downstream stages of

the assembly process.

To illustrate the magnitude of the problem repeats pose, it has been estimated that half or more of the human genome is comprised of repeats [19]. The typical length of a TE is on the order of several kilobases (kbp), ranging up to 20 kbp in eukaryotes [36]. A third of mammalian genomes consists of TE and in vertebrates such as zebrafish they make up more than half of the genome [13]. Since current short-read lengths are in the 100–300 base pairs (bp) range [9], they are unable to span a large number of TE. On the other hand, TR can have motifs as short as 1 bp. They are generally classified as microsatellites if they fall within 1–10bp range, minisatellites if in the 10-100 bp range, and satellites if over 100 bp [32, 42, 65]. While the motif may be fully spanned by a short read, the number of repetitions may not be possible to estimate with short reads alone.

1.4.3 Read coverage

Due to non-uniform genome read coverage in the sequencing data [23], regions of the genome with less short-read coverage will have a more sparse overlap between reads, whereas a highly covered region will have an abundant number of reads that have significant overlap. This is where the choice of k comes into play — a smaller size will capture the overlap in both low and high coverage regions, but will additionally include many spurious overlaps due to repeats, complicating the graph. On the other hand, a larger size will reduce the number of spurious overlaps, but genuine overlaps from lesser covered regions will be missed. To overcome this issue, some *de novo* assemblers, such as SPAdes [8], IDBA [52], SOAPdenovo2 [43], and MEGAHIT [39], use an array of k values, starting from a small k to achieve high connectivity and then proceeding to untangle the graph with higher k values. These methods demonstrate improved assembly quality, but they have been limited to small k value increments or multiple DBG constructions. This is problematic for large genomes (e.g., human), where the assembly graph is large and iterating over a number of k values may significantly inflate the run time. There is also room for improvement in the span of k that is utilized, as it is not efficient to reach

a high k value with small steps.

Another aspect of DBGs is that it relies on sufficient number of k-mers being errorfree, otherwise the overlaps would be missed or we would have false connections. One way this is achieved is by only considering only k-mers that are found above a certain multiplicity (e.g. they appear at least twice), as is done in the ABySS assembler [30]. These k-mers are known as solid k-mers. Another method, often done in conjunction, is tip trimming, which removes short sequences that branch off a longer sequence, presumably due to an error or a false positive in a Bloom filter.

1.4.4 Iterative de Bruijn Graph construction

The IDBA assembler was the first to attempt to use multiple k values instead of a single moderate one in order to improve the output [52]. It introduces the concept of an accumulated DBG, which contains the information from its k value and all the previously processed ones. This accumulated DBG is constructed iteratively, by starting from a low k value and building an initial DBG, and then improving it with increasing k values. The initial graph is constructed from solid k-mers, i.e., k-mers above a given multiplicity in order to filter out erroneous ones. The next step is to assemble contigs, i.e., unambiguous walks in the graph, from which we obtain sequences relatively larger than the input reads. Because there is no ambiguity involved with these sequences, their constituent reads do not belong to any branching points, and so cannot be used to resolve them. For that reason, these reads are removed from the set of reads that will be used to improve the graph. After the construction of the initial graph, the iterative procedure takes the next k value in the ascending order. The step between these values may be as low as 1, but the paper suggests a higher step is possible in order to minimize run time. However, to obtain high quality contigs, a step of 1 is recommended. The improve the graph, situations such as the ones in Figure 1.6 are examined. Branching points such as the ones in case of a repetitive region or an erroneous join may be resolved with the new, higher, k value. Each iteration results in the accumulated DBG, that is being improved.

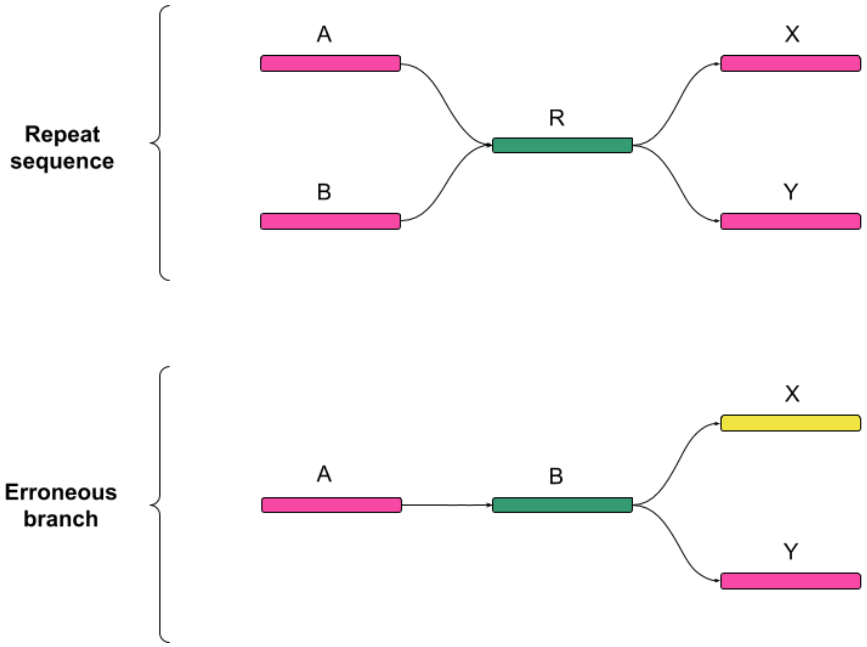

Figure 1.6: Two potentially resolveable branching points in a DBG. For a repeat sequence, a k-mer that spans the repeat R and further into A or B and X or Y can tell which of the ARX, ARY, BRX, BRY paths is correct. Specifically, for a k parameterized DBG, a $k+2$ k-mer is required for support, touching the adjacent nodes with one nucleotide each. Similarly, for a branching point due to an error, a spanning k-mer can cut out the false branch. In this case, the X contig is falsely connected to B, and so a k-mer spanning A, B, and Y, with one nucleotide in A and Y each would suggest that this is the true path.

The algorithm for the resolution transforms all edges in the graph into nodes of $k+1$ size. Then, every pair of neighbouring nodes in this transformation is joined by an edge if it represents two consecutive $k+1$ k-mers and there exists a $k+2$ k-mer containing the sequence of these two nodes joined together. This $k+2$ k-mer is searched for in the set of leftover reads, after reads from previous contigs are removed. The described iterative procedure is repeated for every k value until the specified k_{max} is reached. For each accumulated DBG obtained, reads that belong to contig sequences are removed, and so the set of working reads is decreased on every iteration. Additionally, dead end sequences (tips) shorter than $2 \cdot k$ are trimmed. After all the iterations are done, bubbles are merged. Here, bubbles refer to two parallel paths due to a Single Nucleotide Polymorphisms (SNPs) or indels which appear as two similar sequences. The summary of IDBA algorithm can be seen in Listing 1.1.

Listing 1.1: Pseudocode of the IDBA algorithm. After the construction of the initial DBG, iterative improvements are performed by increasing k and resolving branching paths.

```
# IDBA algorithm summary
k = k_min
build(DBG_k, solid_kmers)
while k < k_max:
  remove_dead_ends(DBG_k)
  reads = reads \ reads_in_contigs(DBG_k)
  DBG_k+step = improve(DBG_k, leftover_reads)
  k += step
remove_dead_ends(DBG_k)
merge_bubbles(DBG_k)
connect_contigs(DBG_k, mate_pairs)
output_contigs(DBG_k)
```

1.4.5 Multisized de Bruijn Graph

The SPAdes assembler uses the concept of a multisized de Bruijn Graph in order to utilize different k sizes. Like IDBA iterations of k values, this approach starts with a low k value and constructs a regular DBG from input reads. The resulting DBG is then condensed by unifying unambiguous k-mer paths into unitigs, as shown on Figure 1.7, which is a common simplification operation in DBG assemblers.

After this operation, the outputted unitigs can form long sequences, often larger than the original reads in non-repetitive regions where the graph does not branch. The next step is to increase the k value and k-merize both the obtained unitigs and original reads to build a new graph from. This new graph benefits from the long sequences built in the previous graph and is subsequently able to get away with larger k-mer size. Each graph built is also processed by simplification operations such as removal of bubbles, chimeric reads, and tips (Figure 1.8).

In IDBA, a k step of 1 is used, iterating over a small range of values. In SPAdes, however, in the interest of minimizing run time and maximizing the range of iterated values, a step of >1 is used. The important thing to keep here in mind is that SPAdes was designed to assemble bacterial genomes, and for that purpose this approach is feasible. However, building a DBG

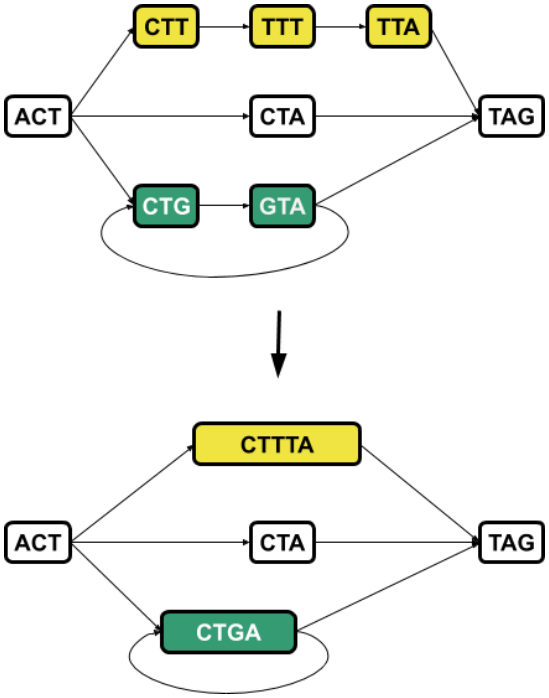

Figure 1.7: An operation commonly done by DBG assemblers – condensing unambiguous k-mer paths into unitigs.

is an expensive operation, both in terms of run time and memory. Even for an assembler optimized for large genomes such as ABySS 2 [30], the initial DBG step takes the most time and memory. To assemble a large genome, reconstruction would be prohibitively slow, giving motivation for an alternative method.

1.5 Research question

Seeing as DBGs are parametrized by a single k value and that so far improving the graph with larger k values was either with small increments or reconstruction of the whole graph, the question is — can this be done better? Specifically, is it possible to use a larger k step to improve upon the initial DBG in a scalable manner that would not require reconstruction. Additionally, the goal is to be able to utilize the new approach on large genomes, meaning that run time and memory requirements should be reasonable.

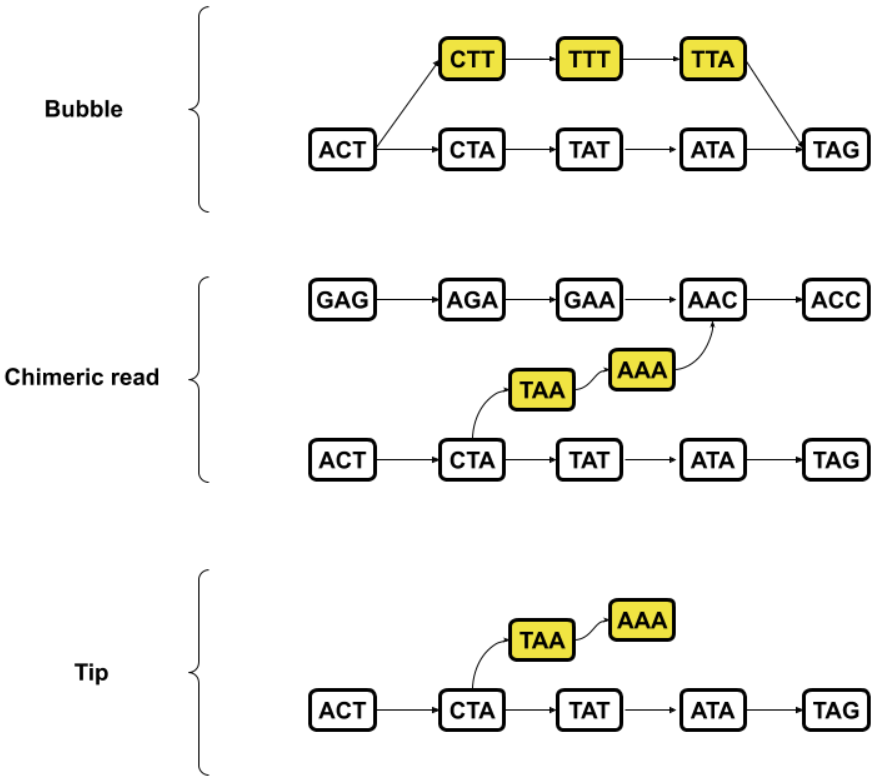

Figure 1.8: Three types of opportunities for graph simplification operations — bubble, chimeric read, and tip. A bubble can be caused by a SNP, an indel, or an error. A chimeric read aligns to distinct, often distant, loci and so erroneously connects contigs. The tip is a short sequence branching off a longer contig, often caused by a read error or a Bloom filter false positive.

1.6 Contribution

The work presented here includes a novel algorithm for the purpose of scalably using an additional k value, and addresses various challenges encountered. Unlikely previous approaches that iterate over a list of k values, here, we pick a single additional k value that bypasses the values in-between, allowing for a faster algorithm. This novelty also allows us to efficiently utilize a Bloom filter for k-mer storage, as it only has to be built once for the k size that is being used. This paradigm enables scalable improvement of the assembly graph and results in higher contiguity and more genes recovered.

Chapter 2

Methods and findings

To address these issues, RResolver utilizes additional short-read information in a scalable manner by taking a large step in k value from the one used to construct the DBG in order to resolve repeats. This large step bypasses multiple short k increments, thus reducing the overall run time, but comes with a set of challenges that have been explored in this study. The initially constructed DBG is worked on directly, without the need for any costly graph reconstruction steps. Additionally, to minimize memory usage, a Bloom filter is employed for k-mer storage. Herein, DBG assembly k is denoted as $k_{assembly}$ and the larger k used by RResolver as $k_{rresolver}$.

2.1 Results

To improve a given DBG assembly, RResolver attempts to find k-mers of size $k_{rresolver}$ along assembly graph paths surrounding a repeat in order to evaluate their correctness. $k_{rresolver}$ sized k-mers are first extracted from reads and stored in a Bloom filter [10] for efficient memory use. To find $k_{rresolver}$ k-mer counts along a path, a sliding window of size $k_{rresolver}$ is used, querying the Bloom filter for presence or absence at every step with a step size of 1bp. In order to make k-mer extraction from reads and paths fast, ntHash [45], a rolling hash algorithm for nucleotide sequences, is used to efficiently calculate hashes of successive k-mers.

Figure 2.1 shows a typical situation in which paths can be evaluated. All paths of three

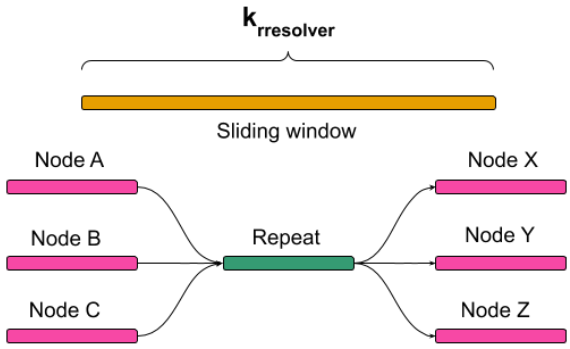

Figure 2.1: For a repeat to be considered, it has to be short enough so that the window of $k_{rresolver}$ size spans the adjacent nodes and has sufficient room to do a number of moves, i.e., tests, dynamically determined based on sequencing coverage.

nodes are evaluated; in Figure 2.1, that would mean all possible paths from nodes to the left to nodes to the right: ARX, ARY, ..., CRZ where R is the repeat. The algorithm is applied to every repeat short enough for the sliding window to span the whole repeat node sequence, overlap the nodes adjacent to the repeat along the path in question, and to perform a sufficient number of tests (sliding window moves). The number of tests is dynamically determined based on the number of expected $k_{rresolver}$ k-mers (if the path were genuine) along each tested path. This number is obtained from the $k_{assembly}$ coverage of the path, provided by the assembler.

$k_{rresolver}$ sized k-mers found are tallied for every path and ones where the k-mer count is above a threshold are considered supported. The unsupported paths are removed from the graph, potentially duplicating the repeat in the process.

2.1.1 False positives

Since Bloom filters may return false positives on query operations, they need to be taken into account when considering path support. To counteract these false positives, a threshold is set for the number of $k_{rresolver}$ k-mers that need to be found along a path for it to be considered supported. A sufficiently high threshold tolerates a number of false positive matches in the Bloom filter before considering a path supported.

The number of false positives depends on a number of factors, such as the number of tests done per path (which depends on sequencing coverage), the number of possible paths, and

the FPR of the Bloom filters. As RResolver is used alongside a short-read assembler, we can assume that it has the same memory constraints and works with the same read dataset. Success with the low memory footprint assembler ABySS shows that RResolver can work with tight memory constraints and gives us an upper limit on expected Bloom filter FPR.

The Bloom filter FPR increases the more elements are stored. To balance this out with sufficient $k_{rresolver}$ k-mer abundance, the number of extracted k-mers per read is equal to the support threshold. This effectively makes one read found along a path sufficient to consider that path supported. K-mers are extracted from the beginning of the read, reducing the effect of the read quality drop towards the 3' end [21].

Figure 2.2 shows the histogram of $k_{rresolver}$ k-mers found along the tested paths for the $k_{assembly} = 95, k_{rresolver} = 134$ *H. sapiens* NA24631 assembly with a threshold of 4 (default). There is a clear separation between the two distributions of unsupported and supported paths, with the first noticeable histogram bar of supported paths at 4 k-mers suggesting that the threshold of 4 is appropriate. The paths with Bloom filter false positives are found between the two distributions, however, due to low FPR of 3.15e-08% for this assembly, they are few and not visible. The spike at 20 k-mers is due to a default minimum number of sliding window moves of 20.

2.1.2 Varying coverage

As the read coverage varies across the genome [23], the number of $k_{rresolver}$ k-mers expected along each path differs. In order to reliably determine whether a path is supported, RResolver calculates the number of tests required to find sufficient number of $k_{rresolver}$ k-mers along a path.

Given assembly $k_{assembly}$ k-mer coverage of a graph node, i.e., the sum of multiplicities of all the $k_{assembly}$ k-mers that make up that node provided by the assembler, the number of $k_{rresolver}$ k-mers can be found proportionally. Since every read provides $l - k_{assembly} + 1$ k-mers of length $k_{assembly}$, where l is read length, the number of reads that have contributed to the node

Figure 2.2: Histogram of the number of $k_{rresolver}$ k-mers found along all tested paths for *H. sapiens* NA24631 assembly. The number of extracted k-mers per read is 4 and as can be seen on the figure, a threshold of 4 to consider a path supported separates the distributions of unsupported and supported paths well. The paths with Bloom filter false positives are found between the two distributions, but because of the low FPR, they are few and not visible. The spike at 20 k-mers found is due to the lower limit on the number of sliding window moves of 20.

in question is determined. Given the number of reads in a node and the length of that node, the approximate number of bases between subsequent reads is calculated as the node length over the number of reads. To find a read along a path, on average, the sliding window should move the number of bases equal to the average number of bases between reads. As each read provides a number of $k_{rresolver}$ k-mers equal to the support threshold, a constant equal to the threshold is added to the formula to ensure that all extracted $k_{rresolver}$ k-mers are found.

Estimating coverage also allows the algorithm to skip over low covered regions of the graph where $k_{assembly}$ has been an appropriate choice and further increase in k size is not helpful. The criteria to skip a region is simply if the number of required tests is greater than the possible number of moves the window can do. For a sliding window, there is only so many moves it can perform while still overlapping all three nodes that form the path in question, giving an upper limit on the number of tests that can be done in a repeat.

If any path in a tested repeat is found to have low coverage such that doing a sufficient

number of sliding window moves is impossible, the whole repeat is skipped. Skipping the repeat is done because, despite possibly knowing whether other paths are supported, the repeat as a whole cannot be resolved accurately without complete information, as trying to resolve it could lead to misassemblies.

The coverage estimation formula (Equation (2.4), further elaborated in Methods section) is only an approximation and its output should be interpreted conservatively. Because of this, the number of tests to be done for any path is multiplied by a factor which can be parametrized. The factor used in the results shown here is 4. The formula inaccuracy is also the rationale behind setting a minimum number of sliding window moves (20). The formula may overestimate the number of $k_{rresolver}$ k-mers expected and perform too few tests, which this lower limit prevents.

There are a couple of possible sources of the coverage formula inaccuracy. First, if a read has an erroneous base call within the 4 extracted $k_{rresolver}$ k-mers, RResolver will miss those k-mers when querying a genuine path, reducing its support and potentially resulting in a misassembly. This is especially problematic for larger reads and $k_{rresolver}$ k-mers, as there are more bases that could be erroneous. It may be the case that the erroneous base call is found at the end of a $k_{rresolver}$ k-mer, while a preceeding $k_{assembly}$ k-mer of the same read might not have that error. This contributes to the inaccuracy of the proportion calculation in the coverage formula.

Another source of error is node $k_{assembly}$ k-mer multiplicity. Since the ABySS assembler provides average multiplicity along a node, the information granularity decreases the longer the node is. For a particularly long node with highly varying coverage, this will lead to overestimation in low-covered parts and underestimation in high-covered. Overestimating the number of expected $k_{rresolver}$ k-mers results in fewer tests done and therefore greater chance of missing a read on a genuine path. While it is possible to simply increase the number of tests overall by a factor, doing so reduces the number of repeats that can possibly be resolved, as the sliding window might not be long enough to do the required number of tests.

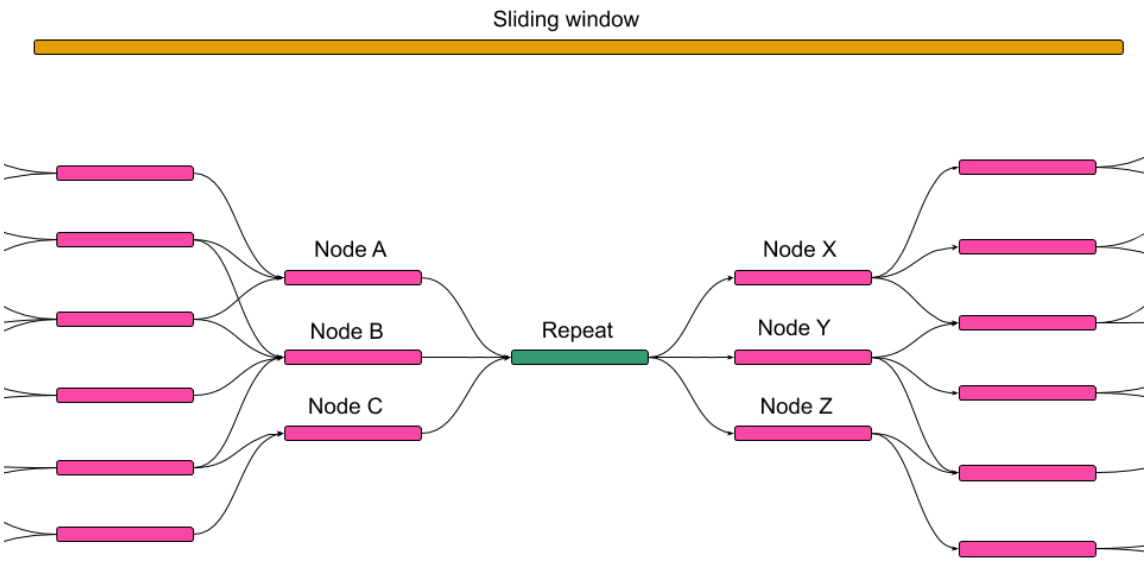

Figure 2.3: In complex repeats with a large number of paths, the nodes adjacent to the tested repeat tend to have short sequences before branching further. If the sequence from the path being tested is shorter than the window, the only way to test the path is to add the sequences of the nodes further away. E.g. if the tested path is ARX, all combinations of paths from the nodes preceding node A to all the nodes succeeding node X that fit within the window are considered. Support found in any of the paths is sufficient to consider the ARX path supported.

2.1.3 Complex repeats

In highly repetitive regions, the graph becomes particularly complex. The incoming and outgoing nodes from the tested repeat can be repeats themselves, often quite short. This can result in the sliding window being longer than the three nodes that are considered as a path. In such cases, the nodes that branch out of the incoming and outgoing nodes are also taken to be possible segments of the path, as shown in Figure 2.3. Branching is done to the extent to which is needed to accommodate the required number of moves with the sliding window to determine support.

Given the branching nodes, all the possible path combinations are tested and if at least one has a sufficient number of $k_{rresolver}$ k-mers, the initially considered path of three nodes is considered supported. For example, in Figure 2.3, if the path in question is ARX, all the nodes preceding node A and succeeding node X that are within the window would be used to form

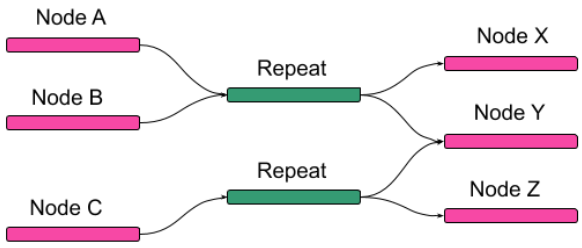

Figure 2.4: If the true paths in Figure 2.1 were found to be ARX, ARY, BRX, BRY, CRY, CRZ, the subgraph would be simplified as shown in the figure. Although the only nodes that can be unambiguously merged here are C and R, narrowing down possible paths benefits downstream algorithms.

the path combinations to test. If ARX is a genuine path, then at least one combination path should have reads, and so if any of them are found to be supported, then ARX is considered supported.

If the number of combinations explodes beyond a set threshold, the paths are randomly subsampled in order to limit run time and false positives. This threshold depends on the Bloom filter FPR, as increasing the number of tested paths increases the chances that a path will be supported by a series of false positive hits.

2.1.4 Repeat resolution

The resulting supported paths map may not unambiguously resolve paths in a repeat, but often simplifies a repetitive region. A simplified repeat does not necessarily immediately merge any nodes and increase contiguity, but helps downstream algorithms such as the contig and scaffolding stages of ABySS. Figure 2.4 shows an example of a possible simplification from the repeat in Figure 2.1. In cases where paths are unambiguously resolved, nodes are immediately merged.

Once all considered paths are examined, for each repeat, a map is constructed that that has supported outgoing nodes for each incoming node. As this may not unambiguously resolve the paths, a simplification procedure is implemented. Incoming nodes are grouped based on which outgoing nodes they have true paths to. All incoming nodes with the exact same set of outgoing nodes form one group. For example, if on Figure 2.1 in the main text, the true paths

are determined to be ARX, ARY, BRX, BRY, CRY, CRZ, incoming nodes A and B would be grouped together as they both go to X and Y, whereas node C would be in its own group as it goes to Y and Z (Figure 2.5). Each group then forms an instance of the repeat, resulting in all paths being supported in the new subgraph.

A summary flowchart of the algorithm can be seen on Figure 2.6. If the dataset used has multiple read sizes, the whole procedure is repeated for each size, starting from the smallest. Each read size works with a distinct $k_{rresolver}$ value either provided or automatically calculated.

2.1.5 Performance assessment

To assess the performance of RResolver and explore the parameter space, the tool was tested on 2x150bp and 2x250bp Illumina data from four human individuals with fold-coverages ranging between 43X and 58X. RResolver is integrated with the ABySS 2 assembler [30] and works on the output of the DBG construction stage. Figure 2.7 shows how the tool fits within the whole pipeline. Figures 2.8 and 2.9 show assembly quality results after scaffolding for a range of ABySS assemblies with different k-mer sizes, with and without RResolver in the pipeline. Assembly quality was assessed using QUAST [27] and BUSCO [58]. Each dataset was tested with a sweep of $k_{assembly}$ values, a common ABySS procedure, using a step size of 3bp. The optimal choice, with respect to NGA50, was kept plus the neighbouring values (+/-6 and +/-12). The choice of $-kc$ ABySS parameter, which specifies minimum k-mer multiplicity to filter out erroneous k-mers, was also selected for the optimal assembly. Dataset information can be found in Experimental Data section.

With RResolver, all ABySS assemblies achieved higher NGA50 and most have higher percentage of complete BUSCO genes, while some have fewer misassemblies. We explored a wide range of $k_{rresolver}$ values between $k_{assembly}$ and read size in order to assess the impact of $k_{rresolver}$ on assembly quality. This information helps develop a heuristic for choosing $k_{rresolver}$ in the absence of a reference that would maximize contiguity and complete BUSCO genes and minimize misasssemblies.

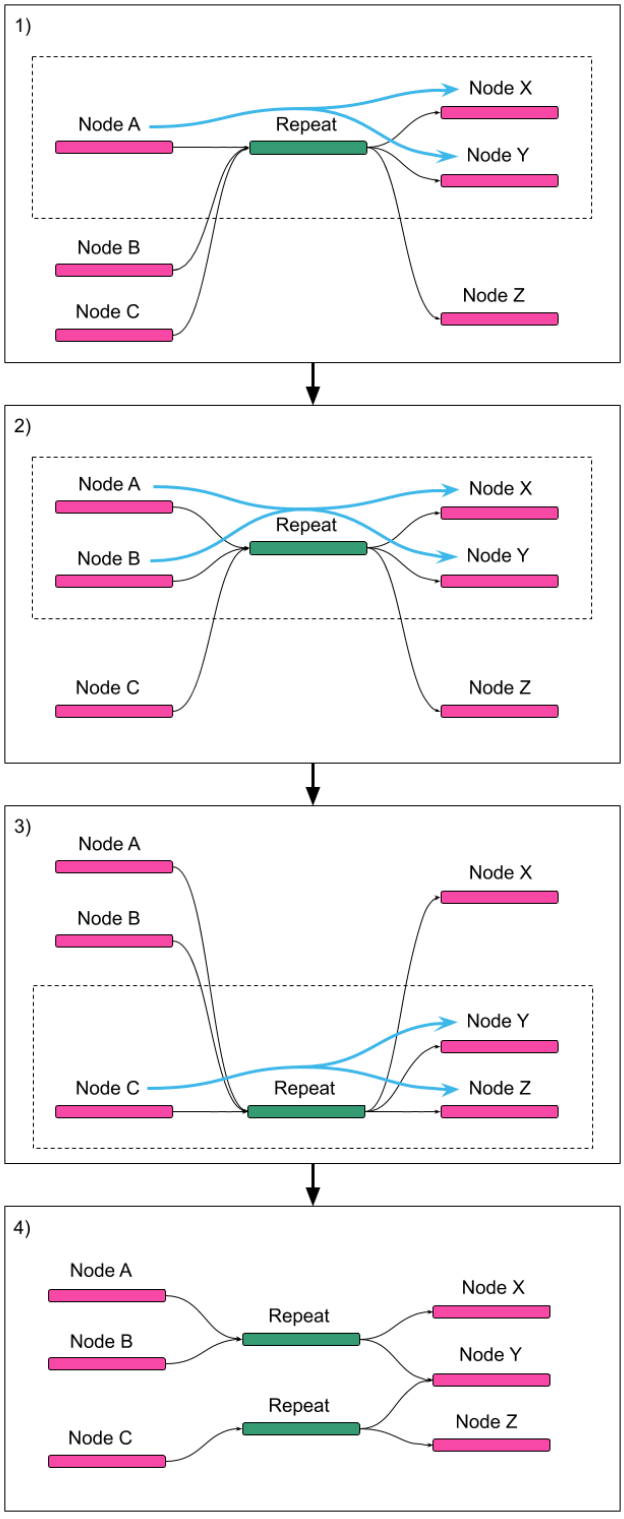

Figure 2.5: Step by step grouping of incoming nodes that have supported paths to the same group of outgoing nodes. In this case, paths ARX, ARY, BRX, BRY, CRY, CRZ were found to be supported. Each group then forms an instance of the repeat, which in this case results in two green repeat sequences.

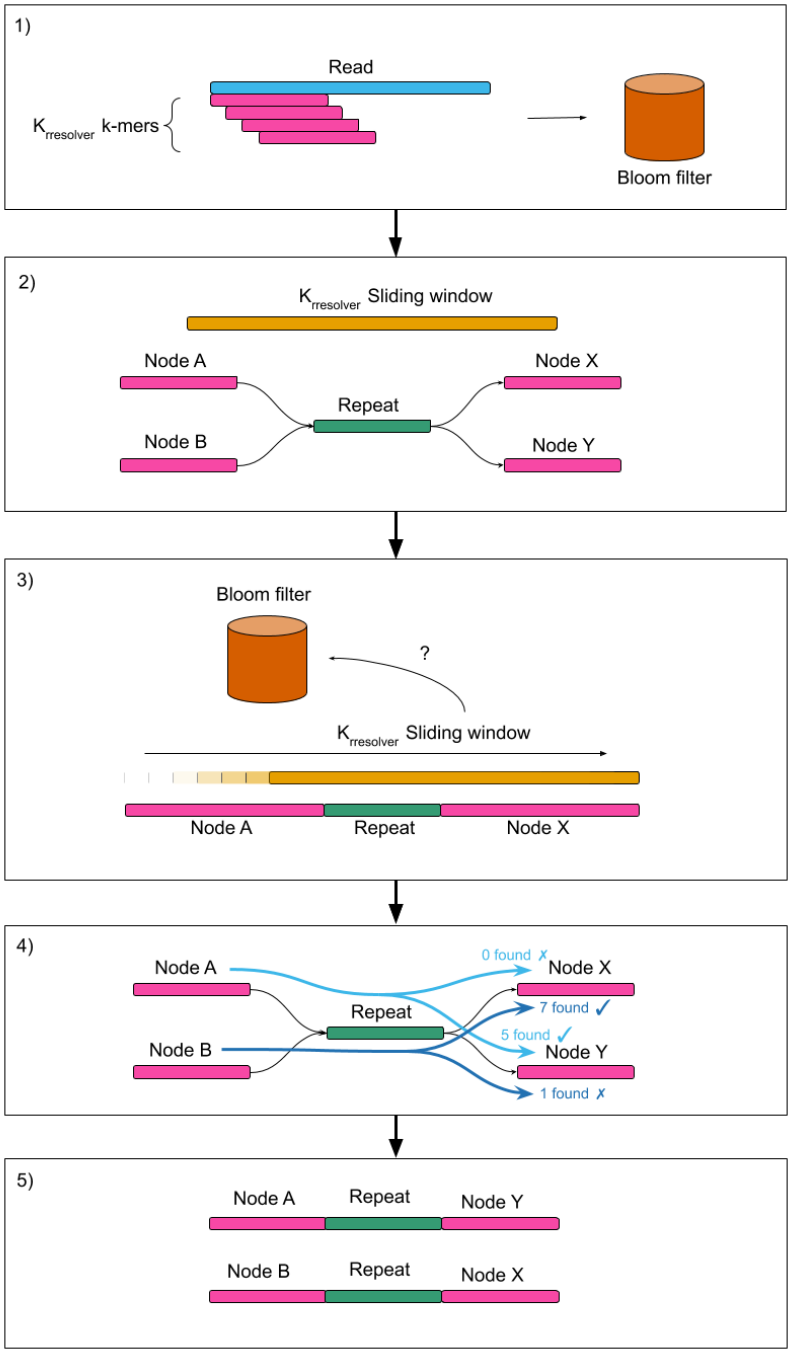

Figure 2.6: 1) $k_{rresolver}$ k-mers are extracted from input reads and inserted into a Bloom Filter. 2) Small repeats that can be spanned by the $k_{rresolver}$ window are identified. 3) All possible paths in the repeat tested by sliding the window along them, with 1bp slide step size, querying the Bloom filter on every step. 4) Paths with sufficient number of $k_{rresolver}$ k-mers are identified. 5) Repeats are simplified by removing unsupported paths and joining supported ones.

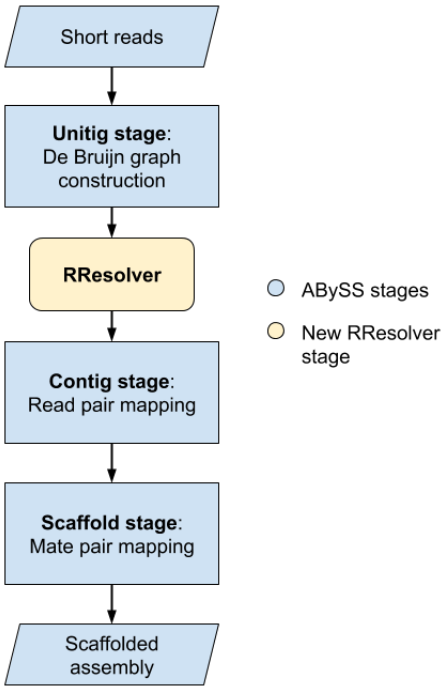

Figure 2.7: RResolver is ran after the DBG stage of the ABySS pipeline, benefiting the downstream contig and scaffold stage with an improved graph.

For 2x150bp reads, increasing $k_{rresolver}$ almost monotonically improves NGA50 and complete BUSCO genes for both datasets. There is, however, a trend of increased misassemblies past a $k_{rresolver}$ value of around 137, giving a limit for how high $k_{rresolver}$ should be. Since RResolver does not make any cuts in the sequences, the reduction of misassemblies comes from the additional repeat resolution enabling the downstream ABySS algorithms to better avoid making erroneous joins.

For 2x250bp reads, increasing $k_{rresolver}$ as high as the read length can deteriorate assembly quality, as shown by increased misassemblies and decreased complete BUSCO genes. The $k_{rresolver}$ values below around 186 are a more appropriate choice, as they yield increased NGA50 and increased complete BUSCO genes without too many additional misassemblies.

For both read 2x150bp and 2x250bp, lower $k_{assembly}$ values benefit more, reducing the effect of a wrongly picked $k_{assembly}$.

The results shown so far are for fold-coverages around 40-50X. Figure 2.10 shows the

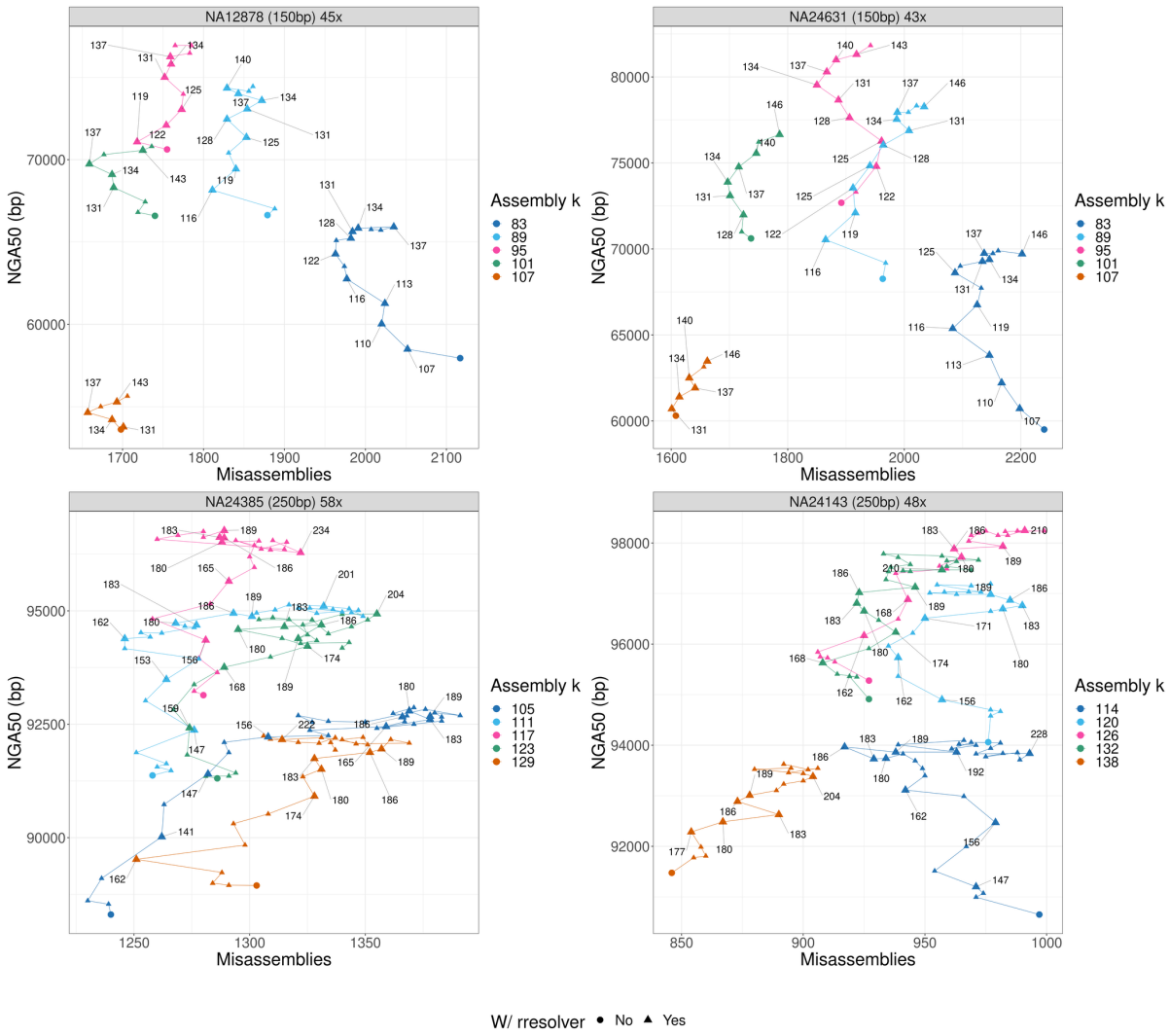

Figure 2.8: NGA50 and misassembly scaffold metrics with and without RResolver. High quality assemblies lean towards top left corner, with high contiguity and low misassemblies. The text labels indicate the $k_{rresolver}$ value used for each data point. Some text labels (for smaller triangles) and overlapping data points are omitted for clarity. All RResolver data points have higher NGA50 than the corresponding baseline assembly, and some have fewer misassemblies. The plot also shows that $k_{rresolver}$ value beyond ~137 and ~186 for 150bp and 250bp assemlibes respectively increases misassemblies without a significant contiguity increase, giving an upper limit when choosing the parameter.

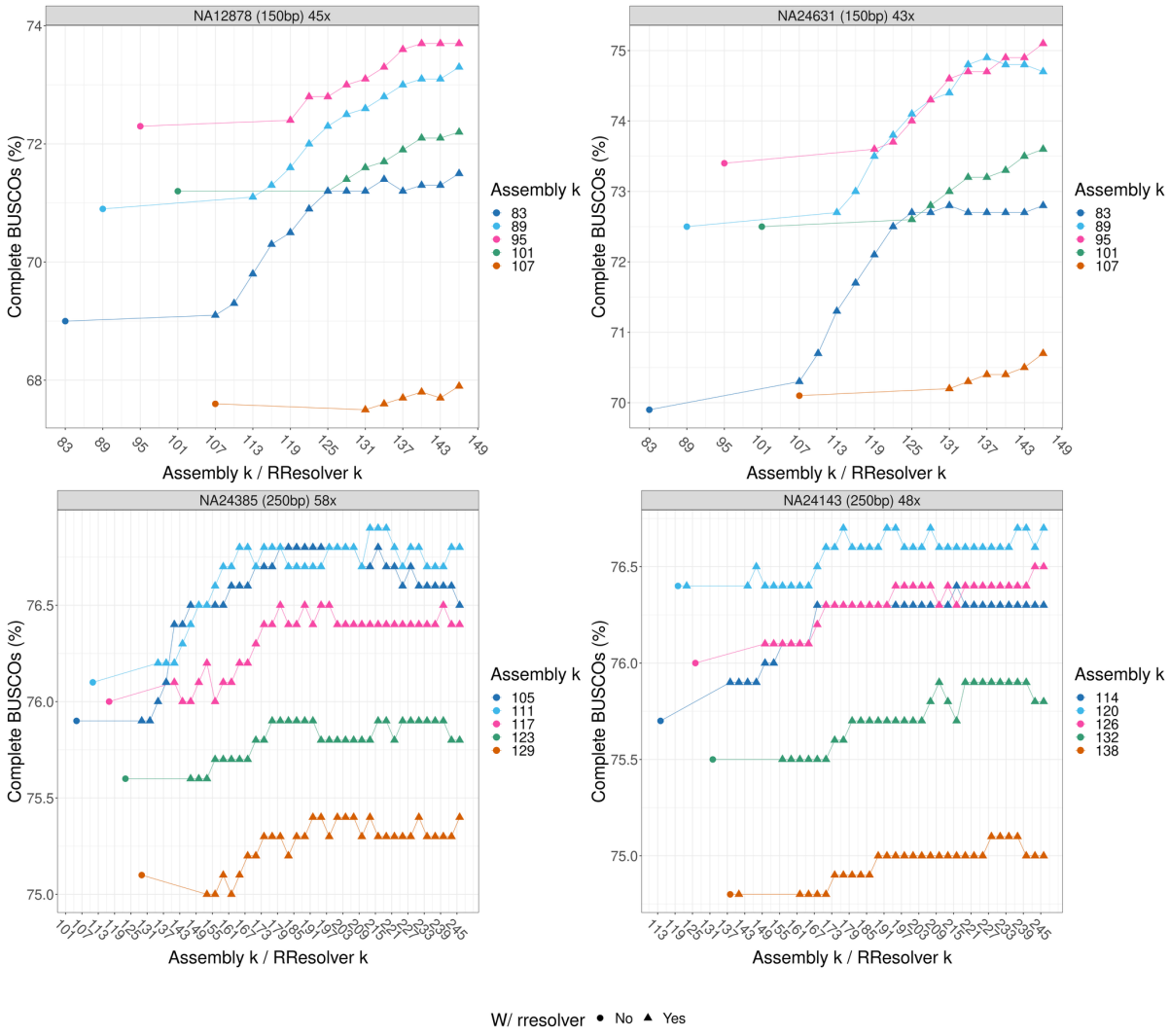

Figure 2.9: Similar to Figure 2.8, the plot shows an exploration of $k_{rresolver}$ values and its effect on complete BUSCO genes. A similar conclusion can be made that assembly quality does not noticeably increase after $k_{rresolver}$ values of around 137 and 186.

NGA50 and misassemblies statistics for a 2x150bp and a 2x250bp dataset, with the read coverage subsampled down to 28X and 33X, respectively, with a step of 5 using seqtk [4]. Across all assemblies, for both 2x150bp and 2x250bp reads, an offset of +45 between $k_{assembly}$ and $k_{rresolver}$ provides a good NGA50 increase while not introducing too many misassemblies. This gives a heuristic to base $k_{rresolver}$ value on and is the recommended approach if assessing multiple $k_{rresolver}$ values is too costly or the reference is unavailable. When using this heuristic, $k_{rresolver}$ should have an upper limit of 137 and 186 for 150bp and 250bp reads respectively in

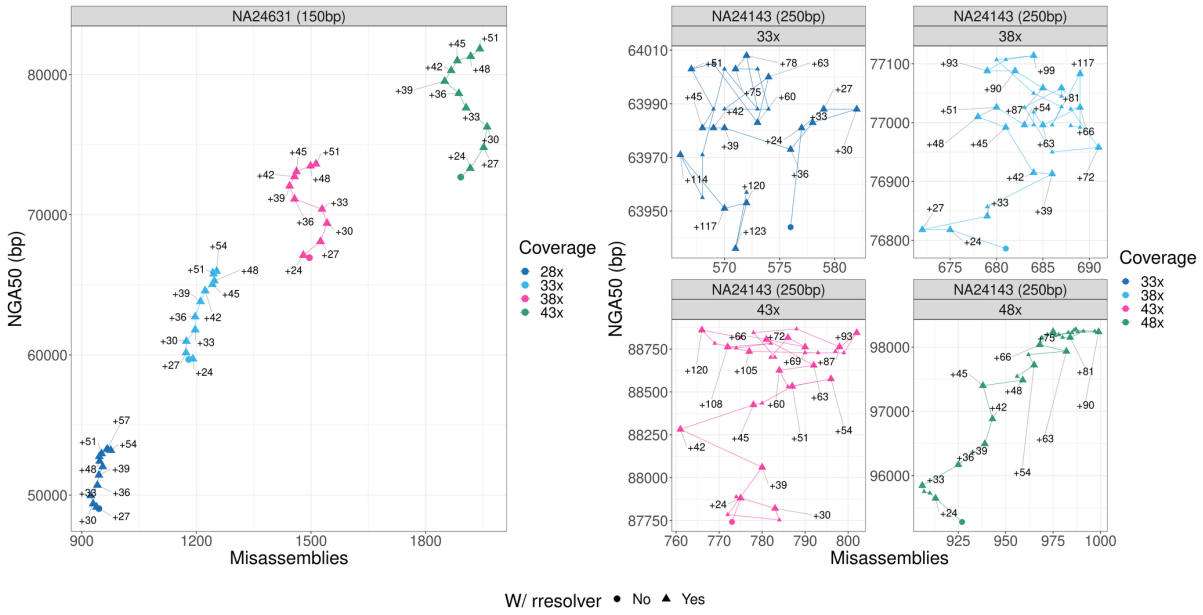

Figure 2.10: NGA50 and misassemblies plots for a 150bp and a 250bp dataset. The 250bp dataset has 4 separate plots as the datapoints are far away from each other between them. The text label on each datapoint indicates the offset between $k_{assembly}$ and $k_{rresolver}$. Across all coverages, an offset of +45 gives a good contiguity improvement without increasing misassemblies too much.

order to minimize introduced misassemblies.

To demonstrate that the algorithm performs well on genomes other than *H. sapiens* (3.1Gbp haploid genome size), Figures 2.11 and 2.12 show results for *C. elegans* and *A. thaliana* (101Mbp and 157Mbp genome sizes respectively) assemblies of 2x110bp and 2x101bp datasets, respectively. For both datasets, we applied the heuristic $k_{rresolver} = k_{assembly} + 45bp$, with read size as upper limit. Interestingly, for *C. elegans*, the ABySS assembly with the highest contiguity ($k_{assembly} = 71bp$) does not yield the highest contiguity final assembly with ABySS+RResolver ($k_{assembly} = 68bp, k_{rresolver} = 107bp$). The assembly yielding the highest contiguity is the one with a lower $k_{assembly}$, meaning that it retains more connections in the graph. While this also results in more false edges, those can be removed by RResolver whereas it cannot recover connectivity lost by higher values of $k_{assembly}$. For *A. thaliana*, while all assemblies have increased contiguity, they come with a trend of increased misassemblies. However, all of the assemblies have a fairly low number of misassemblies in the first place and

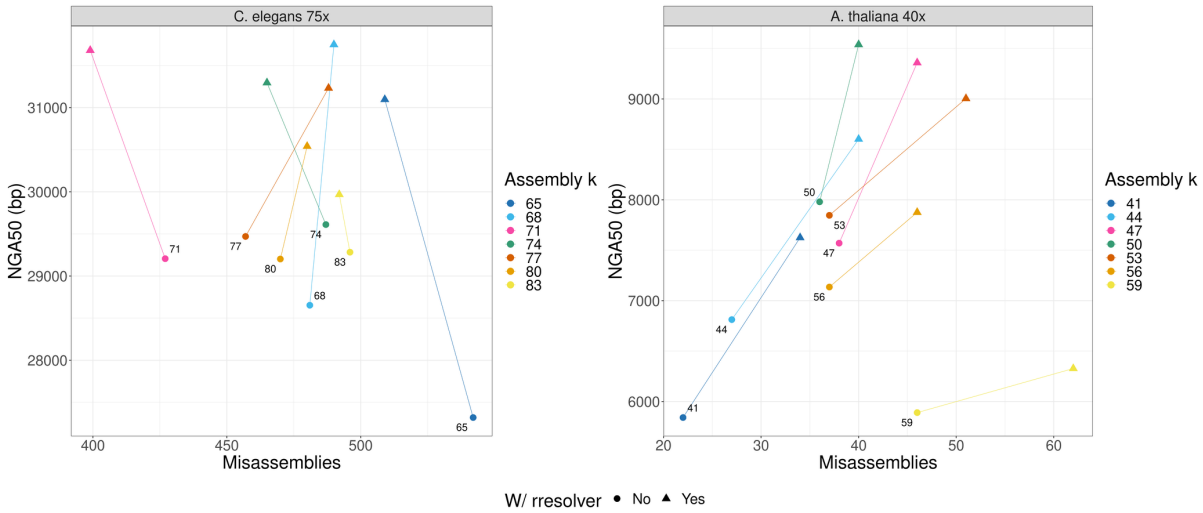

Figure 2.11: NGA50 and misassembly plots for *C. elegans* and *A. thaliana*. The $k_{rresolver} = k_{assembly} + 45bp$ heuristic is used, limited by read size of 110bp and 101bp. Both datasets see an improvement in contiguity, with a small increase in misassemblies.

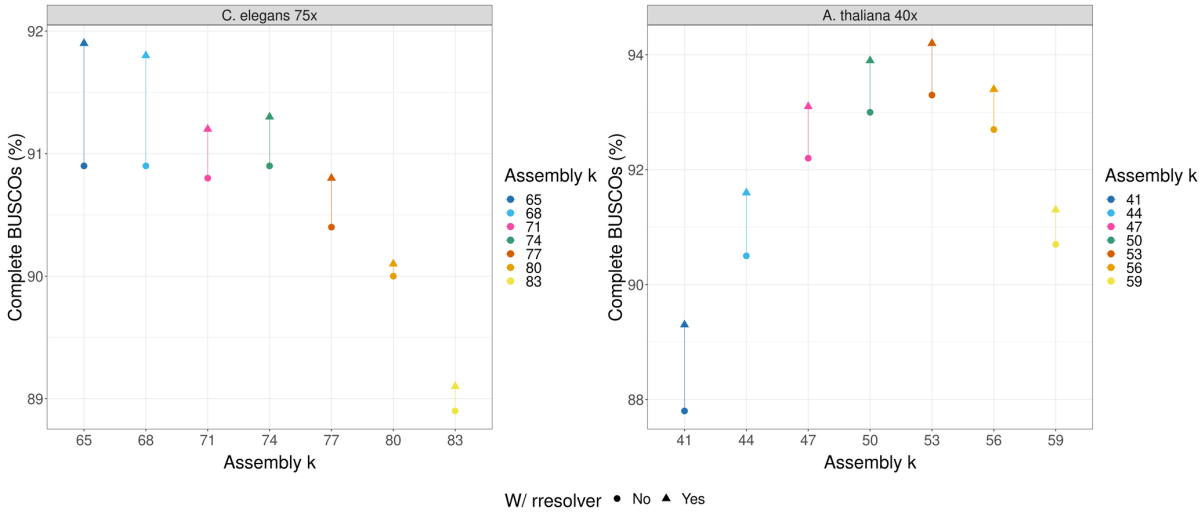

Figure 2.12: Similar to Figure 2.11, instead showing BUSCO results. Assemblies from both datasets show a clear improvement.

the increase is not significant.

Along with the improved assembly quality, the average RResolver run time was only around 3% of the whole ABySS pipeline, with the slowest run time reaching 8%. For *H. sapiens* runs, the RResolver step took between 17 and 70 minutes. The distribution of all assembly run times can be found in Figure 2.13, and machine specifications in Table 2.1.

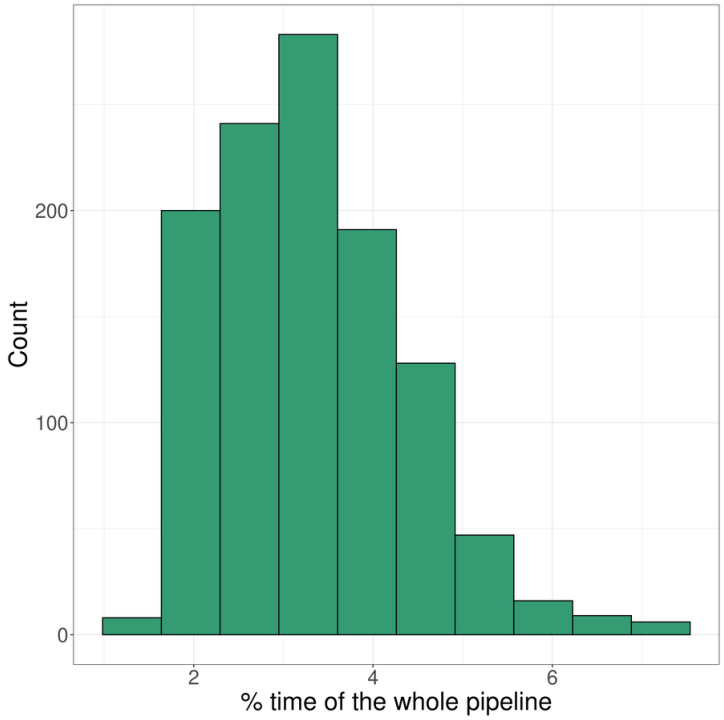

Figure 2.13: Distribution of RResolver run times, in % of the total ABySS pipeline run time.

Table 2.1: Machine specifications for run time benchmarking.

Species	CPU	RAM	OS
C. elegans *A. thaliana* *H. sapiens*	48 x Intel Xeon E5 -2650 @ 2.20 GHz	380 GB	CentOS 6

2.2 Methods

All assemblies were performed using ABySS v2.2.4, with the ABySS+RResolver assemblies having RResolver in the assembly pipeline. For assembly evaluation, QUAST v5.0.2 and BUSCO v5.beta were used. ABySS, QUAST, and BUSCO parameters used can be found in Table 2.2.

The $k_{rresolver}$ value used is either specified or automatically calculated using the heuristic: $k_{rresolver} = k_{assembly} + 45bp$ as described in Results section. Additionally, $k_{rresolver}$ value has an upper limit of 137 and 186 for 150bp and 250bp reads respectively. These numbers were obtained empirically by looking at the relationship of $k_{rresolver}$ value and resulting assembly

Table 2.2: ABySS, QUAST, and BUSCO parameters used to assemble and assess tested genomes. N/A denotes no parameteres were supplied. The $-k$ parameter for ABySS is omitted, as a sweep of values was done.

Species		ABySS	QUAST	BUSCO
C. elegans		-kc2 -B2G -H4 -j48	N/A	-m genome -l nematoda_odb10
A. thaliana		-kc2 -B3G -H4 -j48	N/A	-m genome -l eudicots_odb10
H. sapiens	(NA12878)	-kc2	--large	-m genome -l primates_odb10
	(NA24631)	-kc2		
	(NA24143)	-kc2		
	(NA24385)	-kc3		
		-B40G -H4 -j48		

quality. For read sizes other than 150bp and 250bp, the value is extrapolated with a linear fit:

$$limit = readsize \cdot a + b \tag{2.1}$$

and if we substitute the values for 150bp and 250bp read sizes:

$$\begin{aligned} 137 &= 150 \cdot a + b \\ 186 &= 250 \cdot a + b \end{aligned} \tag{2.2}$$

We get the values $a = 0.49$ and $b = 63.5$, and thus, $limit = readsize \cdot 0.49 + 63.5$.

One of the reasons for an upper limit on the $k_{rresolver}$ value is the short-read base quality trend, which tends to drop sharply towards the read's 3' end [21]. This can be seen in the output of FastQC [7] for NA24631 (150bp) and NA24143 (250bp) in Figure 2.14. For 150bp reads, Phred quality [24] starts noticeably dropping in the 130-140bp range, whereas for 250bp that happens in the 180-190bp range, providing support for the $k_{rresolver}$ upper limit heuristic previously obtained from assembly quality results.

To consider a repeat for path evaluation, its length must be:

$$L_{repeat} \leq k_{rresolver} - (tests - 1) - 2 \cdot margin \tag{2.3}$$

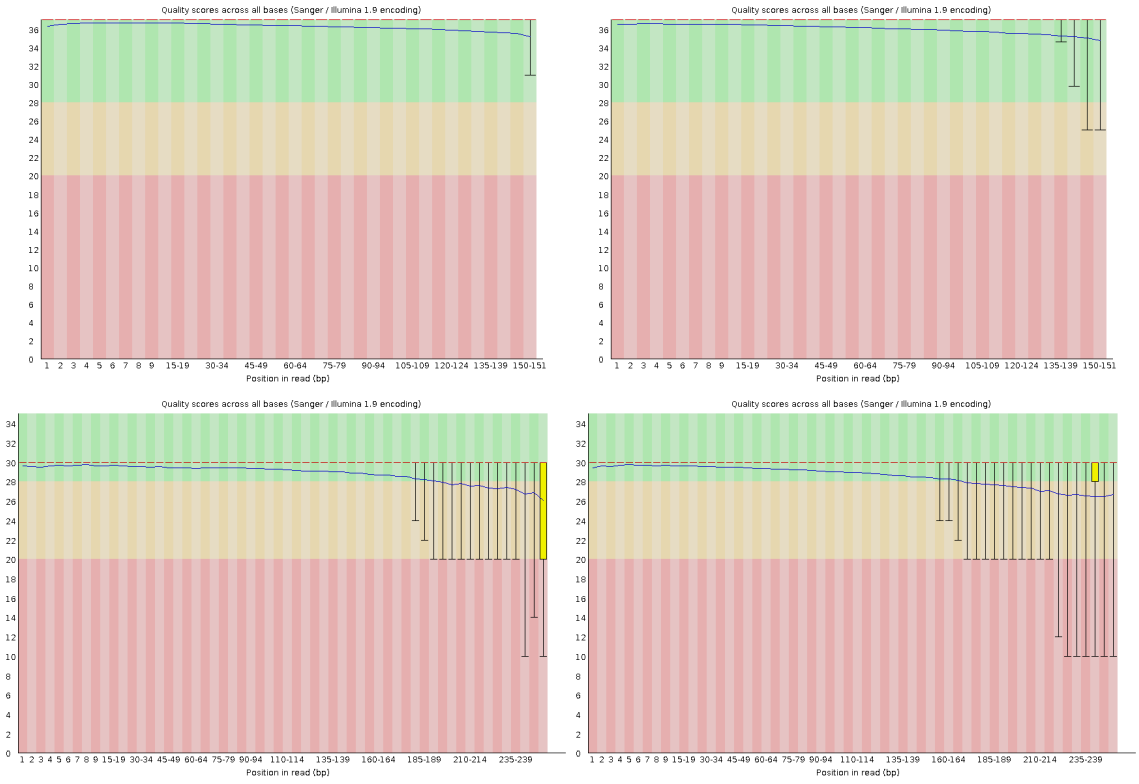

Figure 2.14: The output of FastQC for the *H. sapiens* NA24631 (2x150bp) and NA24143 (2x250bp) reads in first and second row respectively. The common pattern of base quality distribution can be seen — high quality bases from the start with a sharp drop at the end of the read.

where L_{repeat} is the repeat length, $tests$ the required number of tests (sliding window moves), and $margin$ the minimum number of bases the sliding window should touch on adjacent nodes at all times (2 by default).

The formula for the number of required tests is:

$$tests = min(max(m, s \cdot f + t), M) \tag{2.4}$$

where m is the minimum number of tests (20), M maximum number of tests (36), s the approximated space between subsequent reads, f inaccuracy correction factor (4), and t support threshold (4). All numbers shown in parentheses are the default values, and are tunable through

runtime parameters. If we substitute in the values used, we get:

$$tests = min(max(20, s \cdot 4 + 4), 36) \tag{2.5}$$

The inaccuracy correction factor compensates for the errors of coverage estimation. Parameter s is calculated from the $k_{assembly}$ k-mer coverage.

To determine the $k_{assembly}$ coverage of a path in a repeat, the coverages of the incoming node, the repeat node, and the outgoing node are normalized by the formula:

$$Cov_{kassembly} = \frac{\overline{Cov}_{kassembly}}{L - k_{assembly} + 1} \tag{2.6}$$

where $\overline{Cov}_{kassembly}$ is the sum of $k_{assembly}$ k-mer multiplicities of each k-mer in the node, provided by the assembler (as is the case with ABySS), and L the length of the node. This gives us an average multiplicity of each k-mer of a node. Among the three nodes, the lowest coverage is taken to be the coverage of the path, as the higher coverage of other nodes can be explained by different paths going through them.

Then, the read coverage is calculated. Since there can be multiple read lengths in the input dataset and each read length would be an iteration with its own $k_{rresolver}$, the contribution of each read length to node coverage is calculated. This contribution is $l - k_{assembly} + 1$ (with l being the read length) and adjusted for the proportion of those reads in the dataset. The expected normalized number of reads along the path for the current iteration is calculated as:

$$Cov_{read} = \frac{Cov_{kassembly}}{l - k_{assembly} + 1} \cdot p \tag{2.7}$$

where p is the proportion of the reads in the input dataset of read length corresponding to the current $k_{rresolver}$.

Knowing the number of reads along the path and the path length, the approximate number of bases between the start of each read is obtained and from there the number of sliding window

moves. Moving the sliding window number of times equal to this spacing should, on average, find one read on a true path. In practice, the number of moves is multiplied by a factor (4 by default) to offset any inaccuracies of the coverage approximation formula and to make sure all the extracted $k_{rresolver}$ k-mers from the read are found.

To consider a path supported, a threshold of 4 $k_{rresolver}$ k-mers is used. Additionally, 4 $k_{rresolver}$ k-mers are extracter per read, starting from the 5' end. The number of hash functions per $k_{rresolver}$ k-mers when inserting into the Bloom filter is seven by default.

When dealing with complex repeats (Figure 2.3), a maximum of 75 paths are allowed on either side of the repeat for a maximum total of 5625 path combinations. In case there are more than 75, the paths are randomly subsampled down to 75.

Two iterations of graph path evaluation and resolution are done per read size, as the path evaluation completes very quickly and can uncover additional opportunities for repeat resolution.

2.3 Experimental Data

We used six datasets in our experiments, assessing performance for different genome lengths and complexities, and for evaluating proper parameter choice. We used a *C. elegans* N2 strain dataset, with 2x110bp paired end Illumina reads with 75-fold coverage. *A. thaliana* 2x101bp paired end Illumina reads with 40-fold coverage. Four *H. sapiens* datasets — two 2x150bp Illumina (NA12878, NA24631), and two 2x250bp Illumina (NA24143, NA24385) paired end datasets were used with 45-, 43-, 48-, and 58-fold coverage respectively. The *H. sapiens* reference used for reference-based assessment was GRCh38.

C. elegans Illumina PE 2x110bp: DRR008444 https://www.ncbi.nlm.nih.gov/sra/?term=DRR008444

C. elegans reference: ftp://ftp.ncbi.nlm.nih.gov/genomes/archive/old_refseq/Caenorhabditis_elegans

A. thaliana Illumina PE 2x101bp: SAMD00000601 https://www.ebi.ac.uk/ena/browser/view/

SAMD00000601

A. thaliana reference: https://www.ncbi.nlm.nih.gov/assembly/GCF_000001735.4

H. sapiens (NA12878) Illumina PE 2x150bp: ERR3685389 https://www.ebi.ac.uk/ena/browser/view/ERR3685389

H. sapiens (NA24631) Illumina PE 2x150bp: ERR3687419 https://www.ebi.ac.uk/ena/browser/view/ERR3687419

H. sapiens (NA24385) Illumina PE 2x250bp: SRR11321732 https://trace.ncbi.nlm.nih.gov/Traces/sra/?run=SRR11321732

H. sapiens (NA24143) Illumina PE 2x250bp: SRR11321730 https://trace.ncbi.nlm.nih.gov/Traces/sra/?run=SRR11321730

H. sapiens reference: GRCh38 ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/001/405/GCA_000001405.15_GRCh38/seqs_for_alignment_pipelines.ucsc_ids

Source code can be downloaded from: https://github.com/bcgsc/abyss/tree/master/RResolver. The ABySS release with the RResolver algorithm used in the results can be downloaded from: https://github.com/bcgsc/abyss/releases/tag/rresolver-release.

2.4 Error correction

As briefly discussed in Varying coverage section, read errors negatively affect path support estimation. In this section, an experimental approach is assessed that attempts to perform error correction. This error correction method can take into account at most one substitution error per $k_{rresolver}$ k-mer. The method, however, did not improve assembly quality and is presented here to show what may initially seem like a way to improve the RResolver algorithm.

For each $k_{rresolver}$ k-mer extracted from reads, in addition to being inserted in the Bloom filter, a set of spaced seed [35] hash values are calculated so that the union of seed zeroes cover all k-mer bases. These spaced seeds are then inserted into a secondary Bloom filter. Every time the sliding window is moved along a path, first, the non-spaced seeds, primary Bloom filter is queried for the presence of the k-mer. If found, the k-mer is counted towards the support of

the path and the window moves foward one base, as is the case without error correction. If, however, the k-mer is not found, error correction is attempted as this could be due to a read error. The set of spaced seeds' hash values are calculated for the k-mer and queried against the secondary Bloom filter. If this results in a hit, the algorithm takes the wildcard positions of the matching spaced seed as possible positions of where the error may have occurred. Then, one by one, bases in the $k_{rresolver}$ k-mer at wildcard positions are substituted for other bases and each time the $k_{rresolver}$ k-mer is queried against the primary Bloom filter. If this results in a hit, then the last performed substitution is considered to have been an error in a read and the k-mer is counted towards the path support. If no substitution results in a primary Bloom filter hit, it is considered a genuine absence of the k-mer and the window moves one base forward, repeating the whole process.

There are three potential pitfalls for the error correction method — not correcting a sufficient number of errors, producing too many false positives from the Bloom filter queries, and producing too many false positives from repeats in the genome.

To show that the approach of correcting for one error per k-mer is sufficient, the number of recovered reads is plotted. Figure 2.15 shows the percent of reads with no errors, with and without error correction, if all bases had the same quality score for two common short-read lengths. To determine the number of recovered reads, the following formulae are used:

$$\begin{aligned} R_0 &= (1-e)^l \\ R_1 &= (1-e)^l + \binom{l}{1} \cdot e \cdot (1-e)^{l-1} \end{aligned} \tag{2.8}$$

where R_0 is the fraction of reads with no errors, R_1 fraction of reads with with no errors after correcting one base, e chance of a base error, based on Phred quality score of all bases in a read, and l length of the reads. If all bases in a read have at least 30 Phred score, well above 90% of reads can be recovered as errorless after correction. Given that $k_{rresolver}$ k-mers are limited to lengths of about 137bp and 186bp for 150bp and 250bp reads respectively, their base quality is generally 30 and above according to Figure 2.14. This shows that correcting for a single error

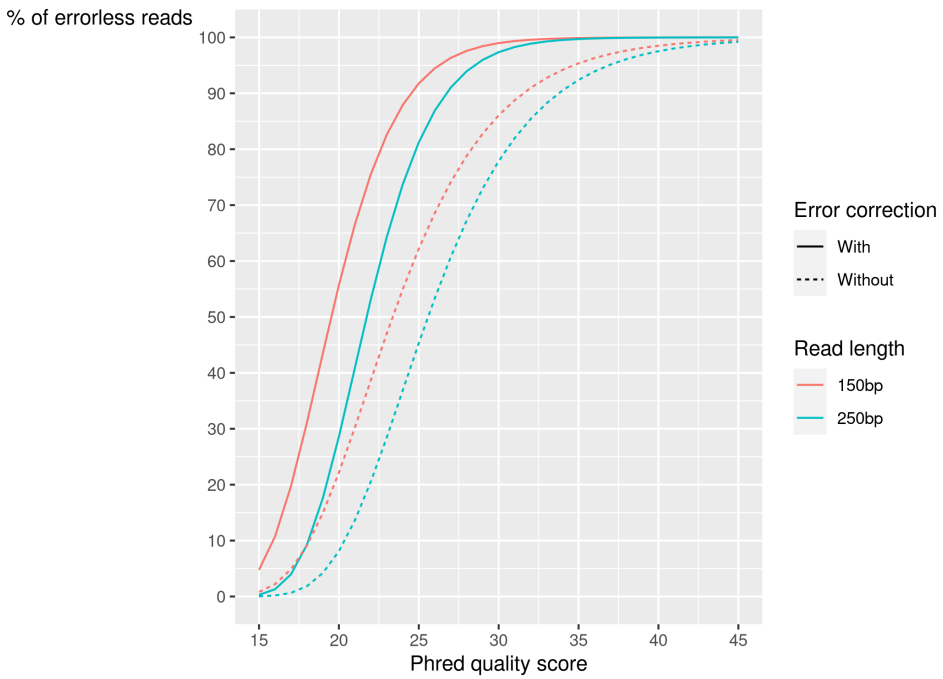

Figure 2.15: The plot shows how many reads would be errorless if all of their bases had the same base quality, with and without error correction.

per k-mer leaves only a small fraction of erroneous k-mers.

As every sliding window move involves multiple queries to the Bloom filters, it needs to be ensured that these queries do not produce unmanageable number of false positives. The first step involves querying the primary Bloom filter, where the FPR can be calculated as:

$$FPR_{primary} = o^h \tag{2.9}$$

where o is the occupancy rate of the Bloom filter, i.e., the fraction of 1 bits across the bit array, and h is the number of hash functions. For the secondary Bloom filter, the FPR is:

$$FPR_{secondary} = 1 - (1 - o^h)^s \tag{2.10}$$

where s is the number of spaced seeds. This gives the probability that at least one spaced seed will be a false positive hit. For the whole process of querying for the presence of a k-mer, the

FPR is then:

$$FPR = FPR_{primary} + (1 - FPR_{primary}) \cdot FPR_{secondary} \cdot FPR_{sub} \tag{2.11}$$

where FPR_{sub} is the probability that any of the wildcard substitutions will give a false positive, calculated as:

$$FPR_{sub} = 1 - (1 - FPR_{primary})^{3 \cdot \frac{k_{rresolver}}{s}} \tag{2.12}$$

The power term here is the number of substitutions, 3 per each of the wildcard positions which are evenly distributed among spaced seeds, i.e., $\frac{k_{rresolver}}{s}$ positions.

To show that the reduced gains with error correction enabled are coming from matches with similar sequences in the genome, a simulated *C. elegans* dataset was generated with DWGSIM [3]. The reads generated from *C. elegans* reference were errorless (DWGIM parameters: `-e 0 -E 0 -C 50 -1 150 -2 150 -r 0 -R 0 -X 0 -y 0`) and RResolver was given 300GiB Bloom filter memory budget to minimize the chance of false positive matches. Two assemblies were done, one with and the other without error correction, with total k-mer query FPR value of $FPR = 5.06e-20\%$ for error correction and $FPR_{primary} = 1.17e-24\%$ for no correction. The error correction approach split the 300GiB memory budget into 65GiB for the primary Bloom filter and 235GiB for the secondary. ABySS parameters used were `-k 117 -kc 2 -B2G -H4`, which gives the optimal assembly for the choice of k and kc. $k_{rresolver}$ was selected with the previosuly obtained heuristic of $k_{rresolver} = k_{assembly} + 45$, limited by the read size, resulting in $k_{rresolver} = 147$. A set of 6 spaced seeds were used with 5 hash values each. The results are provided in Table 2.3. With no read errors and Bloom filter false positives almost completely eliminated, there is still a large discrepancy between the contiguity of the assemblies with and without error correction. Even correcting a single base per $k_{rresolver}$ k-mer was sufficient to have similar sequences from elsewhere in the genome give false support and intefere with the method.

Table 2.3: *C. elegans* simulated dataset assemblies, with and without error correction.

Method	NGA50	Misassemblies
ABySS only	84629	234
RResolver with error correction	86780	229
RResolver without error correction	90809	224

2.5 Visualization software

The figures and visualizations in this work have been generated using the ggplot2 [72] package of the R programming language, the seaborn [68] package of the Python programming language, and Google Drawings[1].

[1] https://docs.google.com/drawings

Chapter 3

Conclusion

3.1 Summary

Generating high quality *de novo* assemblies is crucial for any downstream process. Better assemblies can greatly benefit various clinical applications and have found use in oncological projects [31]. Gene annotation can only go so far if the draft assembly being annotated is of limited quality [54], further emphasizing the point. However, improving *de novo* genome assemblies still has ways to go, as sequencing errors and repetitive sequences are major obstacles to achieving accurate assemblies [20].

Resolving repeats in the assembly graphs has been a widely researched topic. For DBGs, one way in which this has been achieved is using multiple k-mer sizes. The smaller sizes ensure connectivity in the graph whereas the larger sizes resolve repeats and untangle the graph. The current state-of-the-art methods have used multiple k-mer sizes, but only for smaller genomes, leaving a gap in the methodology. The studies so far have not addressed the scalability issues of their methods when dealing with large genomes. The concept of a multisized DBG, as used in the SPAdes assembler, relies on using multiple k values (i.e., k-mer sizes) to build the graph. This requires constructing contigs for each k value, which is prohibitively slow for large genomes. Another approach, as employed by the IDBA assembler, is to make small k increments, making the exploration of a larger range of k values difficult and costly.

There are a number of challenges that come with attempting to use multiple k values approach scalably — high memory usage, long execution times, complex repeats with a large number of possible paths, and errors. The work presented here addresses these challenges and the gap in the methodology, expanding upon the ways in which short-read range information can be used to the fullest extent. In addition to the k value used by DBG, RResolver uses only one additional, larger k value in order to resolve repeats. This is different from the previous approaches of processing a list of k values and is a key enabler of scalability of the algorithm.

In this work, we have demonstrated a method for improving the quality of *de novo* genome assemblies from short-reads by utilizing unused range information. The presented algorithm, RResolver, resolves repeats in a DBG by storing large k-mers in a Bloom filter to estimate graph path support and remove unsupported paths. We have shown that the method consistently increases the contiguity of the assemblies and recovers fragmented or missing genes.

RResolver works seamlessly with the ABySS assembler pipeline, without requiring user involvement. When enabled, the output assembly benefits from higher quality. In this work, RResolver was tested on *C. elegans*, *A. thaliana*, and *H. sapiens* genomes to assess performance on different genome sizes and complexities. Its execution time is only a fraction of the ABySS assembler pipeline it is a part of. We reported that on average RResolver takes 3% of the whole pipeline run time. However, since the output is a simplified graph, the downstream stages benefit from faster run times and so the total run time does not necessarily increase. The ABySS assembler was designed to work on large genomes, and so working within similar run time constraints is important.

RResolver adds one more piece of the puzzle to delivering high quality *de novo* assemblies of large genomes and does so at the early stages of the assembly, benefiting any downstream algorithms that build contigs, scaffold the assembly, or do final polishing.

3.2 Future work

The RResolver algorithm improves assembly contiguity and gene recovery, but does introduce misassemblies in some cases. This may warrant additional investigation in order to find the sources of error and attempt to avoid them. Some additional misassemblies are expected. They can come from genuine structural variations between the assembled individual and the reference. By simplifying the assembly graph, more can be done in downstream assembly stages which also increases chances of misassembly. And, of course, RResolver making wrong decisions directly introduces misassemblies.

Sources of errors come from either a false path being found supported or a genuine path being found unsupported. For the former, the primary source of such error would be Bloom filter false positives. As indicated in Methods and findings chapter, these are addressed and are generally found in few numbers.

Missing support for a genuine path poses a bigger problem. Due to read errors and uncertain coverage, $k_{rresolver}$ k-mers may be missed and a path wrongly found unsupported. If a node merge occurs in this situation, due to a missing genuine path, false node paths may be merged and produce a misassembly. Uncertain coverage could be improved with higher granularity of $k_{assembly}$ k-mer coverage at the assembler level. Alternatively, more conservative approach in long nodes where coverage is more uncertain may help as well.

Bibliography

[1] https://www.genome.gov/human-genome-project/Timeline-of-Events, Accessed December 1, 2020. → page 1

[2] DISCOVAR: Assemble genomes, find variants. https://www.broadinstitute.org/software/discovar/blog. Accessed December 8, 2020. → page 10

[3] DWGSIM whole genome simulator based off of wgsim found in SAMtools. https://github.com/nh13/DWGSIM. Accessed 19 January, 2021. → page 41

[4] Seqtk, a fast and lightweight tool for processing sequences in the FASTA or FASTQ format. https://github.com/lh3/seqtk. Accessed 13 January, 2021. → page 30

[5] An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414): 57–74, Sept. 2012. doi:10.1038/nature11247. → page 4

[6] M. D. Adams. The genome sequence of drosophila melanogaster. *Science*, 287(5461): 2185–2195, Mar. 2000. doi:10.1126/science.287.5461.2185. → page 1

[7] S. Andrews, F. Krueger, A. Segonds-Pichon, L. Biggins, C. Krueger, and S. Wingett. FastQC. Babraham Institute, Jan. 2010. → pages 8, 34

[8] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, A. V. Pyshkin, A. V. Sirotkin, N. Vyahhi, G. Tesler, M. A. Alekseyev, and P. A. Pevzner. SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, May 2012. doi:10.1089/cmb.2012.0021. → pages 10, 12

[9] V. Bansal and C. Boucher. Sequencing technologies and analyses: Where have we been and where are we going? *iScience*, 18:37–41, Aug. 2019. doi:10.1016/j.isci.2019.06.035. → page 12

[10] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970. doi:10.1145/362686.362692. → pages 10, 11, 18

[11] K. R. Bradnam, J. N. Fass, A. Alexandrov, P. Baranay, M. Bechner, I. Birol, S. Boisvert, J. A. Chapman, G. Chapuis, R. Chikhi, H. Chitsaz, W.-C. Chou, J. Corbeil, C. D. Fabbro, T. R. Docking, R. Durbin, D. Earl, S. Emrich, P. Fedotov, N. A. Fonseca, G. Ganapathy, R. A. Gibbs, S. Gnerre, É. Godzaridis, S. Goldstein, M. Haimel, G. Hall, D. Haussler, J. B. Hiatt, I. Y. Ho, J. Howard, M. Hunt, S. D. Jackman, D. B. Jaffe, E. D. Jarvis, H. Jiang, S. Kazakov, P. J. Kersey, J. O. Kitzman, J. R. Knight, S. Koren, T.-W. Lam, D. Lavenier, F. Laviolette, Y. Li, Z. Li, B. Liu, Y. Liu, R. Luo, I. MacCallum, M. D. MacManes, N. Maillet, S. Melnikov, D. Naquin, Z. Ning, T. D. Otto, B. Paten, O. S. Paulo, A. M. Phillippy, F. Pina-Martins, M. Place, D. Przybylski, X. Qin, C. Qu, F. J. Ribeiro, S. Richards, D. S. Rokhsar, J. G. Ruby, S. Scalabrin, M. C. Schatz, D. C. Schwartz, A. Sergushichev, T. Sharpe, T. I. Shaw, J. Shendure, Y. Shi, J. T. Simpson, H. Song, F. Tsarev, F. Vezzi, R. Vicedomini, B. M. Vieira, J. Wang, K. C. Worley, S. Yin, S.-M. Yiu, J. Yuan, G. Zhang, H. Zhang, S. Zhou, and I. F. Korf. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2(1), July 2013. doi:10.1186/2047-217x-2-10. → page 8

[12] B. L. Cantarel, I. Korf, S. M. Robb, G. Parra, E. Ross, B. Moore, C. Holt, A. S. Alvarado, and M. Yandell. MAKER: An easy-to-use annotation pipeline designed for emerging model organism genomes. *Genome Research*, 18(1):188–196, Nov. 2007. doi:10.1101/gr.6743907. → page 4

[13] D. Chalopin, M. Naville, F. Plard, D. Galiana, and J.-N. Volff. Comparative analysis of transposable elements highlights mobilome diversity and evolution in vertebrates. *Genome Biology and Evolution*, 7(2):567–580, Jan. 2015. doi:10.1093/gbe/evv005. → page 12

[14] R. Chikhi and G. Rizk. Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology*, 8(1), Jan. 2013. doi:10.1186/1748-7188-8-22. → pages 10, 11

[15] C.-S. Chin, P. Peluso, F. J. Sedlazeck, M. Nattestad, G. T. Concepcion, A. Clum, C. Dunn, R. O'Malley, R. Figueroa-Balderas, A. Morales-Cruz, G. R. Cramer, M. Delledonne, C. Luo, J. R. Ecker, D. Cantu, D. R. Rank, and M. C. Schatz. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature Methods*, 13(12):1050–1054, Oct. 2016. doi:10.1038/nmeth.4035. → page 3

[16] D. M. Church, L. Goodstadt, L. W. Hillier, M. C. Zody, S. Goldstein, X. She, C. J. Bult, R. Agarwala, J. L. Cherry, M. DiCuccio, W. Hlavina, Y. Kapustin, P. Meric, D. Maglott, Z. Birtle, A. C. Marques, T. Graves, S. Zhou, B. Teague, K. Potamousis, C. Churas, M. Place, J. Herschleb, R. Runnheim, D. Forrest, J. Amos-Landgraf, D. C. Schwartz, Z. Cheng, K. Lindblad-Toh, E. E. Eichler, and C. P. P. and. Lineage-specific biology revealed by a finished genome assembly of the mouse. *PLoS Biology*, 7(5):e1000112, May 2009. doi:10.1371/journal.pbio.1000112. → page 9

[17] S. C. Clark, R. Egan, P. I. Frazier, and Z. Wang. ALE: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome

assemblies. *Bioinformatics*, 29(4):435–443, Jan. 2013. doi:10.1093/bioinformatics/bts723. → page 5

[18] P. Das, L. Sahoo, S. P. Das, A. Bit, C. G. Joshi, B. Kushwaha, D. Kumar, T. M. Shah, A. T. Hinsu, N. Patel, S. Patnaik, S. Agarwal, M. Pandey, S. Srivastava, P. K. Meher, P. Jayasankar, P. G. Koringa, N. S. Nagpure, R. Kumar, M. Singh, M. A. Iquebal, S. Jaiswal, N. Kumar, M. Raza, K. D. Mahapatra, and J. Jena. De novo assembly and genome-wide SNP discovery in rohu carp, labeo rohita. *Frontiers in Genetics*, 11, Apr. 2020. doi:10.3389/fgene.2020.00386. → page 2

[19] A. P. J. de Koning, W. Gu, T. A. Castoe, M. A. Batzer, and D. D. Pollock. Repetitive elements may comprise over two-thirds of the human genome. *PLoS Genetics*, 7(12): e1002384, Dec. 2011. doi:10.1371/journal.pgen.1002384. → page 12

[20] J. F. Denton, J. Lugo-Martinez, A. E. Tucker, D. R. Schrider, W. C. Warren, and M. W. Hahn. Extensive error in the number of genes inferred from draft genome assemblies. *PLoS Computational Biology*, 10(12):e1003998, Dec. 2014. doi:10.1371/journal.pcbi.1003998. → page 43

[21] J. C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. *Nucleic Acids Research*, 36(16):e105–e105, Aug. 2008. doi:10.1093/nar/gkn425. → pages 8, 20, 34

[22] D. Earl, K. Bradnam, J. S. John, A. Darling, D. Lin, J. Fass, H. O. K. Yu, V. Buffalo, D. R. Zerbino, M. Diekhans, N. Nguyen, P. N. Ariyaratne, W.-K. Sung, Z. Ning, M. Haimel, J. T. Simpson, N. A. Fonseca, I. Birol, T. R. Docking, I. Y. Ho, D. S. Rokhsar, R. Chikhi, D. Lavenier, G. Chapuis, D. Naquin, N. Maillet, M. C. Schatz, D. R. Kelley, A. M. Phillippy, S. Koren, S.-P. Yang, W. Wu, W.-C. Chou, A. Srivastava, T. I. Shaw, J. G. Ruby, P. Skewes-Cox, M. Betegon, M. T. Dimon, V. Solovyev, I. Seledtsov, P. Kosarev, D. Vorobyev, R. Ramirez-Gonzalez, R. Leggett, D. MacLean, F. Xia, R. Luo, Z. Li, Y. Xie, B. Liu, S. Gnerre, I. MacCallum, D. Przybylski, F. J. Ribeiro, S. Yin, T. Sharpe, G. Hall, P. J. Kersey, R. Durbin, S. D. Jackman, J. A. Chapman, X. Huang, J. L. DeRisi, M. Caccamo, Y. Li, D. B. Jaffe, R. E. Green, D. Haussler, I. Korf, and B. Paten. Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Genome Research*, 21(12):2224–2241, Sept. 2011. doi:10.1101/gr.126599.111. → pages 4, 8

[23] R. Ekblom, L. Smeds, and H. Ellegren. Patterns of sequencing coverage bias revealed by ultra-deep sequencing of vertebrate mitochondria. *BMC Genomics*, 15(1):467, 2014. doi:10.1186/1471-2164-15-467. → pages 12, 20

[24] B. Ewing, L. Hillier, M. C. Wendl, and P. Green. Base-calling of automated sequencer traces UsingPhred. i. accuracy assessment. *Genome Research*, 8(3):175–185, Mar. 1998. doi:10.1101/gr.8.3.175. → pages 8, 34

[25] S. Fitz-Gibbon, A. L. Hipp, K. K. Pham, P. S. Manos, and V. L. Sork. Phylogenomic inferences from reference-mapped and de novo assembled short-read sequence data

using RADseq sequencing of california white oaks (quercus section quercus). *Genome*, 60(9):743–755, Sept. 2017. doi:10.1139/gen-2016-0202. → page 2

[26] A. Goffeau, B. G. Barrell, H. Bussey, R. W. Davis, B. Dujon, H. Feldmann, F. Galibert, J. D. Hoheisel, C. Jacq, M. Johnston, E. J. Louis, H. W. Mewes, Y. Murakami, P. Philippsen, H. Tettelin, and S. G. Oliver. Life with 6000 genes. *Science*, 274(5287): 546–567, Oct. 1996. doi:10.1126/science.274.5287.546. → page 1

[27] A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, Feb. 2013. doi:10.1093/bioinformatics/btt086. → pages 5, 25

[28] M. Hunt, T. Kikuchi, M. Sanders, C. Newbold, M. Berriman, and T. D. Otto. REAPR: a universal tool for genome assembly evaluation. *Genome Biology*, 14(5):R47, 2013. doi:10.1186/gb-2013-14-5-r47. → page 5

[29] R. M. IDURY and M. S. WATERMAN. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2(2):291–306, Jan. 1995. doi:10.1089/cmb.1995.2.291. → page 2

[30] S. D. Jackman, B. P. Vandervalk, H. Mohamadi, J. Chu, S. Yeo, S. A. Hammond, G. Jahesh, H. Khan, L. Coombe, R. L. Warren, and I. Birol. ABySS 2.0: resource-efficient assembly of large genomes using a bloom filter. *Genome Research*, 27 (5):768–777, Feb. 2017. doi:10.1101/gr.214346.116. → pages 10, 11, 13, 16, 25

[31] F. Jamshidi, E. Pleasance, Y. Li, Y. Shen, K. Kasaian, R. Corbett, P. Eirew, A. Lum, P. Pandoh, Y. Zhao, J. E. Schein, R. A. Moore, R. Rassekh, D. G. Huntsman, M. Knowling, H. Lim, D. J. Renouf, S. J. Jones, M. A. Marra, T. O. Nielsen, J. Laskin, and S. Yip. Diagnostic value of next-generation sequencing in an unusual sphenoid tumor. *The Oncologist*, 19(6):623–630, May 2014. doi:10.1634/theoncologist.2013-0390. → pages 2, 43

[32] A. J. Jeffreys, V. Wilson, and S. L. Thein. Hypervariable 'minisatellite' regions in human DNA. *Nature*, 314(6006):67–73, Mar. 1985. doi:10.1038/314067a0. → page 12

[33] E. W. M. Jr. A history of DNA sequence assembly. *it - Information Technology*, 58(3), Jan. 2016. doi:10.1515/itit-2015-0047. → pages 1, 2

[34] J. D. Kececioglu and E. W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1-2):7–51, Feb. 1995. doi:10.1007/bf01188580. → page 2

[35] U. Keich, M. Li, B. Ma, and J. Tromp. On spaced seeds for similarity search. *Discrete Applied Mathematics*, 138(3):253–263, Apr. 2004. doi:10.1016/s0166-218x(03)00382-2. → page 38

[36] M. G. Kidwell. *Genetica*, 115(1):49–63, 2002. doi:10.1023/a:1016072014259. → page 12

[37] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy. Canu: scalable and accurate long-read assembly via adaptivek-mer weighting and repeat separation. *Genome Research*, 27(5):722–736, Mar. 2017. doi:10.1101/gr.215087.116. → page 3

[38] J. Laskin, S. Jones, S. Aparicio, S. Chia, C. Ch'ng, R. Deyell, P. Eirew, A. Fok, K. Gelmon, C. Ho, D. Huntsman, M. Jones, K. Kasaian, A. Karsan, S. Leelakumari, Y. Li, H. Lim, Y. Ma, C. Mar, M. Martin, R. Moore, A. Mungall, K. Mungall, E. Pleasance, S. R. Rassekh, D. Renouf, Y. Shen, J. Schein, K. Schrader, S. Sun, A. Tinker, E. Zhao, S. Yip, and M. A. Marra. Lessons learned from the application of whole-genome analysis to the treatment of patients with advanced cancers. *Molecular Case Studies*, 1(1):a000570, Sept. 2015. doi:10.1101/mcs.a000570. → page 3

[39] D. Li, C.-M. Liu, R. Luo, K. Sadakane, and T.-W. Lam. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, 31(10):1674–1676, Jan. 2015. doi:10.1093/bioinformatics/btv033. → page 12

[40] R. Li, W. Fan, G. Tian, H. Zhu, L. He, J. Cai, Q. Huang, Q. Cai, B. Li, Y. Bai, Z. Zhang, Y. Zhang, W. Wang, J. Li, F. Wei, H. Li, M. Jian, J. Li, Z. Zhang, R. Nielsen, D. Li, W. Gu, Z. Yang, Z. Xuan, O. A. Ryder, F. C.-C. Leung, Y. Zhou, J. Cao, X. Sun, Y. Fu, X. Fang, X. Guo, B. Wang, R. Hou, F. Shen, B. Mu, P. Ni, R. Lin, W. Qian, G. Wang, C. Yu, W. Nie, J. Wang, Z. Wu, H. Liang, J. Min, Q. Wu, S. Cheng, J. Ruan, M. Wang, Z. Shi, M. Wen, B. Liu, X. Ren, H. Zheng, D. Dong, K. Cook, G. Shan, H. Zhang, C. Kosiol, X. Xie, Z. Lu, H. Zheng, Y. Li, C. C. Steiner, T. T.-Y. Lam, S. Lin, Q. Zhang, G. Li, J. Tian, T. Gong, H. Liu, D. Zhang, L. Fang, C. Ye, J. Zhang, W. Hu, A. Xu, Y. Ren, G. Zhang, M. W. Bruford, Q. Li, L. Ma, Y. Guo, N. An, Y. Hu, Y. Zheng, Y. Shi, Z. Li, Q. Liu, Y. Chen, J. Zhao, N. Qu, S. Zhao, F. Tian, X. Wang, H. Wang, L. Xu, X. Liu, T. Vinar, Y. Wang, T.-W. Lam, S.-M. Yiu, S. Liu, H. Zhang, D. Li, Y. Huang, X. Wang, G. Yang, Z. Jiang, J. Wang, N. Qin, L. Li, J. Li, L. Bolund, K. Kristiansen, G. K.-S. Wong, M. Olson, X. Zhang, S. Li, H. Yang, J. Wang, and J. Wang. The sequence and de novo assembly of the giant panda genome. *Nature*, 463(7279): 311–317, Dec. 2009. doi:10.1038/nature08696. → page 9

[41] K. Lindblad-Toh, , C. M. Wade, T. S. Mikkelsen, E. K. Karlsson, D. B. Jaffe, M. Kamal, M. Clamp, J. L. Chang, E. J. Kulbokas, M. C. Zody, E. Mauceli, X. Xie, M. Breen, R. K. Wayne, E. A. Ostrander, C. P. Ponting, F. Galibert, D. R. Smith, P. J. deJong, E. Kirkness, P. Alvarez, T. Biagi, W. Brockman, J. Butler, C.-W. Chin, A. Cook, J. Cuff, M. J. Daly, D. DeCaprio, S. Gnerre, M. Grabherr, M. Kellis, M. Kleber, C. Bardeleben, L. Goodstadt, A. Heger, C. Hitte, L. Kim, K.-P. Koepfli, H. G. Parker, J. P. Pollinger, S. M. J. Searle, N. B. Sutter, R. Thomas, C. Webber, and E. S. Lander. Genome sequence, comparative analysis and haplotype structure of the domestic dog. *Nature*, 438(7069):803–819, Dec. 2005. doi:10.1038/nature04338. → page 4

[42] M. Litt and J. A. Luty. A hypervariable microsatellite revealed by in vitro amplification of a dinucleotide repeat within the cardiac muscle actin gene. *Am J Hum Genet*, 44(3): 397–401, Mar 1989. → page 12

[43] R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, G. He, Y. Chen, Q. Pan, Y. Liu, J. Tang, G. Wu, H. Zhang, Y. Shi, Y. Liu, C. Yu, B. Wang, Y. Lu, C. Han, D. W. Cheung, S.-M. Yiu, S. Peng, Z. Xiaoqian, G. Liu, X. Liao, Y. Li, H. Yang, J. Wang, T.-W. Lam, and J. Wang. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, 1(1), Dec. 2012. doi:10.1186/2047-217x-1-18. → pages 10, 12

[44] V. Mäkinen, L. Salmela, and J. Ylinen. Normalized n50 assembly metric using gap-restricted co-linear chaining. *BMC Bioinformatics*, 13(1), Oct. 2012. doi:10.1186/1471-2105-13-255. → page 5

[45] H. Mohamadi, J. Chu, B. P. Vandervalk, and I. Birol. ntHash: recursive nucleotide hashing. *Bioinformatics*, page btw397, July 2016. doi:10.1093/bioinformatics/btw397. → page 18

[46] E. W. MYERS. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, Jan. 1995. doi:10.1089/cmb.1995.2.275. → page 2

[47] E. W. Myers. A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204, Mar. 2000. doi:10.1126/science.287.5461.2196. → page 2

[48] E. W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(Suppl 2): ii79–ii85, Sept. 2005. doi:10.1093/bioinformatics/bti1114. → page 2

[49] R. M. Nowak, J. P. Jastrzębski, W. Kuśmirek, R. Sałamatin, M. Rydzanicz, A. Sobczyk-Kopcioł, A. Sulima-Celińska, Ł. Paukszto, K. G. Makowczenko, R. Płoski, V. V. Tkach, K. Basałaj, and D. Młocicki. Hybrid de novo whole-genome assembly and annotation of the model tapeworm hymenolepis diminuta. *Scientific Data*, 6(1), Dec. 2019. doi:10.1038/s41597-019-0311-3. → page 3

[50] G. Parra, K. Bradnam, Z. Ning, T. Keane, and I. Korf. Assessing the gene space in draft genomes. *Nucleic Acids Research*, 37(1):289–297, Nov. 2008. doi:10.1093/nar/gkn916. → page 7

[51] H. Peltola, H. Söderlund, and E. Ukkonen. SEQAID: a DNA sequence assembling program based on a mathematical model. *Nucleic Acids Research*, 12(1Part1):307–321, 1984. doi:10.1093/nar/12.1part1.307. → page 2

[52] Y. Peng, H. C. M. Leung, S. M. Yiu, and F. Y. L. Chin. IDBA – a practical iterative de bruijn graph de novo assembler. In *Lecture Notes in Computer Science*, pages 426–440. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-12683-3_28. → pages 12, 13

[53] P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17): 9748–9753, Aug. 2001. doi:10.1073/pnas.171285098. → pages 2, 10

[54] S. L. Salzberg. Next-generation genome annotation: we still struggle to get it right. *Genome Biology*, 20(1), May 2019. doi:10.1186/s13059-019-1715-2. → pages 4, 43

[55] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts, G. Marcais, M. Pop, and J. A. Yorke. GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome Research*, 22(3):557–567, Jan. 2012. doi:10.1101/gr.131383.111. → pages 5, 8

[56] F. Sanger, S. Nicklen, and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467, Dec. 1977. doi:10.1073/pnas.74.12.5463. → page 1

[57] F. Sanger, A. Coulson, G. Hong, D. Hill, and G. Petersen. Nucleotide sequence of bacteriophage λ DNA. *Journal of Molecular Biology*, 162(4):729–773, Dec. 1982. doi:10.1016/0022-2836(82)90546-0. → page 1

[58] F. A. Simão, R. M. Waterhouse, P. Ioannidis, E. V. Kriventseva, and E. M. Zdobnov. BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, 31(19):3210–3212, June 2015. doi:10.1093/bioinformatics/btv351. → pages 7, 25

[59] J. T. Simpson. Exploring genome characteristics and sequence quality without a reference. *Bioinformatics*, 30(9):1228–1235, Jan. 2014. doi:10.1093/bioinformatics/btu023. → page 8

[60] J. T. Simpson and R. Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Research*, 22(3):549–556, Dec. 2011. doi:10.1101/gr.126953.111. → page 2

[61] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, Feb. 2009. doi:10.1101/gr.089532.108. → page 10

[62] R. Staden. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Research*, 6(7):2601–2610, 1979. doi:10.1093/nar/6.7.2601. → page 2

[63] B. P. Vandervalk, C. Yang, Z. Xue, K. Raghavan, J. Chu, H. Mohamadi, S. D. Jackman, R. Chiu, R. L. Warren, and I. Birol. Konnector v2.0: pseudo-long reads from paired-end sequencing data. *BMC Medical Genomics*, 8(S3), Sept. 2015. doi:10.1186/1755-8794-8-s3-s1. → pages 2, 11

[64] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, J. D. Gocayne, P. Amanatides, R. M. Ballew, D. H. Huson, J. R. Wortman, Q. Zhang, C. D. Kodira, X. H. Zheng, L. Chen, M. Skupski, G. Subramanian, P. D. Thomas, J. Zhang, G. L. G. Miklos, C. Nelson, S. Broder, A. G. Clark, J. Nadeau, V. A. McKusick, N. Zinder, A. J. Levine, R. J. Roberts, M. Simon, C. Slayman, M. Hunkapiller, R. Bolanos, A. Delcher, I. Dew, D. Fasulo, M. Flanigan, L. Florea, A. Halpern, S. Hannenhalli, S. Kravitz, S. Levy, C. Mobarry, K. Reinert, K. Remington, J. Abu-Threideh, E. Beasley, K. Biddick, V. Bonazzi, R. Brandon, M. Cargill, I. Chandramouliswaran, R. Charlab, K. Chaturvedi, Z. Deng, V. D. Francesco, P. Dunn, K. Eilbeck, C. Evangelista, A. E. Gabrielian,

W. Gan, W. Ge, F. Gong, Z. Gu, P. Guan, T. J. Heiman, M. E. Higgins, R.-R. Ji, Z. Ke, K. A. Ketchum, Z. Lai, Y. Lei, Z. Li, J. Li, Y. Liang, X. Lin, F. Lu, G. V. Merkulov, N. Milshina, H. M. Moore, A. K. Naik, V. A. Narayan, B. Neelam, D. Nusskern, D. B. Rusch, S. Salzberg, W. Shao, B. Shue, J. Sun, Z. Y. Wang, A. Wang, X. Wang, J. Wang, M.-H. Wei, R. Wides, C. Xiao, C. Yan, A. Yao, J. Ye, M. Zhan, W. Zhang, H. Zhang, Q. Zhao, L. Zheng, F. Zhong, W. Zhong, S. C. Zhu, S. Zhao, D. Gilbert, S. Baumhueter, G. Spier, C. Carter, A. Cravchik, T. Woodage, F. Ali, H. An, A. Awe, D. Baldwin, H. Baden, M. Barnstead, I. Barrow, K. Beeson, D. Busam, A. Carver, A. Center, M. L. Cheng, L. Curry, S. Danaher, L. Davenport, R. Desilets, S. Dietz, K. Dodson, L. Doup, S. Ferriera, N. Garg, A. Gluecksmann, B. Hart, J. Haynes, C. Haynes, C. Heiner, S. Hladun, D. Hostin, J. Houck, T. Howland, C. Ibegwam, J. Johnson, F. Kalush, L. Kline, S. Koduru, A. Love, F. Mann, D. May, S. McCawley, T. McIntosh, I. McMullen, M. Moy, L. Moy, B. Murphy, K. Nelson, C. Pfannkoch, E. Pratts, V. Puri, H. Qureshi, M. Reardon, R. Rodriguez, Y.-H. Rogers, D. Romblad, B. Ruhfel, R. Scott, C. Sitter, M. Smallwood, E. Stewart, R. Strong, E. Suh, R. Thomas, N. N. Tint, S. Tse, C. Vech, G. Wang, J. Wetter, S. Williams, M. Williams, S. Windsor, E. Winn-Deen, K. Wolfe, J. Zaveri, K. Zaveri, J. F. Abril, R. Guigó, M. J. Campbell, K. V. Sjolander, B. Karlak, A. Kejariwal, H. Mi, B. Lazareva, T. Hatton, A. Narechania, K. Diemer, A. Muruganujan, N. Guo, S. Sato, V. Bafna, S. Istrail, R. Lippert, R. Schwartz, B. Walenz, S. Yooseph, D. Allen, A. Basu, J. Baxendale, L. Blick, M. Caminha, J. Carnes-Stine, P. Caulk, Y.-H. Chiang, M. Coyne, C. Dahlke, A. D. Mays, M. Dombroski, M. Donnelly, D. Ely, S. Esparham, C. Fosler, H. Gire, S. Glanowski, K. Glasser, A. Glodek, M. Gorokhov, K. Graham, B. Gropman, M. Harris, J. Heil, S. Henderson, J. Hoover, D. Jennings, C. Jordan, J. Jordan, J. Kasha, L. Kagan, C. Kraft, A. Levitsky, M. Lewis, X. Liu, J. Lopez, D. Ma, W. Majoros, J. McDaniel, S. Murphy, M. Newman, T. Nguyen, N. Nguyen, M. Nodell, S. Pan, J. Peck, M. Peterson, W. Rowe, R. Sanders, J. Scott, M. Simpson, T. Smith, A. Sprague, T. Stockwell, R. Turner, E. Venter, M. Wang, M. Wen, D. Wu, M. Wu, A. Xia, A. Zandieh, and X. Zhu. The sequence of the human genome. *Science*, 291(5507):1304–1351, Feb. 2001. doi:10.1126/science.1058040. → pages 1, 2

[65] G. Vergnaud. Minisatellites: Mutability and genome architecture. *Genome Research*, 10 (7):899–907, July 2000. doi:10.1101/gr.10.7.899. → page 12

[66] R. L. Warren, C. I. Keeling, M. M. S. Yuen, A. Raymond, G. A. Taylor, B. P. Vandervalk, H. Mohamadi, D. Paulino, R. Chiu, S. D. Jackman, G. Robertson, C. Yang, B. Boyle, M. Hoffmann, D. Weigel, D. R. Nelson, C. Ritland, N. Isabel, B. Jaquish, A. Yanchuk, J. Bousquet, S. J. M. Jones, J. MacKay, I. Birol, and J. Bohlmann. Improved white spruce (picea glauca) genome assemblies and annotation of large gene families of conifer terpenoid and phenolic defense metabolism. *The Plant Journal*, 83 (2):189–212, June 2015. doi:10.1111/tpj.12886. → page 2

[67] R. L. Warren, C. Yang, B. P. Vandervalk, B. Behsaz, A. Lagman, S. J. M. Jones, and I. Birol. LINKS: Scalable, alignment-free scaffolding of draft genomes with long reads. *GigaScience*, 4(1), Aug. 2015. doi:10.1186/s13742-015-0076-3. → page 11

[68] M. L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi:10.21105/joss.03021. URL https://doi.org/10.21105/joss.03021. → page 42

[69] R. M. Waterhouse, F. Tegenfeldt, J. Li, E. M. Zdobnov, and E. V. Kriventseva. OrthoDB: a hierarchical catalog of animal, fungal and bacterial orthologs. *Nucleic Acids Research*, 41(D1):D358–D365, Nov. 2012. doi:10.1093/nar/gks1116. → page 7

[70] R. Waterston and J. Sulston. The genome of caenorhabditis elegans. *Proceedings of the National Academy of Sciences*, 92(24):10836–10840, Nov. 1995. doi:10.1073/pnas.92.24.10836. → page 1

[71] J. L. Weber and E. W. Myers. Human whole-genome shotgun sequencing. *Genome Research*, 7(5):401–409, May 1997. doi:10.1101/gr.7.5.401. → page 1

[72] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. → page 42

[73] D. R. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–829, Feb. 2008. doi:10.1101/gr.074492.107. → page 10