Statistical Methods for Genome Assembly

by

Maayan Bresler

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

in

Engineering — Electrical Engineering and Computer Sciences

and the Designated Emphasis

in

Computational and Genomic Biology

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Yun S. Song, Chair
Professor Sandrine Dudoit
Professor Lior Pachter

Spring 2014

Statistical Methods for Genome Assembly

Abstract

Statistical Methods for Genome Assembly

by

Maayan Bresler

Doctor of Philosophy in Engineering — Electrical Engineering and Computer Sciences

and the Designated Emphasis

in

Computational and Genomic Biology

University of California, Berkeley

Professor Yun S. Song, Chair

In the last decade, sequencing technology has progressed rapidly, leading to much faster and cheaper production of short-read data. The challenge of *assembling* the reads into an accurate reconstruction of the sequenced genome, however, has increased. This is because the assembly problem is made more difficult when the reads are shorter, especially for genomes of most higher organisms, which contain complicated repeat structures. In this thesis we study the algorithmic problem of *de novo* DNA sequence assembly, focusing on the challenge of dealing with genomic repeats. We develop two new assembly tools, as well as initiate the study of information-theoretic limits of shotgun sequencing for realistic genomes.

Our first novel algorithm for DNA assembly, Telescoper, is designed for assembly of telomeres. Due to their many repeats, telomeric regions are notoriously difficult to assemble. Telescoper iteratively extends long paths through a series of read-overlap graphs and evaluates them based on a statistical framework. The algorithm utilizes both short and long-insert libraries in an integrated way throughout the assembly process. This approach is shown to effectively resolve some of the complex repeat structures found in the telomeres of yeast genomes.

Our second novel algorithm for DNA assembly, Piper, takes a statistical approach to resolving ambiguity caused by repeats. A lot of potentially useful information is present in paired-end reads, but due to the inherent noise in the insert length and the combinatorial nature of the problem, it is not clear how to best use this information. Piper selects a *set* of candidate paths through the contig-graph, and scores them based on their likelihood given a generative model for the reads. The output consists of a ranked set of assemblies (rather than a single assembly) in order to give the maximum information available, while still explicitly encoding unresolved ambiguity. On small simulated datasets, Piper produces excellent error-free assemblies.

In the final portion of the thesis, we investigate the information-theoretic limits of DNA sequencing, focusing on the effect of repeats. Specifically, we ask: how many reads of a given length are necessary in order to perfectly reconstruct with a certain target probability? We focus on a simple read model, with noiseless single-end reads, but consider arbitrary genomic sequences. We first prove a lower bound on the read length and the coverage depth required for reconstruction in terms of the repeat statistics of the genome. Building on known algorithms, we design a de Brujin graph based assembly algorithm which can achieve very close to the lower bound for repeat statistics of a wide range of sequenced genomes. The results are based on a set of necessary and sufficient conditions for reconstruction that depend on the DNA sequence and the reads.

To my twin brother, Guy, and our parents, Liora and Yoram.

Contents

List of Figures

List of Tables

Acknowledgments

First, I can't imagine a more dedicated adviser than Yun. Yun invests his whole self in his students, embracing us as colleagues and friends, sacrificing sleep, and sharing, with infinite generosity, his considerable technical wisdom and life advice. He's passionate about sharing his love of research with us, and is always there to direct us towards success.

I feel incredibly lucky to have had the opportunity to work with David Tse, and to witness how he approaches research problems with curiosity, somehow transforming them into meaning and order. I'm grateful beyond words to all the rest of my collaborators; these are the peers I have been privileged to watch most closely, to depend upon most, and from whom to learn the most in my time here. In the course of our work, I developed enormous admiration for my collaborator-friends Sara Sheehan, Peter Jin, and Ravi Pandya, whose kindness, positivity, enthusiasm and wisdom made working together a great pleasure. Guy was all of this, and the best brother I can imagine. Andrew Chan kindly introduced me to assembly.

I'm very grateful to my generous dissertation committee members Sandrine Dudoit and Lior Pachter, and to Elchanan Mossel and Jasper Rhine for serving on my quals committee. I'm grateful for helpful conversations with Nikolaj Bjorner on Z3 and optimization, and to Oliver Zill and Devin Scannell for discussions on telomeres. I thank David Heckerman and Ravi Pandya for a very enjoyable internship at Microsoft Research. I'll always be grateful to Mung Chiang, for an incredibly positive experience doing undergraduate research.

I'm grateful to so many of the inspiring colleagues and friends I met along the way; for the moral support or advice of those in my group and the computational biology designated emphasis, including Anand, Josh, Paul, Jack, Kelley, Wei-Chun, Junming, Jasmine, Matthias, Jonathan, Meromit, Adam, and Shannon; to friends and collaborators in the AMPLab including Jesse, Kristal, Ameet, John, Lester, and Fabian; and to people in other areas of the department including Vidya, Samarth, Urmila, Suman, Gireeja, Erin, Jeremy, Galen, Dapo, Isabelle, and Tselil; and finally, for the friends and house-mates in other departments, including Claire, Elizabeth, Andre, Jeff, Jiayue, Shelly, Sara A., and Sarah D. These people made this place home.

Finally, I'm grateful to my parents Liora and Yoram.

Chapter 1

Introduction

Genome sequencing is a basic tool in biology, used to create reference genomes for new species and to characterize whole-genome variations in multiple individuals of a population. Sequencing was once hugely expensive (estimates of the cost of the Human Genome Project run upwards of $3 billion), but advances in high-throughput sequencing (HTS) technologies have completely reshaped the scene [Check Hayden, 2014]. The cost of generating raw genome sequence data has fallen so much that the $1,000 genome – "full coverage human genomes for less than $1,000" – has actually happened recently [Check Hayden, 2014].

The drop in sequencing cost means that a staggeringly large number of genomes will be sequenced in coming years. Recently undertaken sequencing projects include i5k to sequence 5,000 insect genomes [Arthropod Genomic Consortium, 2014], Genome 10k to sequence thousands of vertebrates [Genome 10K Community of Scientists, 2009], more than 12,000 sequencing projects listed at JGI [2014], and the 3 Million Genomes project to sequence all the DNA of 1 million people, 1 million microorganisms, and 1 million plants and animals announced by sequencing giant BGI (the Beijing Genomics Institute) [Diehl, 2013].

The generation of all this data is only half the story; in order to obtain an accurate whole genome assembly, the data must be assembled. At the highest level, each chromosome can be thought of as a single long string of base pairs (A, C, T, G). The data produced by Next Generation Sequencers (NGS) is a set of substrings, generated at random from the genome, referred to as *reads*. The task of genome assembly is to piece together reads to reconstruct the original genome sequence.

Reads are obtained using various technologies, with varying read length, error rates, and coverage. Sanger-chemistry reads range in length from around 500 to 1500 bp. Illumina reads range from 100 to 250 bp, and cost significantly less, for much higher throughput. Illumina currently occupies the majority of the sequencing market share. Other short-read technologies include Complete Genomics [Drmanac et al., 2010], Helicos [Harris et al., 2008], 454 Life Sciences [Margulies et al., 2005], SOLiD [McKernan et al., 2009], and Ion Torrent [Rothberg et al., 2011]. On the other end of the length spectrum, longer than Sanger reads, are PacBio and Moleculo, both of which offer lower coverage than standard short-read technology, for a higher price, but are capable of generating reads up to 10 kbp long that can be valuable in resolving similar regions [Huddleston et al., 2014], [Voskoboynik et al., 2013].

Initial improvements in assembly from short-read data focused on how to process the sheer quantity of data and how to detect overlaps and correct for sequencing error. The de Bruijn graph proved a useful data structure for this purpose [Pevzner et al., 2001], and is used by pioneering short-read assemblers such as Velvet [Zerbino and Birney, 2008] and EULER-USR [Chaisson et al., 2008], and many subsequent assemblers.

The length of the reads and the amount of overlap between them result in fundamental limitations in the completeness and accuracy of the assembly, as we will discuss in more detail in Chapter 2. In particular, if two consecutive reads along the genome fail to have sufficient overlap, then it typically cannot be determined that they should be adjacent. Similarly, if the genome has *repeats*, i.e. substrings that occur in multiple places in the genome, that are longer than the overlap between adjacent reads, then again, usually the reads cannot be uniquely ordered [Pevzner, 1995, Ukkonen, 1992]. The result is that assembly in repetitive regions or low-coverage regions is poor.

The effect of short read-lengths can be somewhat mitigated through the production of *paired reads*. After DNA fragmentation, both ends of each fragment are sequenced, yielding a pair of reads separated by an unsequenced gap whose length is given by the fragment size and is approximately known. Depending on the library construction method, these are referred to as either paired-end or mate-pair reads, but we'll use the term read-pairs to be inclusive. The paired nature of these reads constitutes a powerful source of information, making genome assembly possible in situations where it is infeasible using just unpaired-reads of the same length or coverage parameters. However, employing paired reads adds an additional computational and algorithmic challenge to the assembly problem.

The difficulty of the problem has led to a large number of genome assembly tools. An incomplete list of genome assembly tools for short-read data includes SOAPdenovo [Li et al., 2010], ALLPATHS 2 [MacCallum et al., 2009], ABySS [Simpson et al., 2009], Cortex [Iqbal et al., 2012], ALLPATHS-LG [Gnerre et al., 2011], Discovar [Computational Research and Development Group, Broad Institute of MIT and Harvard, 2014], the PE-Assembler [Ariyaratne and Sung, 2011], the Paired de Bruijn Graph [Medvedev et al., 2011], SPADES [Bankevich et al., 2012], and IDBA [Peng et al., 2010]. And yet, despite the number of tools available, we are a ways from being able to produce high-quality finished genome sequences.

Current assemblies are more fragmented, more incomplete, and more full of errors than desirable. Salzberg and Yorke [2005] found that for every draft genome they examined, there were hundreds to thousands of regions where sequences were incorrectly joined. Alkan et al. [2011] reported that a *de novo* shotgun assembly of the human genome using short-reads is 16% shorter than the reference assembled using much more expensive sequencing approaches, and that less than 1% of segmental duplications are represented. Treangen and Salzberg [2011] review different repeat types that pose challenges to assembly. More recently, Baker [2012] offers a review of the field, the bottom line of which is that genome assemblies produced using short-read technology should be regarded with care, as drafts.

There are major efforts to make order of the complicated state of assembly. The Assemblathon competitions [Bradnam et al., 2013, Earl et al., 2011] are intended to assess current state-of-the-art methods in genome assembly, and to explore the question of what metrics should be used to evaluate assemblies. The GAGE evaluation [Salzberg et al., 2012] is a another effort to test multiple assemblers on the same data sets, and generates an automated reference-based report of assembly quality. These efforts indicate the community's increasing emphasis on accurate assembly, and should help streamline assembly assessment, enabling a focus on the development of good algorithms.

The repetitive regions that abound in the genome are not just computationally challenging, but also biologically significant. Segmental duplications are continuous portions of DNA that map to two or more genomic locations, sometimes referred to as Low Copy Repeats [Bailey and Eichler, 2006]. Typically, the term is used to refer to duplications longer than 1–5kb with at least 90% sequence identify. Recent interchromosomal exchanges, which happen most frequently in subtelomeric regions, have even higher sequence identity, often exceeding 99% identify. Segmental duplications are prevalent; in total, they account for 5–6% of the genome. In recent years, they've gained recognition for being biologically important as well, as they're now believed to have played important roles in primate evolution and new gene creation, as well as in determining disease susceptibility [Bailey and Eichler, 2006].

1.1 Contributions of this dissertation

In this thesis, we examine the problem of genome assembly from several different perspectives. First, we begin by developing an algorithm, Telescoper, that is designed for long segmental duplications such as those at the yeast telomere. Telescoper extends the assembly in an iterative fashion, exploring multiple branches of conceivable solutions, and choosing among them using statistical scores. It uses paired-reads from multiple libraries with different fragment sizes to validate the assembly as it generates it. Although the current implementation of the algorithm is more of a proof-of-concept than a fully-functional assembler, it suggests an interesting algorithmic approach with the potential to aid repetitive regions. This chapter is based on joint work with Sara Sheehan, Andrew Chan, and Yun S. Song [Bresler et al., 2012].

In the following chapter, we take a second crack at the realistic-data genome assembly problem, again zooming in only on repetitive regions that thwart other algorithms. This time we make several key changes. Whereas Telescoper strives for perfect reconstruction, this time we try to more explicitly annotate uncertainty. In many instances the data are not sufficient to yield perfect reconstruction; in this case, in order to include the true solution, while still recognizing the uncertainty of the assembled output, we produce an enumeration of multiple high-likelihood assemblies along with their likelihoods. We use a preliminary approximation to the likelihood to find candidate assemblies, and then compute a likelihood for each of the candidate assemblies using the assembly likelihood score from Rahman and Pachter [2013]. Encouragingly, we find that in every instance in which the dataset is small enough for the current implementation of our algorithm to run, it includes as output the true solution, with a likelihood no worse than any of the generated alternatives. For the output to be easily interpreted and compared with more traditional assembly outputs, we also produce an encoding of the greatest common substrings of the assemblies, which is consistent with the more familiar FASTA or FASTG assembly output format.

Our algorithm, Piper, takes as input an assembly graph produced by another program. On small datasets of reads simulated from human segmental duplications, and using an error-free input assembly graph, our algorithm improves upon the assembly, and produces more accurate solutions than the other assembly algorithms against which we compare. Future work will involve developing more heuristics to speed up the runtime and enable us to increase the size of the problem on which Piper can be run, and include evaluation on real data rather than simulated data. This chapter is based on joint work with Peter Jin, Ravi Pandya, and Yun S. Song [Bresler et al., 2014].

Finally, we step back from realistic data, to ask a more fundamental question. Algorithms aside, under what circumstances are *the data* sufficient for perfect assembly? Under what circumstances is it impossible? We approach this question from an information-theoretic perspective; given data parameters of read-length and coverage depth, what features of a genome are sufficient to compute the probability of perfect reconstruction? We derive a lower bound on the read length and the coverage depth necessary for perfect reconstruction in terms of the repeat statistics of the genome. We then design a simple de Brujin graph based assembly algorithm, building on earlier works. Using a set of necessary and sufficient conditions for reconstruction that depend on the DNA sequence and on the reads, we show that for the repeat statistics of a wide range of sequenced genomes, the simple proposed algorithm requires only slightly longer and more numerous reads than the lower bound required for perfect reconstruction. The aforementioned necessary and sufficient conditions can be interpreted as analogues of Ukkonen and Pevzner's conditions for sequencing by hybridization [Pevzner, 1995, Ukkonen, 1992]. This chapter is based on joint work with Guy Bresler and David Tse [Bresler et al., 2013].

1.2 Terminology

Throughout the thesis, we adopt the following terms commonly used to describe the output of sequencing technologies and the resulting assemblies:

- **Read-pair:** a pair of sequenced reads from a fragment. The fragment size determines the distance between the two reads, often called the *insert* size. The insert distribution is frequently approximated by a normal distribution. We use the term read-pair regardless of whether the insert is short or long.

- **Library:** a collection of DNA fragments, from which read-pairs are created. Each library has a distribution on the distance between the reads in a pair, called the insert distribution.

- **Mate:** the partner of a read R in a read-pair. When R is oriented with respect to a sequence, we know its mate's relative position and can refer to it as a *left-mate* or *right-mate* (or, as a *left-read* or *right-read*).

- **Contig:** a sequence, which ideally belongs to the original genome, produced from assembling a group of reads. The standard output from an assembly algorithm is a set of contigs. Contigs are often ordered to produce *scaffolds*, which may contain stretches of unknown sequence between the contigs.

- **Contig graph:** a graph in which each vertex is a contig, and directed edges between contigs represent overlapping sequence; i.e. the last k bases of one contig are the same as the first k bases of its neighbor contig. A contig graph is *perfect* if the genome can be spelled out as a path through the graph.

- **Read-overlap graph:** also called a read graph, is a graph in which each vertex is a read, and directed edges between reads represent overlapping sequence; i.e., in the error-free case, the last k bases of one read are the same as the first k bases of its neighbor read, where k is greater than some threshold.

- **Unitig:** a path through the read graph that can be unambiguously merged into a single sequence. A *unitig graph* is an extension of the read-overlap graph idea (similarly for a unitig path), where the vertices are now unitigs.

- **Coverage:** Coverage is defined as the expected number of reads to cover a particular base in the genome. If the genome has length G, read-length L, and number of reads N, then the coverage is NL/G.

Chapter 2

Assembly Using Iterative Extension with Paired-End Reads

2.1 Introduction

In this chapter, we describe a new algorithm to improve *de novo* assembly of highly repetitive regions. Although the ideas presented here are applicable to the assembly of any genomic region, this algorithm was developed with the specific aim of assembling highly repetitive regions such as telomeres.

Telomeres are particularly complex and repetitive, and thus very difficult to assemble correctly. Not only does each telomere contain repeats within itself, but often telomeres on different chromosomes are very similar. Existing assembly algorithms thus frequently fail to assemble telomeric regions from short-read data. Due to this lack of complete assembly, telomere evolution has not been fully characterized, though a great deal is to be gained from it, as telomeres have been linked to aging [McEachern et al., 2000]. High-quality telomere assemblies could help us learn more about the variation in telomeres within and between species. In addition, characterizing telomere gene families and their regulation could help us clarify the function of telomeres and how they change as we age.

In our method, which we name Telescoper, we incorporate the following three algorithmic ideas, the latter two of which make novel use of read-pairs:

1. **Iterative extensions:** A seed sequence is extended iteratively using reads localized to a particular region by their mates, thus allowing for gradual extension into difficult regions. See Section 2.2 for details. As mentioned above, this idea is not new, but it has not yet been fully exploited in a well-used algorithm, despite several potential advantages.

2. **Simultaneous use of short-insert read-pairs in a statistical framework:** Rather than using read-pair information pair by pair to untangle the read-graph, we build

extensions through the graph and *simultaneously* consider all read-pairs mapping to each extension to choose the most probable extension. See Section 2.2.

3. **Simultaneous use of long-insert libraries:** Rather than using long-insert read-pairs only for scaffolding or for filling in gaps between easily assembled contigs, our iterative extension procedure uses long-insert reads during assembly. We look for support of assembled sequence at all insert sizes, so that incorrect assembly can result only if the repetitive structure spans all libraries. See Section 2.2 for further details.

Each of the above ideas helps to resolve repetitive regions. Implicit throughout our algorithm is the principle that in order to assemble difficult regions, one cannot make only safe simplifications, but must also explore several alternative extensions, and use downstream analysis to find and reject false extensions.

We tested the performance of our method on both real and simulated data from the telomeres of the *Saccharomyces cerevisiae* genome, which consists of 16 chromosomes. This is a particularly challenging problem since all such telomeres have a core repetitive component called X ($\approx$ 475 bp long), as well as several combinatorial repeats and sometimes a larger repetitive component (see Saccharomyces Genome Database, `www.yeastgenome.org`). In addition, because *S. cerevisiae* underwent an ancient genome duplication [Kellis et al., 2004], telomeric regions of different chromosomes typically share highly similar repetitive regions. We show that Telescoper is capable of generating more complete and continuous assemblies in the telomeric regions than other state-of-the-art *de novo* assembly algorithms, especially when longer long-insert libraries are used.

Related work

We compare our performance with a number of other assemblers, but notable exceptions that also focus heavily on incorporating read-pair information include works such as ALLPATHS-LG [Gnerre et al., 2011], the PE-Assembler [Ariyaratne and Sung, 2011], and the Paired de Bruijn Graph [Medvedev et al., 2011].

ALLPATHS-LG requires reads of length around 100 bases sequenced from short fragments of length $\approx$ 180 bp so that, on average, each read-pair overlaps by about 20 bases. This means that in general each read-pair can be merged into a single longer read corresponding to the fragment. A drawback of this approach is in the very specific type of data required, which differs from the standard library construction of fragments 300 $\sim$ 500 bp in size. The PE-Assembler builds short stretches in non-repetitive regions first, similar to unitigs (see Section 1.2 for a definition) in a de Bruijn graph, and then extends these iteratively using reads with mates that map to the increasing already-assembled portion. (A similar idea is also used in Parrish et al. [2011] for resequencing with a reference, where insertions are assembled as iterative extensions of existing sequences.) The Paired de Bruijn Graph method entails building a so-called A-Bruijn graph in which vertices track pairs of reads instead of single reads, with two vertices being merged only if the merging is consistent with the associated pairs of reads. To our knowledge, this method remains largely theoretical at this time, and it has been tested only on simulated data with perfect reads.

A theoretical observation from Medvedev et al. [2011] is that longer long-insert libraries can substantially improve assembly. Recent innovations [Peng et al., 2012] in library construction may bring such libraries into the mainstream, so it is timely to develop algorithms that take full advantage of such data.

2.2 Methods

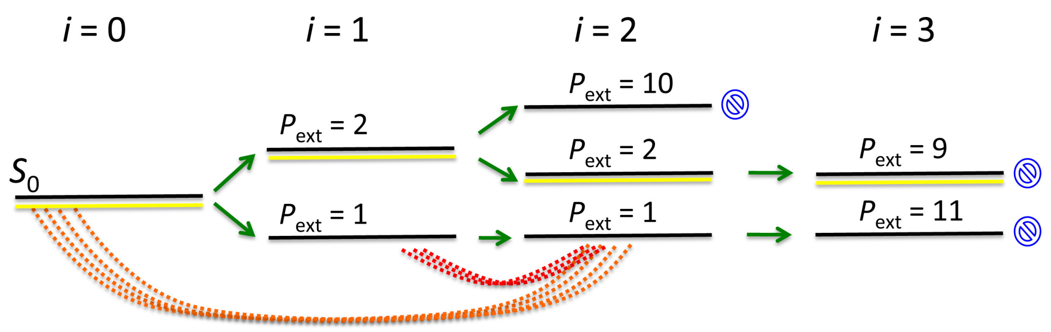
We have two main aims in our algorithm: (i) rather than performing a greedy read-by-read assembly procedure, we build a number of alternative extensions, and score them according to the alignment of read-pairs to each extension, and (ii) we use long-insert read-pairs not only for scaffolding or gap filling, but also as part of the assembly itself, to check that the local assembly is consistent on a longer scale.

Our algorithm begins with a set of non-repetitive *seed strings*, as could be taken from a reference genome, if it exists, or be assembled from a de Bruijn graph. At present, we use seeds of length 500 bp from the reference, at position 40 kb from the end of the chromosome, although contigs produced from any other algorithm could be used. The goal is to then independently extend each contig to produce a more complete assembly.

A high-level overview of the algorithm is illustrated in Figure 2.1. The algorithm proceeds by extending the end of the contig iteratively by a fixed amount, N_{new}, per iteration, as detailed in Section 2.2. We fix the extension length (usually a few hundred bases) as a conservative measure. Because multiple extensions are frequently possible, the result is an "extension graph" (*e-graph*) in which each extension node (*e-node*) contains N_{new} bases of new sequence that serve as a possible extension for that *e-node*'s parent. A path from the root (the seed string) to a leaf represents a series of extensions that form a single lengthened contig. The aim is for the *e-graph* to contain a path corresponding to the true sequence, ideally terminating close to the end of the desired chromosome, and for this path to be identifiable as the best.

Our algorithm will be most tractable if the *e-graph* is sparse, so at each iteration there are as few extensions as possible (and the true extension is among them). The criteria for pruning and terminating the *e-graph* are discussed in Section 2.2. We first explain our methods for (i) listing possible extensions for a given *e-node* in the *e-graph*, (ii) scoring each extension based on the alignment of short-insert read-pairs, and (iii) scoring each extension based on the alignment of long-insert read-pairs.

In the following description, we assume without loss of generality that we are extending to the right.



```
Build an e-graph of possible extensions:
    Begin with $S_0$, the seed e-node.
    For each depth i in e-graph, for each e-node:
        1) find e-node's possible extensions (green arrow)
        2) score extension using short-insert library (red arcs)
        3) update extension's score $P_{ext}$ using long-insert library (orange arcs)
        4) decide whether extension should be abandoned (⊘ symbol)
Read off contigs:
    Choose best path of e-nodes (yellow underline on path)
```

Figure 2.1: High-level description of the algorithm. Beginning with the seed string S_0, the algorithm iteratively performs the steps described to construct an *e-graph* data structure, from which a contig or contigs can be read. For simplicity, only a few example arcs are shown; in reality, red arcs are present between each consecutive pair of *e-nodes*, and orange arcs can be present between a given *e-node* and any of its preceding *e-nodes*.

Iterative extension of assembly

The extension step consists of finding possible extensions of a given *e-node*; the extensions will in turn become *e-nodes* themselves. We fix the length of each *e-node* so that most right-reads in the new extension will have left-mates mapping to the *e-node* rather than behind it. In our implementation, we choose this length, denoted N_{tot}, to be the mean insert length plus the standard deviation of the short-insert library. In the case of multiple short-insert libraries, one can use the largest short-insert length for computing N_{tot}.

The extension step is depicted in Figure 2.2. It begins by mapping all the left-reads to the *e-node* to obtain right-mates extending off the right end of the *e-node* into unknown region yet to be assembled, i.e. the left-mate maps to the *e-node* and the right-mate dangles off the end, as illustrated in Figure 2.2(a). We say that these right-mates form a read "cloud".

The reads in the read cloud are error-corrected, then used to construct a read-overlap graph, which is transformed into a unitig graph as depicted in Figure 2.2(b). More details on error-correction and read-overlap graph construction are provided in the Supplementary

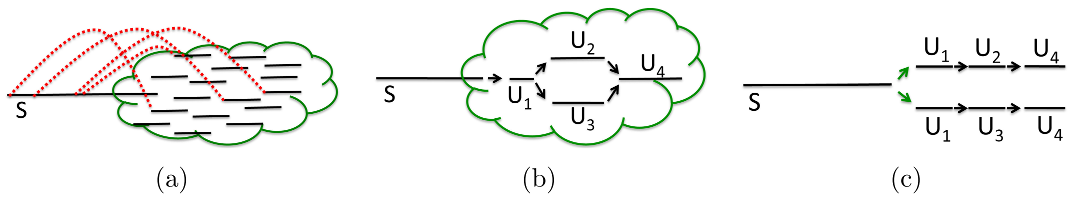


Figure 2.2: Illustration of step 1 of Figure 2.1, finding an *e-node* S's possible extensions. (a) A read "cloud" consists of those right-reads with left-mates that map to S. (b) The reads in the cloud are then error corrected and organized into a read-graph, which is in turn converted into a unitig graph. (c) Paths through the unitig graph correspond to possible extensions.

Material. The unitig graph encodes a list of candidate extensions for the contig, as illustrated in Figure 2.2(c). Each new *e-node* consists of N_{new} bases of new extension plus $(N_{\text{tot}} - N_{\text{new}})$ bases from the end of the previous *e-node*.

There are several advantages to this localized assembly. First, it reduces ambiguities caused by repeats. For a read-pair from another location to interfere with the area under construction, its left-read must map to the previous *e-node* while the right-read must overlap with another read in the read cloud. Second, because it restricts assembly to a small region, there is ample memory to store complicated information about the reads and their relationships. This information can be thrown out as we move to other regions of the graph. This local use of information enables more complex use of read-pairs, as described in the rest of this section.

Simultaneous use of short-insert read-pairs in statistical scoring of extensions

While existing assembly algorithms make use of read-pairs in various ways, the information contained in read-pairs has not yet been fully exploited. In other assemblers, read-pairs are used primarily to connect unitigs with expected insert sizes. We can obtain additional power by scoring potential extensions according to the features derived from the aligned read-pairs.

We first evaluate extensions based on the likelihood of gaps in short-insert read-pair coverage. Each extension consists of an ordered sequence of unitigs, as in Figure 2.2(c). Each right-read in an assembled unitig will have a left-mate mapping to earlier sequence in the previous *e-node*. The set of left-mates associated with reads in unitig U is denoted M_U; see Figure 2.3(a).

In our model, we make the simplifying assumption of a uniform coverage distribution. Let x denote the distance from the right end of a left-read relative to the start of unitig U, as pictured in Figure 2.3(b). We denote by $f_U(x)$ the expected number of left-reads in M_U spanning position x; see Figure 2.3(c). We compute $f_U(x)$ by convolving the expected insert distribution $h(\cdot)$ with the uniform distribution over the stretch of U on which right-mates

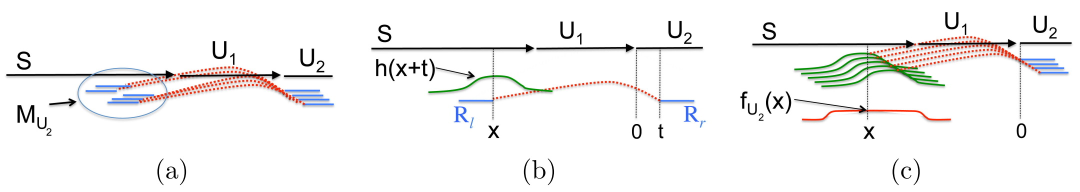


Figure 2.3: Computing the expected number of left-reads mapping back from a unitig U_2 to the previous *e-node* S. (a) M_{U_2} denotes the set of reads mapping from unitig U_2 to the previous *e-node* S. (b) For a right-read R_r located at position t in unitig U_2, the probability of its left-mate R_l mapping to S at a distance x behind U_2 is $h(x+t)$, where $h(\cdot)$ is the expected insert distribution. (c) The expected number of reads at position x behind unitig U_2 is given by $f_U(x)$ defined in (2.1).

can begin:

$$f_U(x) = \sum_{t=0}^{L(U)-\ell} \lambda \cdot h(x+t), \tag{2.1}$$

where $L(U)$ is the length of U, ℓ is the read length, and λ is the probability of a read arriving at position t; note that λ is equal to $C/(2\ell)$, where C is the coverage. False unitigs will typically have gaps in the empirical distribution $\hat{f}_U(x)$, as illustrated in Figure 2.4(b). Let $\mathrm{Gap}(U)$ denote the set of such gaps associated with U. For a gap $g \in \mathrm{Gap}(U)$ of length $\geq \ell/2$, we compute a penalty equal to the number of mates expected in g, obtained by summing $f_U(x)$ over g's coordinates. The preliminary score for an extension is then the sum of these penalties over all gaps and all unitigs in the extension:

$$p_{\mathrm{ext}} = \sum_{U \in \mathrm{extension}} \sum_{g \in \mathrm{Gap}(U)} \sum_{x \in g} f_U(x). \tag{2.2}$$

To produce a final score P_{ext} for each possible extension, we add p_{ext} to a *contig gap penalty*, equal to λ times the largest gap size (denoted by g_c in Figure 2.4(c)) between two adjacent unitigs, i.e. the expected number of reads to fall in that gap. The best extensions (i.e., those with the lowest P_{ext} scores) are kept, as described in more detail in Section 2.2.

Simultaneous use of long-insert libraries

Telescoper utilizes all libraries simultaneously during assembly, rather than using long-insert libraries only during scaffolding or gap-filling, as is typical in other assembly algorithms. The main idea is that once long paths have been formed in the *e-graph*, any further extension can be evaluated on the basis of its agreement with the current *e-graph* according to each library. Having produced and pruned a set of extensions using just the short-insert library in steps 1 and 2 of our algorithm (see Figure 2.1), the third step aims to confirm that a proposed extension is supported by read-pair information from all other libraries simultaneously. For

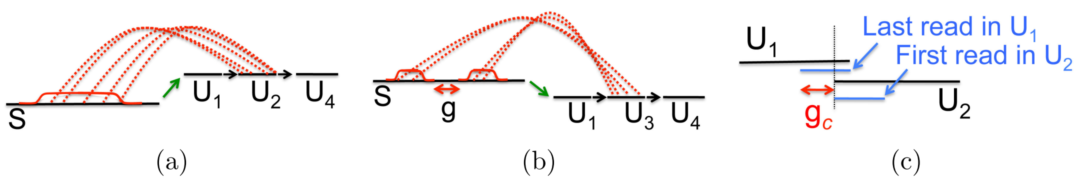


Figure 2.4: Illustration of step 2 of Figure 2.1, scoring an *e-node*'s possible extensions using short-insert read-pairs. (a) The penalty for unitig U_2 is 0 because no gaps of size $\geq \ell/2$ exist (where ℓ is the read length). (b) The penalty for unitig U_3 is > 0 because a gap, denoted g, of size $\geq \ell/2$ exists. (c) The size of contig gap g_c is the distance between the reads that define the end and start of two adjacent unitigs.

there to be ambiguity in extension choice, there must be repeats at lengths corresponding to all library sizes.

To test for long-insert read-pair support of a potential extension, we first gather all read-pairs of which right-reads map to the extension and left-reads map to the previous *e-nodes* in the path up the *e-graph*. Then, if the right-reads fully cover the proposed extension, even possibly without overlaps, we consider the extension to be fully supported. Partial support is computed as a linear function of the fraction of the extension that is covered by the right-reads. This support measure is then multiplied by the short-insert score P_{ext} to obtain a single final score.

Choosing extensions for continuation

For a given *e-node*, upon finding all its possible extensions, at most B top scoring (the lower the better) extensions are retained for computational tractability. In our analysis we use $B = 4$. We create a new *e-node* for each of these top scoring extensions, and assign a running score equal to the sum of its extension score and its parent *e-node*'s running score. Then, at each depth in the *e-node* graph, the B top scoring *e-node*s are marked for pursuit.

An *e-node* is terminated if it cannot be lengthened by the extension operation, if its extension score plus the scores of two previous ancestral extensions exceeds a threshold, or if a specified maximum depth is reached.

To track the parallel success of alternative *e-node* paths and keep their number in check, we use breadth first search to explore the *e-graph*. If two different sequences of *e-nodes* end with equivalent *e-nodes* at a particular depth, we allow the two *e-nodes* to merge. This kind of merging of *e-nodes* reduces the computational burden.

2.3 Empirical Results

In this section, we compare Telescoper's performance with that of other short-read assembly algorithms, including ABySS [Simpson et al., 2009], ALLPATHS 2 [MacCallum et al., 2009], SGA [Simpson and Durbin, 2012], SOAPdenovo [Li et al., 2010], and Velvet [Zerbino and Birney, 2008].

Because of limited space, we focus on short-read data in the ensuing discussion. However, as detailed in the Supplementary Material, we also considered a combination of short-insert short-read data and long-insert Sanger read data, and observed that Telescoper compares favorably with other algorithms, including Celera [Myers, 2000], which was designed for Sanger reads.

Data and experiment setup

We studied the performance on both simulated and real data from strain S288C of *S. cerevisiae*. We obtained a reference genome from Saccharomyces Genome Database (`www.yeastgenome.org`), which was created through extensive, systematic sequencing to produce a very accurate assembly, including the telomeric regions. As mentioned earlier, because of ancient genome duplication and complex yeast telomere structure, the telomeres of different chromosomes typically share highly similar repetitive regions, which poses challenges to assembly.

We considered different types of data to test the robustness of the algorithms and to study the effect of insert distributions on performance:

Simulated Data D1 consisted of read-pairs with two insert distributions, one short and one long. The read length was 101 bp for both types. The *short-insert* reads had coverage depth 100X and an insert distribution with mean 400 bp and variance 75 bp. The *long-insert* reads had coverage depth 20X and an insert distribution with mean 10 kb and variance 1 kb. Simulation details are provided in the Supplementary Material.

Simulated Data D2 consisted of two read-pair data sets with the same insert distributions and coverages as D1, but with a reduced read length of 50 bp.

Real Data D3 consisted of Illumina read-pairs from a sequencing library preparation using Cre-Lox recombination. The reads, as described in Van Nieuwerburgh et al. [2012], were sorted using DeLoxer into reads categorized as *short-insert* ($0 \sim 400$ bp fragments, mean 220 bases) or *long-insert* ($1 \sim 5$ kb, mean 2.3 kb). The reads varied in length from $30 \sim 100$ bp. We truncated reads to 50 bases in order to provide algorithms with high-quality, uniform-length reads. We used coverage 120X for the short-insert data and 40X for the long-insert data. The performance of Telescoper does not degrade with higher coverage data.

We sought to assess assembly for the 40 kb telomeric regions at the ends of each of *S. cerevisiae*'s 16 chromosomes. To this end, we simulated data only from this region. For the real data we used the full data set, but restricted evaluation statistics of the produced contigs to those alignable to the 32 telomeres, each of length 40 kb.

Details of running the various algorithms, including parameter settings and runtimes, can be found in the Supplementary Material. To optimize the performance of the other algorithms, insert distribution and coverage parameters were provided where appropriate. We did not include SGA for D2 and D3 since it was designed for reads of at least 100 bp.

	Produced (kb)				Aligned (kb)					
Assembler	N50	NG50	Max	Total	N50	NG50	Max	Total	%Aligned	%Covered
Telescoper	40.0	40.0	41.0	1208	40.0	40.0	40.0	1172	97.0	90.4
ABySS	31.0	31.0	39.0	1296	31.8	31.8	39.3	1244	95.9	84.7
ALLPATHS2	35.2	33.0	39.0	1047	35.2	33.4	40.0	1032	98.5	80.6
SOAPdenovo	25.0	24.0	39.0	1149	28.6	24.6	40.0	1068	92.9	82.3
Velvet	13.9	9.0	31.0	964	13.9	9.5	31.6	947	98.2	73.7
SGA	31.2	27.0	39.0	1110	31.6	27.2	40.0	1075	96.8	82.0

(a) Results for simulated data D1 (read length = 101 bp)

	Produced (kb)				Aligned (kb)					
Assembler	N50	NG50	Max	Total	N50	NG50	Max	Total	%Aligned	%Covered
Telescoper	39.0	38.0	39.0	1162	38.8	38.3	39.8	1155	99.4	90.3
ABySS	12.1	8.0	31.0	1097	13.7	8.9	31.6	966	88.0	75.0
ALLPATHS2	32.0	27.0	39.0	968	32.8	27.7	40.0	950	98.2	74.3
SOAPdenovo	25.0	21.0	39.0	988	24.6	20.8	40.0	954	96.5	74.3
Velvet	14.0	9.0	31.0	955	14.2	9.5	31.9	939	98.3	73.2

(b) Results for simulated data D2 (read length = 50 bp)

Table 2.1: Summary of assembly results based on simulated data from 32 telomeric regions each of length 40 kb. "%Aligned" is the ratio of Total Aligned to Total Produced, while "%Covered" is the fraction of the telomeric regions covered by contigs.

Assembly performance

Several standard metrics exist for measuring assembly performance in the absence of a reference genome. They include the length of the largest contig, the total length of all contigs, and N50 (which is equal to the longest contig length such that the sum of the lengths of all longer contigs is half the total output assembly). An additional metric is NG50 [Earl et al., 2011], which is similar to N50 but more comparable across assembly algorithms. When the genome length is known, then rather than using each algorithm's estimate of the genome size, which can fluctuate widely depending on the threshold at which small contigs are output, one can use the true genome size. Thus NG50 is defined as the length of the longest contig such that the sum of all longer contigs is half the total genome size. We considered the above-mentioned metrics in our study.

To investigate assembly accuracy, we mapped each contig to the reference genome using NUCmer from the MUMMER package [Delcher, 2002]. For each contig, we determined to which telomere it maps best according to the total number of aligned bases. The number of aligned bases in each contig forms a more useful foundation for accuracy-informed continuity statistics than the direct number of bases in each contig. Therefore, we also computed the aforementioned metrics using these aligned lengths.

The results of our study for simulated data are summarized in Table 2.1, while the results for the real data are shown in Table 2.2. These results are for the 32 telomeric regions, each

	Aligned (kb)				
Assembler	N50	NG50	Max	Total	%Covered
Telescoper	34.5	32.8	39.2	980	75.8
ABySS	12.0	8.3	31.3	971	75.3
ALLPATHS2	26.3	16.5	40.0	923	70.1
SOAPdenovo	21.4	16.2	39.3	879	68.6
Velvet	11.8	6.9	31.3	928	72.2

Table 2.2: Summary of results for real data D3. The contigs produced by each algorithm were aligned to the 32 telomeric regions each of length 40 kb. As before, "%Covered" is the fraction of the telomeric regions covered by contigs.

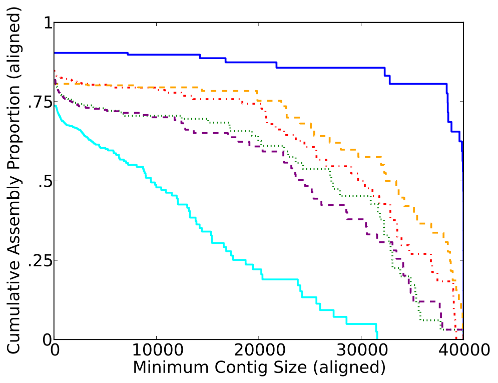
of length 40 kb. As the tables show, Telescoper exhibited the best performance under most metrics, with notable margins from the second best method. As shown in Table 2.1, reducing the read length from 101 bp to 50 bp while keeping all other parameters the same worsened the performance of most algorithms, with ABySS being the most affected.

Figure 2.5 provides a more detailed picture of contig length distribution. These plots show the cumulative proportion for all aligned contigs exceeding the contig size indicated on the x-axis. NG50 can be read from the plots as the x-coordinates at which each curve hits the 50% mark of bases output relative to the reference. The best possible curve is the constant function $y = 1$, so the closer a curve is to that line, the better the performance. Note that for any given minimum contig size (the x-axis value), Telescoper produced more alignable bases than all other methods compared, for all three data sets. Furthermore, Figures 2.5(a)-(b) illustrate that Telescoper is more robust to a decrease in read length than are the other algorithms.

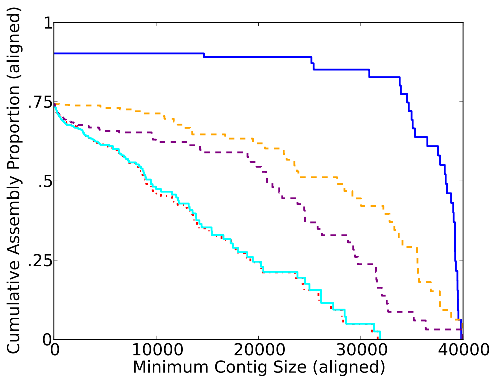
For Telescoper, the observed difference between the corresponding curve in Figure 2.5(b) and that in Figure 2.5(c) is largely attributable to the difference in the insert-size distribution. On simulated 50 bp data with long-inserts with mean length 2.2 kb and short-inserts with mean length 400 bp, the performance of Telescoper was similar to that shown in Figure 2.5(c) (see Supplementary Material), suggesting that Telescoper is robust the complications of real data and that the observed good performance of Telescoper in Figure 2.5(b) is due to its improved ability to take advantage of a longer (10 kb instead of 2.2 kb) long-insert distribution.

Of further importance is the extent to which an algorithm produces false bases or contigs. Because we forced each contig to align to a single telomere, chimeric contigs created by joining portions of different telomeres were penalized as having bases that do not align. As shown in the "% Aligned" column of Table 2.1, Telescoper was the top performer in this regard for D2, and followed ALLPATHS 2 and Velvet closely for D1.

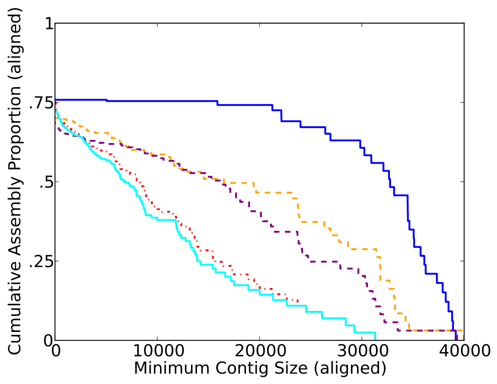
Finally, we considered visually examining the alignments of contigs onto each telomeric region. Figure 2.6 shows the results for two chromosomes, with contigs from each assembly algorithm aligned to them. For each algorithm, each contig is represented by a different



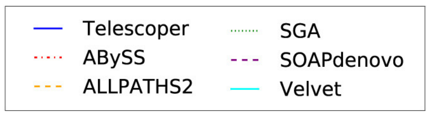
(a) Simulated data D1: 101 bp reads, 10 kb long-insert mean



(b) Simulated data D2: 50 bp reads, 10 kb long-insert mean



(c) Real data D3: 50 bp reads, 2.3 kb long-insert mean



(d) Legend.

Figure 2.5: The cumulative proportion of all aligned contigs exceeding the contig size indicated on the x-axis. These plots illustrate the continuity and completeness of different assemblies. For any given minimum contig length, Telescoper produced more aligned bases. NG50 can be read from this graph as the x-coordinates at which each curve hits the 50% mark of bases output relative to the reference. (a) Results on simulated data D1. (b) Results on simulated data D2. (c) Results on real data D3. (d) The legend for subfigures (a)-(c).

color, so more colors per method indicates a larger number of contigs. For each telomeric region shown, Telescoper produced a single contig for almost the entire region, while other algorithms often produced many small contigs.

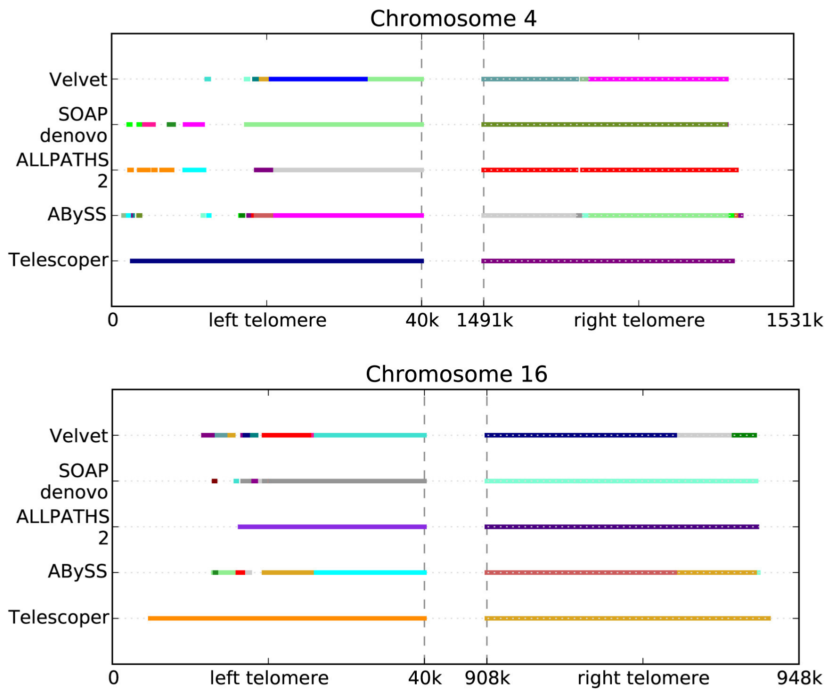


Figure 2.6: Contig continuity results for real data D3. The left and right telomeric regions (separated by the dotted line) for two different chromosomes are shown, with the aligned contigs displayed for each assembly algorithm. Different colors represent different contigs in the produced assembly, so more colors per method indicates a larger number of contigs. For each telomeric region shown, Telescoper produced a single contig for almost the entire region, while other algorithms often produced many small contigs.

2.4 Conclusion

We have introduced several new ideas for *de novo* genome assembly, geared towards highly repetitive regions. Our preliminary assembler, Telescoper, proceeds by iteratively extending paths and selecting between them using the empirical distributions formed by both long-insert and short-insert paired-end reads. Although our iterative extension step is similar to the method from Parrish et al. [2011], we note that their method is designed for the targeted assembly of an individual's insertions relative to a reference genome.

The utility of Telescoper was validated in a study on both real and simulated data from the 40 kb telomeric regions of each chromosome of *S. cerevisiae*. For all three data sets tested, Telescoper produced more continuous assemblies than the other algorithms considered. In our evaluations, we tried to include the strongest and most popular algorithms with available implementation. Unfortunately, ALLPATHS-LG [Gnerre et al., 2011] could not be included, because of its small-fragment library requirement mentioned in Introduction. We considered several standard metrics for comparing assemblies, but we note that the task of comparing genome assemblies is a large one, with several papers exclusively devoted to it [Earl et al., 2011, Salzberg et al., 2012].

Other researchers are currently working on algorithms for identifying assembly errors using features derived from read-mapping. Rather than having this be a downstream process, we believe that it would help to incorporate such features directly into an assembly algorithm. Here we make an effort in this direction by scoring assembly extensions according to read-mapping statistics. Although the scoring scheme used in this paper may not be optimal, we have demonstrated that the idea of simultaneously pursuing multiple extensions, and concurrently using multiple libraries to score and select among them is promising.

The current implementation of Telescoper can be used as a finishing algorithm to extend contigs into repetitive regions and produce better assemblies for telomeres. Other applications include targeted *de novo* assembly of structural variants and highly variant regions such as human leukocyte antigen. Future work would include extending the ideas presented here to whole genome assembly, improving error-correction, producing more exhaustive listings of potential paths, and more thorough evaluation of the alternate paths. Also, additional validation metrics such as those explored by Salzberg et al. [2012] can be incorporated as well.

We often see cases where, if we took the union of all assemblies, we could produce a much better final product. This suggests that assembly is not a solved problem, and that the strengths of different algorithms can potentially be combined to produce better assemblies. We believe the ideas behind Telescoper have the potential to improve *de novo* assembly significantly and provide a comprehensive picture of previously unresolved repetitive regions.

Chapter 3

Maximum Likelihood Assembly

3.1 Introduction

In this chapter we continue the theme of this thesis, and describe a new *de novo* assembly algorithm targeted towards resolving repeats. Many types of repeats occur in genomic sequences, including triple repeats, near-repeats, tandem repeats, inversions, and rearrangments. Such repeats form a major barrier to perfect genome assembly, and current state-of-the-art assembly algorithms produce sequences that are often very different from the original.

The vast majority of assembly algorithms take the same high-level approach. The algorithm first generates a contig graph; next, the algorithm attempts to use mate-pair information to elongate the contigs, or at least to determine their relative position to create scaffolds. Repeats result in a more fragmented contig graph with more complex relationships between the contigs, creating a more challenging problem for the contig-elongation step. It is desirable for a contig graph to be *perfect*; a contig graph is said to be perfect if the reference sequence can be traced out as a path through the graph. Yet, even with a perfect contig graph, most assemblers fail to generate long contigs in the presence of complicated repeat structure. This holds true even if the mate-pair reads are, to a good approximation, consistent with only one solution. Note that in such a situation the assembly problem is statistically feasible, and the problem is largely a computational one.

The situation is of course made more challenging when there are read errors, or when reads multi-map to multiple contigs, or when multiple solutions can explain the reads, but some solutions are more likely than others.

One approach to creating longer contigs from a contig graph is to sequentially apply heuristic steps. However, it can be challenging to know when to apply them or what heuristics to use. An alternative, principled approach, is to pose the assembly problem as a maximum-likelihood/optimization problem or as a constraint satisfaction problem, where in the latter, the constraints are requirements that mapped read-pairs have a good orientation and appropriate distance in the assembly.

The idea of treating assembly as a maximum-likelihood problem (i.e. choosing the assembly that maximizes the probability of the observed reads) has been around for a while. Maximimum likelihood assembly was proposed by Myers [1995]. It is also mentioned in the Sanger assembler Celera [Myers, 2000] to identify collapsed repeats, in Medvedev and Brudno [2009], and in metagenomics [Laserson et al., 2011] and quasi-species (ShoRAH [Zagordi et al., 2011], ViSpA [Astrovskaya et al., 2011]).

A more developed framework, using mate pairs and errors, is discussed in recent assembly evaluation papers CGAL [Rahman and Pachter, 2013], ALE [Clark et al., 2013], and Ghodsi et al. [2013], which use the likelihood not to do assembly, but to compare assemblies produced by different programs, or with different parameters. We will use CGAL in this chapter to evaluate the assemblies we produce.

Other papers use likelihood to a limited extent, to choose between few alternatives. For instance, Dindel [Albers et al., 2011] generates eight candidate haplotypes in a given realignment window, and then considers each indel applied, and chooses the most likely one.

In our algorithm, called Piper, we begin with a contig graph produced by another assembler, and search for a full assembly solution represented as a set of paths through the graph. The problem is then reduced to choosing the correct path set through the graph, among the many candidate path sets. Our method compares the potential path sets using a likelihood framework: given a generative model, we keep only the solutions that produce the observed reads with high likelihood. Because the large number of solutions prohibits evaluation of likelihoods for them all, we impose a number of hard constraints to reduce the search space of assemblies. In some instances, these hard constraints are actually sufficient to determine a single assembly.

We draw our examples in this section from repeat-rich regions and segmental duplications, i.e. duplications followed by continued genome evolution. Such regions are difficult to assemble, and many assembly algorithms output collapsed sequences that are shorter than the truth, or simply fail to assemble at all. Genomic regions with similar content are biologically important, because they reflect different selective pressures from the rest of the genome, which underscores the need to produce accurate assemblies despite the difficulty.

We compared Piper to existing algorithms as follows. We first generated a contig graph using the assembler Discovar [Computational Research and Development Group, Broad Institute of MIT and Harvard, 2014], and then ran our assembler on its output. According to results from Assemblathon 1 [Earl et al., 2011], SOAPdenovo [Li et al., 2010] and ALLPATHS-LG [Gnerre et al., 2011] were the top two performing assemblers at the time. Hence we compare the results of our likelihood-based assembly to ALLPATHS-LG [Gnerre et al., 2011], SOAPdenovo2 (the successor to SOAPdenovo) [Luo et al., 2012], as well as SGA [Simpson and Durbin, 2012].

Since the algorithm Piper takes a contig graph as input, comparison to other algorithms that take a contig graph will be more accurate than comparison to algorithms which generate a potentially inferior contig graph. Both SGA and SOAP-denovo separate their contig graph creation and scaffolding stages, so we can run the scaffolding stages on the same Discovar output that we input to Piper. Dedicated scaffolders that are not part of assemblers are generally designed to solve a different problem: GRASS [Gritsenko et al., 2012] defines the "contig scaffolding problem" (CSP) as being that of finding a single position and orientation for each contig, so as to minimize the unsatisfied scaffolding constraints. Indeed, most of the scaffolding algorithms described in Hunt et al. [2014] explicitly do not handle repeats; for instance, GRASS [Gritsenko et al., 2012], OPERA [Gao et al., 2011] and MIP [Salmela et al., 2011] remove contigs with read-depth coverage that suggests the contig may be a repeat. Bambus [Pop, 2003] and SSPACE [Boetzer et al., 2011] are both greedy algorithms that place each contig at only one location. Therefore, we compare Piper only with the afore-mentioned assembly algorithms and their scaffolding procedures.

Our work is unique in that it is the first assembler of which we are aware that attempts to do *de novo* assembly by explicitly generating a set of alternative assemblies, and choosing amongst them using a likelihood computed using paired-reads. The results on small simulated datasets are very good: we are able to completely assemble regions that no other algorithm does.

3.2 Methods

In this section, we describe in detail the Piper algorithm. The algorithm takes as input a contig-graph and outputs a set of assemblies that are all likely solutions. Often more than one assembly is equally likely, so we cannot narrow down the solution precisely. Each assembly consists of a set of lists of overlapping contigs, i.e. a set of strings. Piper outputs a set of contigs in which it is confident, i.e. the maximal substrings that appear in all of its assemblies. Piper also outputs a graph connecting these lists of contigs, as is standard in fastg format.

Likelihood formulation. A key component of the algorithm is computation of the likelihood of an assembly, i.e., the probability of obtaining the given set of reads from a proposed assembly. At this stage the contig graph is assumed to be fixed, and the only variable is the assignment of multiplicities to the contigs and the paths through them.

Our computation of the likelihood of the assembly is based on the following generative model. All genome assemblies are assumed to have the same prior probability. Long-insert read-pairs are generated independently from the sequence as follows: a location is chosen for the left read of a read-pair, uniformly at random from all positions along the sequence, and an insert size is chosen according to a predetermined insert size distribution. If the right read's generated position is beyond the length of the sequence, the read-pair is discarded. Errors in the read are generated according to some sequence-independent distribution. (Ideally, errors in the read would be modeled as being generated according to a distribution that is a function of the position in the read, the true base, and the quality score, but at present we focus on a more tractable model.)

The probability of observing a particular set of N reads given the assembly is the product of the probability of observing N reads given the assembly length A, the arrival rate λ (i.e., the inverse of the expected distance between reads), and the read-pair probabilities. Borrowing some notation from Rahman and Pachter [2013], and using $P_{\text{poiss}}(X|y)$ to indicate the probability that a Poisson-distributed random variable with rate y takes on value X, the likelihood is

$$l(\mathcal{A};\mathcal{R}) = \log P_{\text{poiss}}(N|\lambda A) + \log \prod_{i=1}^{N} p(r_i|\mathcal{A}), \tag{3.1}$$

where $\mathcal{A}$ is the assembled sequence, and the ith read-pair has likelihood

$$p(r_i|\mathcal{A}) = \sum_{j=1}^{M_i} \frac{1}{A} p_F(l_{i,j}) p_E(e_{i,j}).$$

That is, the probability of generating a particular read-pair is the sum of the probabilities of generating it from each of the A^2 locations from which it could have been generated. We approximate the locations from which it could have been generated as the genome locations to which the read pair aligns with a small edit distance, and use just the M_i distinct mapping locations j that are returned by a read-mapper which aligns the reads to the assembly $\mathcal{A}$. The probability of generating the read-pair from a particular location depends on the total number of locations (A) from which it could have been drawn, the distribution of fragment size, $p_F(\cdot)$, and the distribution on error probabilities, $p_E(\cdot)$. Each mapping location implies a fragment size $l_{i,j}$, and (together with the sequence specified by $\mathcal{A}$ to be present at the location j) a set of errors $e_{i,j}$. This yields the read probability $\frac{1}{A} p_F(l_{i,j}) p_E(e_{i,j})$.

This model is a somewhat coarser version than that used by Rahman and Pachter [2013], who propose to compare whole genome assemblies produced by competing assemblers by computing their likelihoods, rather than $N50$ or some other statistic. They also model sequence-dependent error, as well as sequence-dependent arrival rate. They note that given their model, the quantity is precise when all possible mapping locations are used, but can be approximated by using just the top mappings.

In order to penalize assemblies with incorrect or missing sequence, Rahman and Pachter [2013] force all reads to align somewhere, and then penalize according to the implied errors. In our application, we simply discard the reads that do not align to the contig graph. This is appropriate in our setting where the contig graph is fixed, and the assemblies differ only in the multiplicity of the contigs and in the implied offsets between them, in which case the poorly-aligned reads do not contain important differentiating information.

Decomposition

In order to make the assembly process more efficient, we do not consider all possible assemblies, a.k.a. sets of paths through the contigs, but only a subset. We restrict the subset of possible assemblies in this section.

To begin with, we will want to work with a unitig graph in addition to the contig graph that we are given as input. Recall that a **unitig** is an unbranched path in the read graph. In order to obtain unitigs from a contig graph, we replace reads in this context with the set of L-mers in the contigs and create edges between "reads" only when they overlap by $L-1$ bases. Equivalently, we could define a unitig seeded around an L-mer x to be the longest string containing x such that every other L-mer y in the unitig occurs, in a path through the contig graph, only where the entire unitig appears. The unitig set corresponding to a contig set is the set of unitigs seeded at all L-mers in the contigs. The unitig set has two nice properties: First, every contig can be written (uniquely) as an ordered list of unitigs. Second, every error-free read maps uniquely to a single unitig. Note that many unitigs are present in more than one contig.

A read-pair **spans** a pair of unitigs (c, d) if one read maps to unitig c and the other maps to unitig d, with orientations consistent with a path from c to d. We include the case where $c{=}d$.

If we assume that reads can be correctly mapped to unitigs, then the formulation of the likelihood simplifies. Each read-pair spans two unitigs (or one unitig), and, letting $\mathcal{U}$ denote the set of all unitig pairs, and $\mathcal{R}_{cd}$ the subset of reads that span a unitig pair (c, d), we can rewrite the likelihood as

$$\begin{aligned} l(\mathcal{A};\mathcal{R}) &= \log P_{\text{poiss}}(N|\lambda A) + \sum_{i=1}^{N} \log p(r_i|\mathcal{A}) \\ &= \log P_{\text{poiss}}(N|\lambda A) + \sum_{(c,d)\in\mathcal{U}} \sum_{i\in\mathcal{R}_{cd}} \log p(r_i|\mathcal{A}). \end{aligned}$$

In other words, we partition the reads into sets according to the pair of unitigs which they span. Now we wish to find the probability of generating the read-pairs spanning a unitig pair. This will depend on the number of instances and gaps with which the unitig pair appears in the assembly. To get these values, it is sufficient to know which of the paths that connect the unitig pair appear in the assembly, along with the number of times they appear.

Let M denote some length, e.g. the max insert size. Then define the set of **pathlets** $\mathcal{Z}_f^M$ to be the set of paths through the contig graph starting at contig f and elongated until the stopping condition is met, where the stopping condition is that the string as measured from the start of the 2nd contig to the end of the pathlet has length at least M or until the final contig in the pathlet has no outgoing edges. If M is the max insert length, then any read-pair mapping with its left-read into f will have its right-read somewhere in the pathlet (rather than to the right of the pathlet). Then

$$\mathcal{Z}^M := \bigcup_f \mathcal{Z}_f^M$$

is the set of all pathlets in the graph. We define the pathlets on the contig graph rather than the unitig graph, because the contig graph has far fewer paths.

We define the set of variables $\{m_z : z \in \mathcal{Z}^M\}$ to be the *multiplicities* of the pathlets, i.e. m_z is the number of times pathlet z's contig sequence appears in the reconstructed genome. If $m_z = 0$ then that pathlet does not appear in the genome.

We can think of $\{m_z : z \in \mathcal{Z}^M\}$ as specifying the "filled-in reads" that we would have observed if we knew the interior of every long-insert read-pair. In other words, just as the unitig graph formed from the K-mer graph with $K = L - 1$ is the best we can do for single-read assembly, $\{m_z : z \in \mathcal{Z}^M\}$ implies a graph that is the best we could do with complete sequencing of the fragments from which the read-pairs were made. This is, of course, an upper bound to how well we could hope to assemble using paired-end reads, since we will, in general, find multiple such graphs that are consistent with the data.

The pathlets satisfy some basic **flow constraints.** In order to explain them, we introduce a particular definition of a suffix of a path: suffix(z) is the contig-list created from z by removing just the first contig in z's contig-list. Due to how we defined $\mathcal{Z}^M$, we note that pathlets in $\mathcal{Z}^M$ that contain g as their second node can be thought of as the product-set of choice of contig prior to g and choice of suffix beginning at g. Letting $\mathcal{S}^M$ denote the set of suffixes of paths in $\mathcal{Z}^M$ that are also strict prefixes for some pathlet, we obtain the following flow constraints, which essentially just enforce that flow into a path $s \in \mathcal{S}^M$ is equal to the flow out:

$$\sum_{z \in \mathcal{Z}^M | \text{ suffix}(z) = \text{ s}} m_z = \sum_{y \in \mathcal{Z}^M | \text{ prefix}(y) = s} m_y \quad \forall s \in \mathcal{S}^M. \tag{3.2}$$

The flow constraints would be violated if a chromosome starts with a suffix, or ends in a suffix that is the strict prefix of some pathlet. That will only happen if the end of a chromosome (or assembled region) is identical to some sequence interior to another chromosome or scaffold. We could also consider permitting a fixed number of violations of the flow constraints, with some penalty, but do not do so here. We note that given our definitions, it is not a problem for two chromosomes to start with the same prefix or to end with the same suffix.

For ease of exposition we assume for a moment that each unitig in a pathlet occurs just once in that pathlet, though we shortly elaborate on the case where it does not. The likelihood can be rewritten in terms of these variables $\{m_z : z \in \mathcal{Z}^M\}$, as

$$\begin{aligned} l(\mathcal{A}; \mathcal{R}) &\approx \log P(N|\lambda, A) + \sum_{c,d \in \mathcal{U}} \sum_{i \in \mathcal{R}_{cd}} \log \sum_{z \in \mathcal{Z}_{cd}} m_z p(r_i|z) \\ &= \log P(N|\lambda, A) + \sum_{c,d \in \mathcal{U}} \sum_{i \in \mathcal{R}_{cd}} \log \sum_{z \in \mathcal{Z}_{cd}} m_z \frac{1}{A} p_F(l_{i,z}) p_E(e_{i,c,d}), \end{aligned}$$

where $\mathcal{Z}_{cd} = \{z \text{ s.t. } d \in z | z \in \bigcup_{f \text{ s.t. } c \in f} \mathcal{Z}_f^M\}$ is the set of pathlets that start at one of the contigs (f) containing the unitig c and where the pathlet also contains the unitig d, downstream of c, and $l_{i,z}$ is the fragment size implied by mapping read-pair i to the pathlet z (obtained by mapping the reads in read-pair i onto the unitigs c, d and mapping c, d onto

z) and $e_{i,c,d}$ is the errors implied in mapping the read-pair i to unitigs c, d. Note that the definition of $\mathcal{Z}_{cd}$ ensures that a particular unitig pair (c, d) is counted only once per path that flows through it; any two pathlets in $\mathcal{Z}_{cd}$ cannot come from the same physical place in the genome.

The likelihood above can be easily modified when a unitig maps in multiple places to z. If the unitig c appears twice in the first contig of z, or if the unitig d appears twice in the pathlet, then we replace $m_z \frac{1}{A} p_F(l_{i,z}) p_E(e_{i,c,d})$ by $m_z \sum_{k=1}^{M_z} \frac{1}{A} p_F(l_{i,z,k}) p_E(e_{i,c,d})$ where k enumerates over the M_z pairs of mapping locations of the unitig c onto the contig f and of the unitig d onto the pathlet z , and $l_{i,z,k}$ is the implied fragment size from mapping read-pair i onto the unitigs c, d and then using the kth mapping of (c, d) onto z.

We note that because we mapped each read to a unique unitig, the factor $p_E(e_{i,c,d})$ can be pulled out into its own sum, which is independent of the assembly. Intuitively, this is because once we made the approximation of being confident in the mapping of the read-pair to the unitig pair, the errors no longer play a role.

Unitig multiplicity constraints: In order to limit the space of solutions over which we must search, we can impose some hard constraints on the unitig multiplicities, which will be satisfied with high probability. Specifically, we can place constraints on m_c (multiplicity of unitig c) that it lie in the interval $m_c^- < m_c < m_c^+$. We obtain m_c^- and m_c^+ as the min and max values respectively of m_c using

$$\begin{aligned} m_c^- &= \min\left(\left\{ m_c \in \mathbb{N} \middle| \log \frac{P(N_c|m_c\gamma_c)}{P(N_c|m_c^*\gamma_c)} < T_1 \right\}\right) \\ m_c^+ &= \max\left(\left\{ m_c \in \mathbb{N} \middle| \log \frac{P(N_c|m_c\gamma_c)}{P(N_c|m_c^*\gamma_c)} < T_1 \right\}\right) \end{aligned} \tag{3.3}$$

for some user-input threshold T_1, where γ_c is the expected number of reads N_c to arrive in contig c (given its length, the read arrival rate λ, the read-length, and the read-mapping error-rate, and assuming it occurs just once), and $m_c^* = \text{argmax}_{m_c} P(N_c|m_c\gamma_c)$. We use the heuristic of $P(N_c|m_c\gamma_c) = P_{\text{poiss}}(N_c|m_c\gamma_c) + \epsilon^{N_c}$ (with a small value of ϵ), in order to account for the fact that reads can be mismapped, and to allow a unitig c to have multiplicity $m_c = 0$ even if a read mapped to it. All told, these heuristic bounds are very similar to the estimate of multiplicity using the A-statistic in Myers [2000].

We then require that $\sum_{z \in \mathcal{Z}_c} m(z)$ fall within the hard constraints, where $\mathcal{Z}_c = \{z | z \in \bigcup_{f \text{ s.t. } c \in f} \mathcal{Z}_f^M\}$. If the unitig appears more than once within a contig f, then we count the pathlets in $\mathcal{Z}_f^M$ the corresponding number of times.

Unitig pair multiplicity constraints: Similarly, we can impose hard constraints on the unitig-pair multiplicities. Define the variable $\eta_{cd}(g) = \mathbb{E}(N_{c,d}|l_{cd})$ to be the expected number of read-pairs between unitigs c and d if the unitigs were to appear a single time in the genome, given their separation distance l_{cd}, their lengths, the read arrival rate λ, and the read mapping error rate. We note that up to a constant factor related to the read-mapping error,

$$\eta_{cd} \propto \int_{s=0}^{\text{len}(c)} \int_{t=0}^{\text{len}(d)} \lambda p_F(l_{cd} + s + t) dt ds.$$

Then given the set of pathlets between c and d, we can find η^+_{cd} and η^-_{cd} the max and min values over the various values of $l_{c,d}$.

Define $m_{cd} = \sum_{z\in\mathcal{Z}_{cd}} m_z$. Then we can bound this quantity, by requiring that the relative likelihood between the most likely multiplicity and the least not exceed some threshold T_2.

Note that we could add additional, tighter constraints, by placing bounds on subsets of the pathlets, but do not do so in our current implementation.

Further approximating the likelihood: For unitig pairs $(c, d) \in \mathcal{U}$ with all offsets implied by pathlets containing the unitigs having very similar length, we can make the approximation that $p_F(l_{i,z}) = \text{const}$ with respect to z. Dropping terms that are constant with respect to the assembly, this yields

$$\begin{aligned}\sum_{i\in\mathcal{R}_{cd}} \log\left(\sum_{z\in\mathcal{Z}_{cd}} m_z \frac{1}{A} p_F(l_{i,z})\right) &= \sum_{i\in\mathcal{R}_{cd}} \log \sum_{z\in\mathcal{Z}_{cd}} \frac{1}{A} m_z + \text{const}\\ &= N_{cd} \log \sum_{z\in\mathcal{Z}_{cd}} \frac{1}{A} m_z + \text{const}.\end{aligned}$$

We note that $\sum_{c,d\in\mathcal{U}} m_{cd}\eta_{cd} = \mathbb{E}(N|\lambda, A)$.

Doing some algebra (which we omit here), then assuming all unitig-pairs are connected by a single-length path, we can rewrite the total likelihood function as

$$\begin{aligned} l(\mathcal{A};\mathcal{R}) &\approx \sum_{c,d\in\mathcal{U}} \left(\log E(N_{c,d}|\lambda) + \sum_{i\in\mathcal{R}_{cd}} \log \sum_{z\in\mathcal{Z}_{cd}} m_z p_F(l_{i,z})\right) + \text{const}\\ &= \sum_{c,d\in\mathcal{U}} \log P_{\text{poiss}}(N_{c,d}|m_{cd}\eta_{cd}) + \text{const},\end{aligned}$$

where the constant does not depend on the variables $m(z)$.

One advantage of this formulation is that each term should be small, which can help in the computation to minimize the total.

At present we actually do not include any contribution to the likelihood from unitig pairs whose path-lengths differ by more than some threshold. We should do this though; we could consider the breakdown of the multiplicity over the subsets of pathlets in $\mathcal{Z}_{cd}$ partitioned according to the implied offset between c and d and compute the likelihood contribution.

Finally, we add to the log-likelihood the log-likelihood that one would get based on just single-end reads. Whereas in the previous step we excluded the contribution of some unitig pairs, for this value it is easy to include all unitigs.

Optimization Having defined the variables, some hard constraints, and likelihood function, we would like to find the assemblies that maximize the likelihood. To this end we use z3, a weighted constraint solver / theorem prover being developed at Microsoft Research (https://z3.codeplex.com). z3 takes both hard constraints, which must be satisfied, and soft-asserts, which when satisfied, contribute a weight. The sum of the weights from the satisfied soft-asserts will yield our approximate likelihood.

The problem as entered to z3 is

- variables are pathlet multiplicities $\{m_z : z \in \mathcal{Z}^M\}$
- flow constraints imposed as hard constraints, Eq. 3.2
- hard constraints on flow through individual unitigs, Eq. 3.3
- hard constraints on flow through unitig pairs
- soft constraints on multiplicities of unitig-pairs. Our current implementation only considers unitig pairs where the distance between the unitigs is pathlet-independent. For each unitig-pair, for each value of the multiplicity that it can take, we incur some weight. The weight comes from the probability that the unitig-pair multiplicity takes on value I given the number of spanning read-pairs observed (N_{cd}), and is a term in the likelihood.
$$\sum_{z \in \mathcal{Z}_{cd}} m_z = I, I \in \{0 \ldots \text{maxI}\},$$
each with weight
$$w_{cd}(I) = -\log P_{poisson}(N_{c,d}|\text{rate} = I\eta_{cd}),$$
for $c \neq d$
- soft constraints on single-unitigs: Similar to the above, but counting the reads mapping within a single unitig, to measure its multiplicity.

The hard constraints do not change the likelihood, and are just there to help reduce the search space. The weights of the satisfied soft-constraints add up to our approximation of the likelihood.

We run the z3 solver on this input, obtaining the minimum-weight set of values for $\{m_z : z \in \mathcal{Z}^M\}$. We then exclude that solution, and search for the next likely one, repeating, until we reach a solution with much smaller likelihood.

Processing the output from z3: Because we made many approximations in our likelihood score computation, we also want to get a more accurate score for each assembly. For a given set of pathlet multiplicities, we evaluate an assembly consistent with the pathlet multiplicities (any such assembly; they should all have very similar likelihoods) using CGAL [Rahman and Pachter, 2013].

We keep all the solutions within some user-defined distance of the top one (often there are multiple solutions with similar likelihoods).

To convert from $\{m_z : z \in \mathcal{Z}^M\}$ to an assembly, we note that the $\{m_z : z \in \mathcal{Z}^M\}$ variables form a "pathlet graph", where nodes are pathlets and edges are generated when the suffix and prefix of two pathlets match. If there is any subgraph with a single source/sink in that subgraph, then if there is a single Hamiltonian cycle consistent with the pathlet multiplicities, then we replace that subgraph with the Hamiltonian cycle. We merge any unbranched chains in this graph, similarly to how we merged reads in unitig formation or

Kmers in the condensed Kmer graph in other chapters. The result of all this is a simplified version of the pathlet graph, where each node is a contig.

We also create a consensus solution across all the assemblies. The consensus solution is just a set of (greatest) substrings that are common to all solutions, which we output as contigs. Any sequences of contigs that remain in some of the assemblies but not all of them are also output as contigs, but annotated as uncertain. We also include information about the edges between the contigs that we output.

Details: In order to speed up the computation of the η_{cd}'s, as well as to have fewer constraints in z3, we do not print unitig-pair constraints if either unitig has length less than 150 or if the start-to-end distance between the unitigs is less than the min likely insert size or if the end-to-start distance between the unitigs is more than the max likely insert size.

Rounding the weights makes the z3 computation run much faster, so we round them to the nearest 0.2 (an ad-hoc choice of value).

In enumerating the pathlets, we avoid traversing a unitig more times than our estimate of its max multiplicity; otherwise we had datasets on which the number of pathlets in the graph was far too large to use.

3.3 Results

Data and comparisons

We evaluate our method's performance on a few segmental duplications in the human genome. For each region we simulate two different libraries from the haploid sequence from NCBI:

- D1: a short-insert (600bp fragment) library of 250 bp reads ("Broad-style short insert library")
- D2: and a long-insert (2000bp fragment) library of 101 bp reads. ("long insert library")

The first library is the required input library for the program Discovar, which is a state-of-the-art *de novo* assembler that works with small datasets, and is intended for assembly so accurate that SNPs and small structural variants (SVs) can be called. The second is a long-insert library, such as is required by most whole genome *de novo* assemblers in order to produce accurate assemblies. ALLPATHS-LG, for example, requires a long-insert library, in addition to a short-insert library with fragment size no more than 3 times the read-length. Discovar accepts only a short-insert library, with the same restrictions as ALLPATHS-LG's short-insert library. With the exception of ALLPATHS-LG and Discovar, most other whole genome *de novo* assemblers are flexible and can use either a short-insert library alone, or a short-insert library together with one or more long-insert libraries.

We evaluate our results using the GAGE evaluation script [Salzberg et al., 2012], which returns a report on the assembly quality, including the information categories listed in 3.2.

Genomic Regions

We consider a small number of datasets, ranging in size and complexity. We chose Nebulin due to its mention by Jaffe et al. [2013]. The other datasets were obtained from Human Segmental Duplication Database [Bailey et al., 2001]. For datasets without a perfect repeat longer than 600 bp, we used only a short-insert library in evaluating Piper, in order to see how well it could do with limited information. We gave the other algorithms both libraries in every case (additional libraries should never degrade the assembly).

Isolated duplication 1 Our first example is the simplest possible. The genomic regions chr4:99801213-99802294 and chr17:47500655-47501746 contain a perfect repeat of copy number 2, of length 1091. For our first example, we take these two regions, plus 5000bp on either side, and simulate reads. The resulting contig graph produced by Discovar, and shown in Fig 3.1(a) is very simple, and contains just 5 contigs: the repeat, the two preceding regions, and the two following regions. The results for each algorithm are shown in Table 3.1(a); Piper, ALLPATHS-LG, and SOAP all do well, with each of these producing one correct or nearly-correct scaffold for each region from the two chromosomes. SOAP also produces an extra short contig corresponding to the repeat. In other examples of a single perfect repeat surrounded by non-repetitive sequence, we saw similar results.

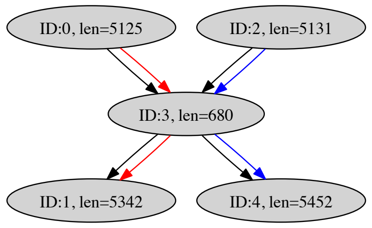
Tandem duplication 1 The region chr3:197162964-197164034 contains a tandem segmental duplication of length 1113, with some differences between the two copies. We surrounded it with 10,000bp of flanking sequence on either end of the region, so that algorithms could infer the insert distribution. For this dataset, we used only the short-insert library for Piper, because none of the exact repeats were longer than 600bp.

The contig graph input to all the algorithms is shown in Fig 3.1(a). The results of the scaffolding/assembly algorithms are summarized in Table 3.1(a). Using just the 600bp library, we produced two assemblies with very similar CGAL likelihood scores (difference less than 3.0, although the higher-likelihood solution was in fact the correct one), so our output for this was a set of contigs present in both assemblies, including the output contigs created from the input contigs (5, 2, 4, 8) and (5, 3, 7, 8).

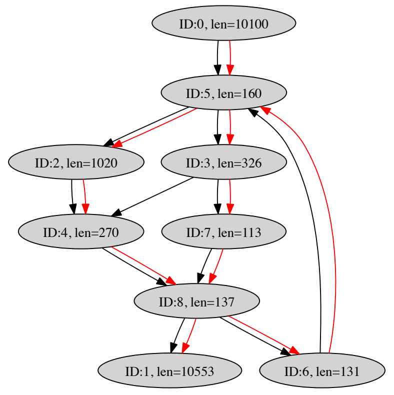
Tandem duplication 2 The region chr9:35372441-35373492 contains a tandem segmental duplication of length 1055. While at first glance this dataset is not more interesting than our "Tandem duplication 1", it contains many many more repeats on the length scale of 101bp than at the length 250bp, making read-mapping ambiguous. For this dataset, we used only the short-insert library for Piper, because none of the exact repeats were longer than 600bp. The contig graph input to all the algorithms is shown in Fig 3.1(b). The results of the scaffolding/assembly algorithms are summarized in Table 3.1(b). In this case we produced two assemblies that were equally likely according to our approximate likelihood score, but which were markedly different according to their CGAL scores, allowing us to provide a single likely assembly.

Nebulin Nebulin is a muscle protein gene that is roughly 249 kbp long and has undergone two duplications. The resulting three regions are about 10.5 kbp each, have over 99% sequence identity to one another, and no two of them are obviously more similar than the other two, leading to the interpretation that the duplications date back to approximately the same time [Björklund et al., 2010]. This gene is similar to other repetitive eukaryotic genes: it is composed of protein domain repeats, and has undergone segmental duplications, causing interesting repeat patterns. It also has an interesting evolutionary history across different species [Björklund et al., 2010]. We will consider the human version.

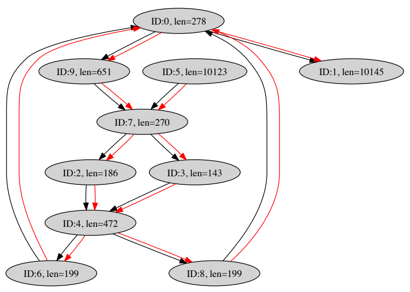
The contig graph input to all the algorithms is shown in Fig 3.1(c). The results of the scaffolding/assembly algorithms are summarized in Table 3.1(c). Piper's results on this dataset were very good; it correctly reproduce the entire gene's sequence, whereas none of the other assemblers or scaffolders, including ALLPATHS-LG, output more than a single "consensus" sequence for the 3 copies of the repeat of interest.



(a) Isolated duplication 1



(b) Tandem duplication 1



(c) Tandem duplication 2

Figure 3.1: To create each graph, reads were generated and input to Discovar. The graphs pictured are the contig graphs after converting from an edge-string to node-string format. Black edges denote overlap. Colored edges are superimposed on this black-and-white graph to indicate the paths consistent with the reference, e.g. for Fig. 3.1(a), the true assembly consists of the contigs ((0, 3, 1), (2, 3, 4)).

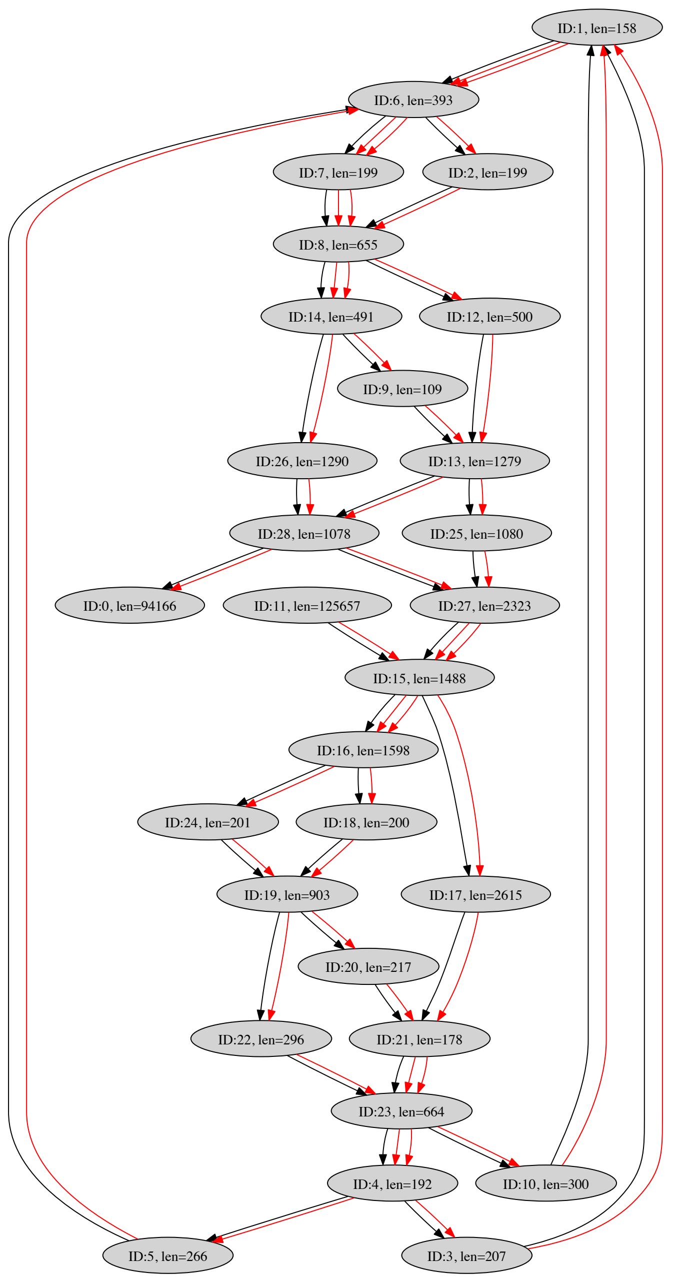


Figure 3.2: The output graph produced by Discovar for Nebulin. To create the graph, reads were generated and input to Discovar. The graph pictured is the contig graph after converting from an edge-string to node-string format. Black edges denote overlap. Colored edges are superimposed on this black-and-white graph to indicate the paths consistent with the reference.

	Contigs		Coverage					Errors		
Asm	N	N50	MR	CR	DR	Ch	MA	S	I	SV
D	5 / 5 / 0	5145	16	680	0	0	0	0	0 / 0	0 / 0 / 0
P^S	2 / 2 / 0	10949	158	0	0	0	0	0	0 / 0	0 / 0 / 0
A^A	2 / 5 / 0	10231	108	141	0	0	0	0	0 / 1	0 / 0 / 2
Sg^A	11 / 9 / 0	5214	258	679	1142	0	0	0	0 / 0	0 / 0 / 0
So^A	241 / 57 / 1	416	118	1247	51009	12808	291	3	0 / 0	0 / 0 / 0
Sg^S	5 / 5 / 0	5145	16	680	0	0	0	0	0 / 0	0 / 0 / 0
So^S	3 / 5 / 0	10712	268	605	0	0	736	0	18 / 0	0 / 0 / 0

(a) Running assemblers and scaffolders on the isolated duplication 1 data set.

	Contigs		Coverage					Errors		
Asm	N	N50	MR	CR	DR	Ch	MA	S	I	SV
D	3 / 3 / 0	10073	33	651	1039	2243	0	0	0 / 0	0 / 0 / 0
P^S	4 / 4 / 0	10100	0	0	131	131	0	0	0 / 0	0 / 0 / 0
A^A	1 / 2 / 0	20340	339	2104	0	0	0	0	0 / 0	0 / 0 / 0
Sg^A	102 / 10 / 0	10017	146	0	24119	0	0	0	0 / 0	0 / 0 / 0
So^A	238 / 62 / 0	329	14	1569	49788	10901	29	0	0 / 0	0 / 0 / 0
Sg^S	3 / 3 / 0	10073	33	802	1190	2243	0	0	0 / 0	0 / 0 / 0
So^S	1 / 2 / 0	22486	108	457	1024	223	577	57	32 / 0	0 / 1 / 0

(b) Running assemblers and scaffolders on the tandem duplication 1 data set.

	Contigs		Coverage					Errors		
Asm	N	N50	MR	CR	DR	Ch	MA	S	I	SV
D	6 / 6 / 0	10123	34	1020	329	727	0	0	0 / 0	0 / 0 / 0
P^S	1 / 1 / 0	22498	34	0	0	0	0	0	0 / 0	0 / 0 / 0
A^A	1 / 3 / 0	19569	811	2654	0	0	0	1	0 / 1	0 / 1 / 0
Sg^A	6 / 6 / 0	9937	195	1188	2638	2995	0	0	0 / 0	0 / 0 / 0
So^A	3 / 5 / 0	22166	7	1335	1808	2621	0	0	0 / 0	0 / 0 / 0
Sg^S	6 / 6 / 0	10123	34	1020	329	727	0	0	0 / 0	0 / 0 / 0
So^S	4 / 4 / 0	21734	109	788	1171	554	799	7	18 / 0	0 / 0 / 0

(c) Running assemblers and scaffolders on the tandem duplication 2 data set.

	Contigs		Coverage					Errors		
Asm	N	N50	MR	CR	DR	Ch	MA	S	I	SV
D	102 / 82 / 0	11476	27	19774	18346	15742	1370	173	76 / 4	0 / 0 / 0
P^S	1 / 1 / 0	249149	0	0	0	0	0	0	0 / 0	0 / 0 / 0
A^A	4 / 10 / 0	88902	32648	5039	0	0	1	14	0 / 3	0 / 1 / 0
Sg^S	22 / 22 / 0	125657	0	14271	637	1035	0	0	0 / 0	0 / 0 / 0
So^S	13 / 14 / 0	231765	219	16025	97336	1184	7185	2	222 / 0	0 / 0 / 0

(d) Running assemblers and scaffolders on the nebulin data set.

Table 3.1: Evaluation results using the GAGE evaluation script [Salzberg et al., 2012]. Please see Table 3.2 for the legend.

Abbrev.	Description
Asm	Assemblies
D	original Discovar contigs
P	Piper
A	ALLPATHS-LG
Sg	SGA
So	SOAPdenovo2
	Contigs
N	num. of final scaffolds / post-correction contigs or scaffolds / missing assembly contigs or scaffolds
N50	N50 statistic for final scaffolds
	Coverage
MR	missing reference bases
CR	compressed reference bases
DR	duplicated reference bases
Ch	chaff bases
MA	missing assembly bases
	Errors
S	SNPs
I	indels < 5 bp / indels ≥ 5 bp
SV	inversions / relocations / translocations

Table 3.2: Legend for the GAGE evaluation [Salzberg et al., 2012] results table. Note that, in Table 3.1, programs marked with superscript A assemble their own contigs, while programs marked with superscript S use the contigs from Discovar.

3.4 Conclusion

We have proposed an algorithm for generating a set of high-likelihood assemblies that are consistent with an input contig graph and paired-read data. When the problem size is small enough for the algorithm to run in at most a few minutes, we found that our algorithm was able to produce a list of assemblies that always included the true assembly. Moreover, in many cases, a single solution was found with higher likelihood score than the other assemblies, even when other assemblers produced multiple contigs. This suggests that frequently the data are sufficient for assembly, and the obstacle to assembly is a computational one.

The main challenge is to improve the run-time of the algorithm so that it scales to large problems. One possible approach is to apply additional heuristics in order to rule out more assemblies early in the search process. Another possibility is to apply a divide and conquer approach by partitioning the problem, either into independent or overlapping components, which can be optimized separately and then merged. Partitioning the problem into independent components is not a new idea, e.g. Dayarian et al. [2010] observe, "If the removal of a contig divides the scaffolding graph into two components, then the scaffolding can be solved independently for these two components, both of which also include the removed node."

An interesting open question centers on how the assembly performance varies with different library features, such as the mean and variance in insert size, the read length, and the read number. Evaluating the change in reconstruction ability as a function of these parameters would be interesting, both for the likelihood-based approach, and for more traditional assemblers. In general, it seems that a larger variance in insert size will negatively affect the efficiency with which heuristics can be used to discard assemblies, but potentially improve the resolution with which the likelihood score can distinguish competing assemblies.

In order to have practical use, our algorithm needs to be tested and validated on real data. Whereas our simulated data was haploid, real genomes are diploid. Also, real sequence data have systematic sequencing errors, and sequence-dependent coverage, which we do not model. Furthermore, real data includes reads from the whole genome, unlike our simulated data, where we only had reads from the relatively small region of interest. The incorporation of additional heuristics as well as further testing are necessary to make the algorithm practical for resolving segmental duplications.

Despite its current limitations, this assembly method shows superior performance on small datasets, with simulated data: it can often resolve segmental duplications that other algorithms cannot.

Chapter 4

Information-Theoretic Requirements for Perfect Assembly

4.1 Introduction

We present a framework for the design of optimal assembly algorithms for shotgun sequencing under the criterion of complete reconstruction. We derive a lower bound on the read length and the coverage depth required for reconstruction in terms of the repeat statistics of the genome. Building on earlier works, we design a de Brujin graph based assembly algorithm which can achieve very close to the lower bound for repeat statistics of a wide range of sequenced genomes, including the GAGE datasets [Salzberg et al., 2012]. The results are based on a set of necessary and sufficient conditions on the DNA sequence and the reads for reconstruction. The conditions can be viewed as the shotgun sequencing analogue of Ukkonen-Pevzner's necessary and sufficient conditions for Sequencing by Hybridization [Pevzner, 1995, Ukkonen, 1992].

4.2 Approach

The difficulty of comparing algorithms is evidenced by the recent assembly evaluations Assemblathon 1 [Earl et al., 2011] and GAGE [Salzberg et al., 2012], where which assembler is "best" depends on the particular dataset as well as the performance metric used. In part this is a consequence of metrics for partial assemblies: there is an inherent tradeoff between larger contiguous fragments (contigs) and fewer mistakes in merging contigs (misjoins). But more fundamentally, independent of the metric, performance depends critically on the dataset, i.e. length, number, and quality of the reads, as well as the complexity of the genome sequence. With an eye towards the near future, we seek to understand the interplay between these factors by using the intuitive and unambiguous metric of *complete* reconstruction[1]. Note

[1]The notion of complete reconstruction can be thought of as a mathematical idealization of the notion of "finishing" a sequencing project as defined by the National Human Genome Research Institute [National

that this objective of reconstructing the original DNA sequence from the reads contrasts with the many *optimization-based* formulations of assembly, such as shortest common superstring (SCS) [Kececioglu and Myers, 1995], maximum-likelihood [Myers, 1995], Medvedev and Brudno [2009], and various graph-based formulations Pevzner et al. [2001], Myers [2005]. When solving one of these alternative formulations, there is no guarantee that the optimal solution is indeed the original sequence.

Given the goal of complete reconstruction, the most basic questions are 1) **feasibility**: given a set of reads, is it *possible* to reconstruct the original sequence? 2) **optimality**: which *algorithms* can successfully reconstruct whenever it is feasible to reconstruct? The feasibility question is a measure of the intrinsic *information* each read provides about the DNA sequence, and for given sequence statistics depends on characteristics of the sequencing technology such as read length and noise statistics. As such, it can provide an algorithm-independent basis for evaluating the efficiency of a sequencing technology. Equally important, algorithms can be evaluated on their relative read length and data requirements, and compared against the fundamental limit.

In studying these questions, we consider the most basic shotgun sequencing model where N noiseless reads[2] of a fixed length L base pairs are uniformly and independently drawn from a DNA sequence of length G. In this statistical model, feasibility is rephrased as the question of whether, for given sequence statistics, the correct sequence can be reconstructed with probability $1 - \epsilon$ when N reads of length L are sampled from the genome. We note that answering the feasibility question of whether each N, L pair is sufficient to reconstruct is equivalent to finding the minimum required N (or the *coverage depth* $c = NL/G$) as a function of L.

A lower bound on the minimum coverage depth needed was obtained by Lander and Waterman [1988]. Their lower bound $c_{\text{LW}} = c_{\text{LW}}(L, \epsilon)$ is the minimum number of randomly located reads needed to cover the entire DNA sequence with a given target success probability $1 - \epsilon$. While this is clearly a necessary condition, it is in general not tight: only requiring the reads to cover the entire genome sequence does not guarantee that consecutive reads can actually be stitched back together to recover the original sequence. Characterizing when the reads can be reliably stitched together, i.e. determining feasibility, is an open problem. In fact, the ability to reconstruct depends crucially on the *repeat statistics* of the DNA sequence.

An earlier work Motahari et al. [2012] has answered the feasibility and optimality questions under an i.i.d. model for the DNA sequence. However, real DNA, especially those of eukaryotes, have much longer and complex repeat structures. Here, we are interested in determining feasibility and optimality given *arbitrary* repeat statistics. This allows us to evaluate algorithms on statistics from already sequenced genomes, and gives confidence in predicting whether the algorithms will be useful for an *unseen* genome with similar statistics.

Our approach results in a pipeline, which takes as input a genome sequence and desired

Human Genome Research Institute, NIH, 2012], where finishing a chromosome requires at least 95% of the chromosome to be represented by a contiguous sequence.

[2]Reads are thus exact subsequences of the DNA.

success probability $1 - \epsilon$, computes a few simple repeat statistics, and from these statistics computes a feasibility plot that indicates for which L, N reconstruction is possible. Fig. 4.1 displays the simplest of the statistics, the number of repeats as a function of the repeat length ℓ. Fig. 4.2 shows the resulting feasibility plot produced for the statistics of human chromosome 19 (henceforth hc19) with success probability 99%. The horizontal axis signifies read length L and the vertical axis signifies the normalized coverage depth $\bar{c} := c/c_{\text{LW}}$, the coverage depth c normalized by c_{LW}, the coverage depth required as per Lander and Waterman [1988] in order to cover the sequence.

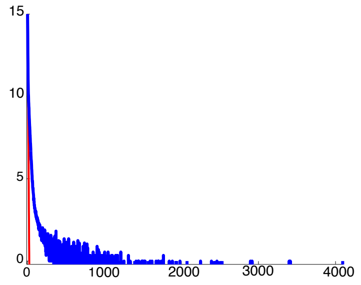


Figure 4.1: For hc19, a log plot of number of repeats as a function of the repeat length ℓ. Red line is what would have been predicted by an i.i.d. fit.

Since the coverage depth must satisfy $c \geq c_{\text{LW}}$, the normalized coverage depth satisfies $\bar{c} \geq 1$, and we plot the horizontal line $\bar{c} = 1$. This lower bound holds for *any* assembly algorithm. In addition, there is another lower bound, shown as the thick black nearly vertical

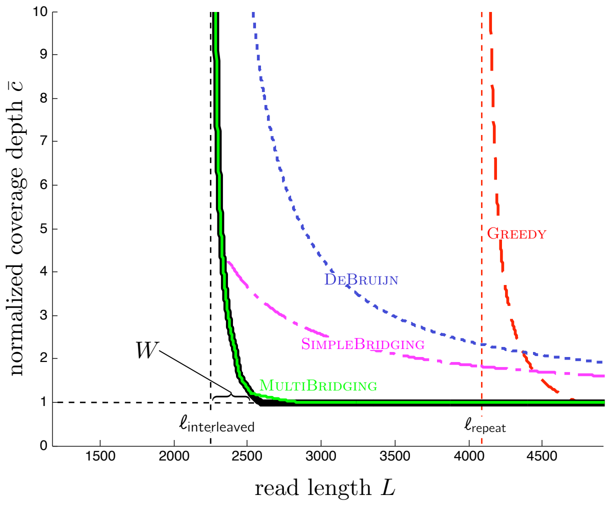


Figure 4.2: Thick black lines are lower bounds on feasibility which holds for all algorithms, and colored curves are performance achieved by specific algorithms. Four such curves are shown: the greedy algorithm and three de Brujin graph based algorithms.

line in Fig. 4.2. In contrast to the coverage lower bound, this lower bound is a function of the repeat statistics. It has a vertical asymptote at $L_{\text{crit}} := \max\{\ell_{\text{interleaved}}, \ell_{\text{triple}}\} + 1$, where $\ell_{\text{interleaved}}$ is the length of the longest interleaved repeats in the DNA sequence and ℓ_{triple} is the length of the longest triple repeat (see Section 4.3 for precise definitions). Our lower bound can be viewed as a generalization of a result of Ukkonen [1992] for Sequencing by Hybridization to the shotgun sequencing setting.

Each colored curve in the feasibility plot is the lower boundary of the set of feasible N, L pairs for a specific algorithm. The rightmost curve is the one achieved by the greedy algorithm, which merges reads with largest overlaps first (used for example in TIGR [Sutton et al., 1995], CAP3 [Huang, 1999], and more recently SSAKE [Warren et al., 2007]). As seen in Fig. 4.2, its performance curve asymptotes at $L = \ell_{\text{repeat}}$, the length of the longest repeat. De Brujin graph based algorithms (e.g. Idury and Waterman [1995] and Pevzner et al. [2001]) take a more global view via the construction of a de Brujin graph out of all the K-mers of the reads. The performance curves of all K-mer graph based algorithms asymptote at read length $L = L_{\text{crit}}$, but different algorithms use read information in a variety of ways to resolve repeats in the K-mer graph and thus have different coverage depth requirement beyond read length L_{crit}. By combining the ideas from several existing algorithms (including Pevzner et al. [2001], Peng et al. [2010]) we designed MultiBridging, which is very close to the lower bound for this dataset. Thus Fig. 4.2 answers, up to a very small gap, the feasibility of assembly for the repeat statistics of hc19, where successful reconstruction is desired with probability 99%.

We produce similar plots for a dozen or so datasets (see supplementary material). For datasets where $\ell_{\text{interleaved}}$ is significantly larger than ℓ_{triple} (the majority of the datasets we looked at, including those used in the recent GAGE assembly algorithm evaluation Salzberg et al. [2012]), MultiBridging is near optimal, thus allowing us to characterize the fundamental limits for these repeat statistics (Fig. 4.9). On the other hand, if ℓ_{triple} is close to or larger than $\ell_{\text{interleaved}}$, there is a gap between the performance of MultiBridging and the lower bound (see for example Fig. 4.3). The reason for the gap is explained in Section 4.4.

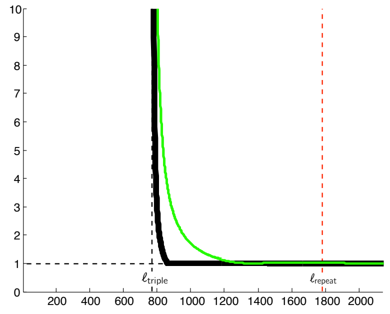


Figure 4.3: Performance of MultiBridging on *P Marinus*, where $\ell_{\text{triple}} > \ell_{\text{interleaved}}$.

An interesting feature of the feasibility plots is that for typical repeat statistics exhibited by DNA data, the minimum coverage depth is characterized by a *critical phenomenon*: If the read length L is below $L_{\text{crit}} = \ell_{\text{interleaved}}$, reliable reconstruction of the DNA sequence is impossible no matter what the coverage depth is, but if the read length L is slightly above L_{crit}, then covering the sequence suffices, i.e. $\bar{c} = c/c_{\text{LW}} = 1$. The sharpness of the critical phenomenon is described by the size of the *critical window*, which refers to the range of L over which the transition from one regime to the other occurs. For the case when MultiBridging is near optimal, the width W of the window size can be well approximated as:

$$W \approx \frac{L_{\text{crit}}}{2r+1}, \quad \text{where } r := \frac{\log \frac{G}{L_{\text{crit}}}}{\log \epsilon^{-1}} . \tag{4.1}$$

For the hc19 dataset, the critical window size evaluates to about 19% of L_{crit}.

In Sections 4.3 and 4.4, we discuss the underlying analysis and algorithm design supporting the plots. The curves are all computed from formulas, which are validated by simulations in Section 4.5. We return in Section 4.6 to put our contributions in a broader perspective and discuss extensions to the basic framework. All proofs can be found in the appendix.

4.3 Lower bounds

In this section we discuss lower bounds, due to coverage analysis and certain repeat patterns, on the required coverage depth and read length. The style of analysis here is continued in Section 4.4, in which we search for an assembly algorithm that performs close to the lower bounds.

Coverage bound

Lander and Waterman's coverage analysis [Lander and Waterman, 1988] gives the well known condition for the number of reads N_{LW} required to cover the entire DNA sequence with probability at least $1-\epsilon$. In the regime when $L \ll G$, one may make the standard assumption that the starting locations of the N reads follow a Poisson process with rate $\lambda = N/G$, and the number N_{LW} is to a very good approximation given by the solution to the equation

$$N_{\text{LW}} = \frac{G}{L} \log \frac{N_{\text{LW}}}{\epsilon} . \tag{4.2}$$

The corresponding coverage depth is $c_{\text{LW}} = N_{\text{LW}} L/G$. This is our baseline coverage depth against which to compare the coverage depth of various algorithms. For each algorithm, we will plot

$$\bar{c} := \frac{c}{c_{\text{LW}}} = \frac{N}{N_{\text{LW}}} ,$$

the coverage depth required by that algorithm normalized by c_{LW}. Note that $\bar{c}$ is also the ratio of the number of reads N required by an algorithm to N_{LW}. The requirement $\bar{c} \geq 1$ is due to the lower bound on the number of reads obtained by the Lander-Waterman coverage condition.

Ukkonen's condition

A second constraint on reads arises from repeats. A lower bound on the read length L follows from Ukkonen's condition [Ukkonen, 1992]: if there are *interleaved repeats* or *triple repeats* in the sequence of length at least $L-1$, then the likelihood of observing the reads is the same for more than one possible DNA sequence and hence correct reconstruction is not possible. Fig. 4.4 shows an example with interleaved repeats. (Note that we assume $1-\epsilon > 1/2$, so random guessing between equally likely sequences is not viable.)

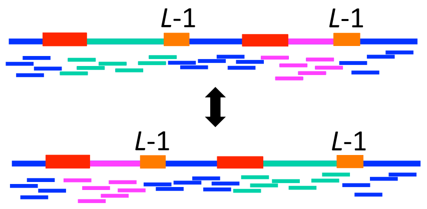


Figure 4.4: The likelihood of observing the reads under two possible sequences (the green and magenta segments swapped) is the same. Here, the two red subsequences form a repeat and the two orange subsequences form another repeat.

We take a moment to carefully define the various types of repeats. Let $\mathbf{s}_t^\ell$ denote the length-ℓ subsequence of the DNA sequence $\mathbf{s}$ starting at position t. A *repeat* of length ℓ is a subsequence appearing twice, at some positions t_1, t_2 (so $\mathbf{s}_{t_1}^\ell = \mathbf{s}_{t_2}^\ell$) that is maximal (i.e. $s(t_1-1) \neq s(t_2-1)$ and $s(t_1+\ell) \neq s(t_2+\ell)$). Similarly, a *triple repeat* of length ℓ is a subsequence appearing three times, at positions t_1, t_2, t_3, such that $\mathbf{s}_{t_1}^\ell = \mathbf{s}_{t_2}^\ell = \mathbf{s}_{t_3}^\ell$, and such that neither of $s(t_1-1) = s(t_2-1) = s(t_3-1)$ nor $s(t_1+\ell) = s(t_2+\ell) = s(t_3+\ell)$ holds[3]. A *copy* is a single one of the instances of the subsequence's appearances. A *pair* of repeats refers to two repeats, each having two copies. A pair of repeats, one at positions t_1, t_3 with $t_1 < t_3$ and the second at positions t_2, t_4 with $t_2 < t_4$, is *interleaved* if $t_1 < t_2 < t_3 < t_4$ or $t_2 < t_1 < t_4 < t_3$ (Fig. 4.4). The length of a pair of interleaved repeats is defined to be the length of the shorter of the two repeats.

Ukkonen's condition implies a lower bound on the read length,

$$L > L_{\text{crit}} := \max\{\ell_{\text{interleaved}}, \ell_{\text{triple}}\} + 1\,.$$

Here $\ell_{\text{interleaved}}$ is the length of the longest pair of interleaved repeats on the DNA sequence and ℓ_{triple} is the length of the longest triple repeat.

Ukkonen's condition says that for read lengths less than L_{crit}, reconstruction is impossible no matter what the coverage depth is. But it can be generalized to provide a lower bound on the coverage depth for read lengths greater than L_{crit}, through the important concept of *bridging* as shown in Figure 4.5. We observe that in Ukkonen's interleaved or triple repeats, the actual length of the repeated subsequences is irrelevant; rather, to cause confusion it is enough that all the copies of the pertinent repeats are unbridged. This leads to the following theorem.

[3]Note that a subsequence that is repeated f times gives rise to $\binom{f}{2}$ repeats and $\binom{f}{3}$ triple repeats.

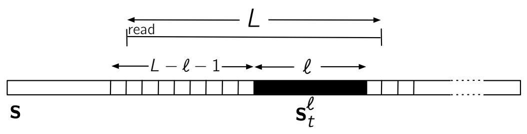


Figure 4.5: A subsequence $\mathbf{s}_t^\ell$ is bridged if and only if there exists at least one read which covers at least one base on both sides of the subsequence, i.e. the read arrives in the preceding length $L - \ell - 1$ interval.

Theorem 1. *Given a DNA sequence* $\mathbf{s}$ *and a set of reads, if there is a pair of interleaved repeats or a triple repeat whose copies are all unbridged, then there is another sequence* $\mathbf{s}'$ *of the same length under which the likelihood of observing the reads is the same.*

For brevity, we will call a repeat or a triple repeat *bridged* if at least one copy of the repeat is bridged, and a pair of interleaved repeats *bridged* if at least one of the repeats is bridged. Thus, the above theorem says that a necessary condition for reconstruction is that all interleaved and triple repeats are bridged.

How does Theorem 1 imply a lower bound on the coverage depth? Focus on the longest pair of interleaved repeats and suppose the read length L is between the lengths of the shorter and the longer repeats. The probability this pair is unbridged is $(p_{\ell_{\text{interleaved}}}^{\text{unbridged}})^2$, where

$$
\begin{aligned}
p_\ell^{\text{unbridged}} &:= \mathbb{P}[\ell\text{-length subseq. is unbridged}] \\
&= e^{\frac{N}{G}(L-\ell-1)^+}. \qquad (4.3)
\end{aligned}
$$

Theorem 1 implies that the probability of making an error in the reconstruction is at least 1/2 if this event occurs. Hence, the requirement that $P_{\text{error}} \leq \epsilon$ implies a lower bound on the number of reads N:

$$
N \geq \frac{G}{(L - \ell_{\text{interleaved}} - 1)\ln(1/(2\epsilon))}. \qquad (4.4)
$$

A similar lower bound can be derived using the longest triple repeat. A slightly tighter lower bound can be obtained by taking into consideration the bridging of *all* the interleaved and triple repeats, not only the longest one, resulting in the black curve in Fig. 4.2.

4.4 Towards optimal assembly

We now begin our search for algorithms performing close to the lower bounds derived in the previous section. Algorithm assessment begins with obtaining deterministic sufficient conditions for success in terms of repeat-bridging. We then find the necessary N and L in order to satisfy these sufficient conditions with a target probability $1 - \epsilon$. The required coverage depth for each algorithm depends only on certain repeat statistics extracted from the DNA data, which may be thought of as *sufficient statistics*.

Greedy algorithm

The greedy algorithm, denoted Greedy, with pseudocode in section B.3, is described as follows. Starting with the initial set of reads, the two fragments (i.e. subsequences) with maximum length overlap are merged, and this operation is repeated until a single fragment remains. Here the overlap of two fragments $\mathbf{x}, \mathbf{y}$ is a suffix of $\mathbf{x}$ equal to a prefix of $\mathbf{y}$, and merging two fragments results in a single longer fragment.

Theorem 2. Greedy *reconstructs the original sequence* $\mathbf{s}$ *if every repeat is bridged.*

Theorem 2 allows us to determine the coverage depth required by Greedy: we must ensure that all repeats are bridged. By the union bound,

$$\mathbb{P}[\text{some repeat is unbridged}] \leq \sum_m a_m \left(p_m^{\text{unbridged}}\right)^2 , \tag{4.5}$$

where $p_m^{\text{unbridged}}$ is defined in (4.3) and a_m is the number of repeats of length m. Setting the right-hand side of (4.5) to ϵ ensures $P_{\text{error}} \leq \epsilon$ and yields the performance curve of Greedy in Fig. 4.2. Note that the repeat statistics $\{a_m\}$ are sufficient to compute this curve.

Greedy requires $L > \ell_{\text{repeat}} + 1$, whereas the lower bound has its asymptote at $L = \ell_{\text{interleaved}} + 1$. In chromosome 19, for instance, there is a large difference between $\ell_{\text{interleaved}} = 2248$ and $\ell_{\text{repeat}} = 4092$, and in Fig 4.2 we see a correspondingly large gap. Greedy is evidently sub-optimal in handling interleaved repeats. Its strength, however, is that once the reads are slightly longer than ℓ_{repeat}, coverage of the sequence is sufficient for correct reconstruction. Thus if $\ell_{\text{repeat}} \approx \ell_{\text{interleaved}}$, then Greedy is close to optimal.

K-mer algorithms

The greedy algorithm fails when there are unbridged repeats, even if there are no unbridged *interleaved* repeats, and therefore requires a read length much longer than that required by Ukkonen's condition. As we will see, K-mer algorithms do not have this limitation.

Background

In the introduction we mention Sequencing By Hybridization (SBH), for which Ukkonen's condition was originally introduced. In the SBH setting, an optimal algorithm matching Ukkonen's condition is known, due to Pevzner [1995].

Pevzner's algorithm is based on finding an appropriate cycle in a K-mer graph (also known as a de Bruijn graph) with $K = L - 1$ (see e.g. Compeau et al. [2011] for an overview). A K-mer graph is formed by first creating a node in the graph for each unique K-mer (length K subsequence) in the set of reads, and then adding an edge with overlap $K-1$ between any two nodes representing K-mers that are *adjacent* in a read, i.e. offset by a single nucleotide. Edges thus correspond to unique $(K+1)$-mers in $\mathbf{s}$ and paths correspond to longer subsequences obtained by merging the constituent nodes. There exists a cycle corresponding to the original sequence $\mathbf{s}$, and reconstruction entails finding this cycle.

As is common, we will replace edges corresponding to an unambiguous path by a single node (c.f. Fig. 4.6). Since the subsequences at some nodes are now longer than K, this is no longer a K-mer graph, and we call the more general graph a sequence graph. The simplified graph is called the *condensed sequence graph*.

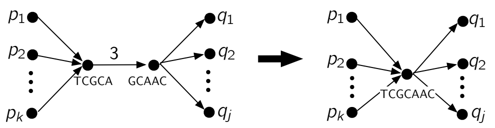


Figure 4.6: Contracting an edge by merging the incident nodes. Repeating this operation results in the condensed graph.

The condensed graph has the useful property that if the original sequence $\mathbf{s}$ is reconstructible, then $\mathbf{s}$ is determined by a unique Eulerian cycle:

Theorem 3. *Let $\mathbb{G}_0$ be the K-mer graph constructed from the $(K+1)$-spectrum $\mathcal{S}_{K+1}$ of $\mathbf{s}$, and let $\mathbb{G}$ be the condensed sequence graph obtained from $\mathbb{G}_0$. If Ukkonen's condition is satisfied, i.e. there are no triple or interleaved repeats of length at least K, then there is a unique Eulerian cycle $\mathcal{C}$ in $\mathbb{G}$ and $\mathcal{C}$ corresponds to $\mathbf{s}$.*

Theorem 3 characterizes, deterministically, the values of K for which reconstruction from the $(K+1)$-spectrum is possible. We proceed with application of the K-mer graph approach to shotgun sequencing data.

Basic K-mer algorithm

Starting with Idury and Waterman [1995], and then EULER algorithm of Pevzner et al. [2001], most current assembly algorithms for shotgun sequencing are based on the K-mer graph. Idury and Waterman [1995] made the key observation that SBH with subsequences of length $K+1$ can be *emulated* by shotgun sequencing if each read overlaps the subsequent read by K: the set of all $(K+1)$-mers within the reads is equal to the $(K+1)$-spectrum $\mathcal{S}_{K+1}$. The resultant algorithm DEBRUIJN which consists of constructing the K-mer graph from the $(K+1)$-spectrum observed in the reads, condensing the graph, and then identifying an Eulerian cycle, has sufficient conditions for correct reconstruction as follows.

Theorem 4. DEBRUIJN *with parameter choice K reconstructs the original sequence* $\mathbf{s}$ *if:*

(a) $K > \ell_{interleaved}$

(b) $K > \ell_{triple}$

(c) adjacent reads overlap by at least K

Lander and Waterman's coverage analysis applies also to Condition (c) of Theorem 4, yielding a normalized coverage depth requirement $\bar{c} = 1/(1 - K/L)$. The larger the overlap K, the higher the coverage depth required. Conditions (a) and (b) say that the smallest K one can choose is $K = \max\{\ell_{\text{triple}}, \ell_{\text{interleaved}}\} + 1$, so

$$\bar{c} = \frac{1}{1 - \frac{\max\{\ell_{\text{triple}}, \ell_{\text{interleaved}}\}+1}{L}} \, . \tag{4.6}$$

The performance of DeBruijn is plotted in Fig. 4.2. DeBruijn significantly improves on Greedy by obtaining the correct first order performance: given sufficiently many reads, the read length L may be decreased to $\max\{\ell_{\text{triple}}, \ell_{\text{interleaved}}\} + 1$. Still, the number of reads required to approach this critical length is far above the lower bound. The following subsection pursues reducing K in order to reduce the required number of reads.

Improved K-mer algorithms

Algorithm DeBruijn ignores a lot of information contained in the reads, and indeed all of the K-mer based algorithms proposed by the sequencing community (including Idury and Waterman [1995], Pevzner et al. [2001], Simpson et al. [2009], Gnerre et al. [2011], MacCallum et al. [2009], Zerbino and Birney [2008]) use the read information to a greater extent than the naive DeBruijn algorithm. Better use of the read information, as described below in algorithms SimpleBridging and MultiBridging, will allow us to relax the condition $K > \max\{\ell_{\text{interleaved}}, \ell_{\text{triple}}\}$ for success of DeBruijn, which in turn reduces the high coverage depth required by Condition (c).

Existing algorithms use read information in a variety of distinct ways to resolve repeats. For instance, Pevzner et al. [2001] observe that for graphs where each edge has multiplicity one, if one copy of a repeat is bridged, the repeat can be resolved through what they call a "detachment". The algorithm SimpleBridging described below is very similar, and resolves repeats with two copies if at least one copy is bridged.

Meanwhile, other algorithms are better suited to higher edge multiplicities due to higher order repeats; IDBA (Iterative DeBruijn Assembler) Peng et al. [2010] creates a series of K-mer graphs, each with larger K, and at each step uses not just the reads to identify adjacent K-mers, but also all the unbridged paths in the K-mer graph with smaller K. Although not stated explicitly in their paper, we observe here that if all copies of every repeat are bridged, then IDBA correctly reconstructs.

However, it is suboptimal to require that *all* copies of every repeat up to the maximal K be bridged. We introduce MultiBridging, which combines the aforementioned ideas to simultaneously allow for single-bridged double repeats, triple repeats in which all copies are bridged, and unbridged non-interleaved repeats.

SimpleBridging

SimpleBridging improves on DeBruijn by resolving bridged 2-repeats (i.e. a repeat with exactly two copies in which at least one copy is bridged by a read). Condition (a)

$K > \ell_{\text{interleaved}}$ for success of DeBruijn (ensuring that no interleaved repeats appear in the initial K-mer graph) is updated to require only no *unbridged* interleaved repeats, which matches the lower bound. With this change, Condition (b) $K > \ell_{\text{triple}}$ forms the bottleneck for typical DNA sequences. Thus SimpleBridging is optimal with respect to interleaved repeats, but it is suboptimal with respect to triple repeats.

SimpleBridging deals with repeats by performing surgery on certain nodes in the sequence graph. In the sequence graph, a repeat corresponds to a node we call an *X-node*, a node with in-degree and out-degree each at least two (e.g. Fig. 4.7). A self-loop adds one each to the in-degree and out-degree. The cycle $\mathcal{C}(\mathbf{s})$ traverses each X-node at least twice, so X-nodes correspond to repeats in $\mathbf{s}$. We call an X-node traversed exactly twice a 2-X-node; these nodes correspond to 2-repeats, and are said to be bridged if the corresponding repeat in $\mathbf{s}$ is bridged.

In the repeat resolution step of SimpleBridging (illustrated in Fig. 4.7), bridged 2-X-nodesare duplicated in the graph and incoming and outgoing edges are inferred using the bridging read, reducing possible ambiguity.

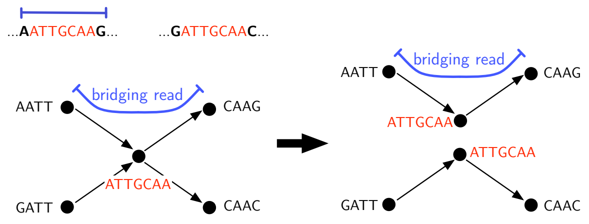


Figure 4.7: An example of the bridging step in SimpleBridging.

Theorem 5. SimpleBridging *with parameter choice K reconstructs the original sequence* $\mathbf{s}$ *if:*

(a) all interleaved repeats are bridged

(b) $K > \ell_{triple}$

(c) adjacent reads overlap by at least K.

By the union bound,

$$\begin{aligned} &\mathbb{P}[\text{some interleaved repeat is unbridged}] \\ &\leq \sum_{m,n} b_{m,n} \left(p_m^{\text{unbridged}}\right)^2 \left(p_n^{\text{unbridged}}\right)^2 \end{aligned} \tag{4.7}$$

where $b_{m,n}$ is the number of interleaved repeats in which one repeat is of length m and the other is of length n. To ensure that condition (a) in the above theorem fails with probability no more than ϵ, the right hand side of (4.7) is set to be ϵ; this imposes a constraint on the coverage depth. Furthermore, conditions (b) and (c) imply that the normalized coverage depth $\bar{c} \geq 1/(1-(\ell_{\text{triple}}+1)/L)$. These two constraints together yield the performance curve of SimpleBridging in Figure 4.2.

MultiBridging

We now turn to triple repeats. As previously observed, it can be challenging to resolve repeats with more than one copy [Pevzner et al., 2001], because an edge into the repeat may be paired with more than one outgoing edge. As discussed above, our approach here shares elements with IDBA [Peng et al., 2010]: we note that increasing the node length serves to resolve repeats. Unlike IDBA, we do not increase the node length globally.

As noted in the previous subsection, repeats correspond to nodes in the sequence graph we call *X-nodes*. Here the converse is false: not all repeats correspond to X-nodes. A repeat is said to be *all-bridged* if *all* repeat copies are bridged, and an X-node is called all-bridged if the corresponding repeat is all-bridged.

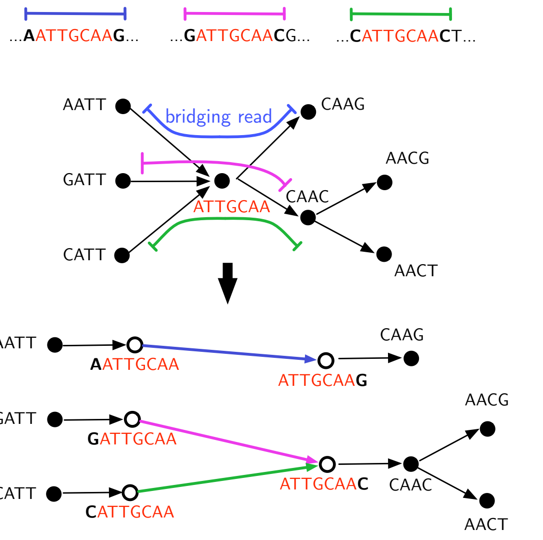


Figure 4.8: MultiBridging resolves an X-node with label ATTGCAA corresponding to a triple repeat.

The requirement that triple repeats be all-bridged allows them to be resolved *locally* (Fig. 4.8). The X-node resolution procedure given in Step 4 of MultiBridging can be interpreted in the K-mer graph framework as increasing K locally so that repeats do not appear in the graph. In order to do this, we introduce the following notation for extending nodes: Given an edge $(\mathbf{v}, \mathbf{q})$ with weight $a_{\mathbf{v},\mathbf{q}}$, let $\mathbf{v}^{\rightarrow \mathbf{q}}$ denote $\mathbf{v}$ extended one base to the right along $(\mathbf{v}, \mathbf{q})$, i.e. $\mathbf{v}^{\rightarrow \mathbf{q}} = \mathbf{v}\,\mathbf{q}^{1}_{a_{\mathbf{vq}}+1}$ (notation introduced in Sec. 4.3). Similarly, let ${}^{\mathbf{p}\rightarrow}\mathbf{v} = \mathbf{p}^{1}_{\text{end}-a_{\mathbf{pv}}}\,\mathbf{v}$. MultiBridging is described as follows.

Algorithm 1 MultiBridging. Input: reads $\mathcal{R}$, parameter K. Output: sequence $\hat{\mathbf{s}}$.

```
K-mer steps 1-3:
1. For each subsequence $\mathbf{x}$ of length $K$ in a read, form a node with label $\mathbf{x}$.
2. For each read, add edges between nodes representing adjacent $K$-mers in the read.
3. Condense the graph (c.f. Fig. 4.6).
4. Bridging step: (See Fig. 4.8). While there exists a bridged X-node $\mathbf{v}$: (i) For each edge $(\mathbf{p}_i, \mathbf{v})$ with weight $a_{\mathbf{p}_i,\mathbf{v}}$, create a new node $\mathbf{u}_i = {}^{\mathbf{p}_i\to}\mathbf{v}$ and an edge $(\mathbf{p}_i, \mathbf{u}_i)$ with weight $1 + a_{\mathbf{p}_i,\mathbf{v}}$. Similarly for each edge $(\mathbf{v}, \mathbf{q}_j)$, create a new node $\mathbf{w}_j = \mathbf{v}^{\to\mathbf{q}_j}$ and edge $(\mathbf{w}_j, \mathbf{q}_j)$. (ii) If $\mathbf{v}$ has a self-loop $(\mathbf{v}, \mathbf{v})$ with weight $a_{\mathbf{v},\mathbf{v}}$, add an edge $(\mathbf{v}^{\to\mathbf{v}}, {}^{\mathbf{v}\to}\mathbf{v})$ with weight $a_{\mathbf{v},\mathbf{v}} + 2$. (iii) Remove node $\mathbf{v}$ and all incident edges. (iv) For each pair $\mathbf{u}_i, \mathbf{w}_j$ adjacent in a read, add edge $(\mathbf{u}_i, \mathbf{w}_j)$. If exactly one each of the $\mathbf{u}_i$ and $\mathbf{w}_j$ nodes have no added edge, add the edge. (v) Condense graph.
5. Finishing step: Find an Eulerian cycle in the graph and return the corresponding sequence.
```

Theorem 6. *The algorithm* MultiBridging *reconstructs the sequence* $\mathbf{s}$ *if:*

(a) all interleaved repeats are bridged

(b) all triple repeats are ***all-bridged***

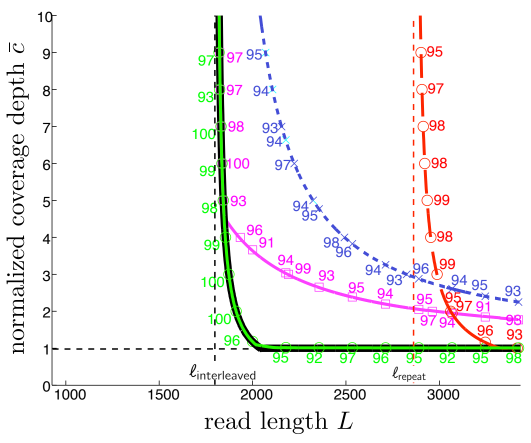
(c) the sequence is covered by the reads.

A similar analysis as for SimpleBridging yields the performance curve of MultiBridging in Figure 4.2.

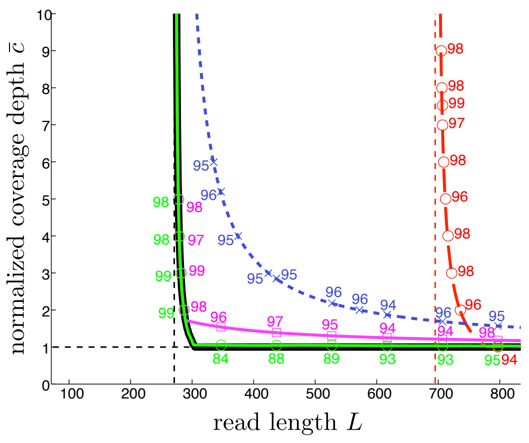
Gap to lower bound

The only difference between the sufficient condition guaranteeing the success of MultiBridging and the necessary condition of the lower bound is the bridging condition of *triple* repeats: while MultiBridging requires bridging *all three copies* of the triple repeats, the necessary condition requires only bridging *a single copy.* When ℓ_{triple} is significantly smaller than $\ell_{\text{interleaved}}$, the bridging requirement of interleaved repeats dominates over that of triple repeats and MultiBridging achieves very close to the lower bound. This occurs in hc19 and the majority of the datasets we looked at. (See Fig. 4.9 and the plots in the supplementary material.) A critical phenomenon occurs as L increases: for $L < L_{\text{crit}}$ reconstruction is impossible, over a small critical window the bridging requirement of interleaved repeats (primarily the longest) dominates, and then for larger L, coverage suffices.

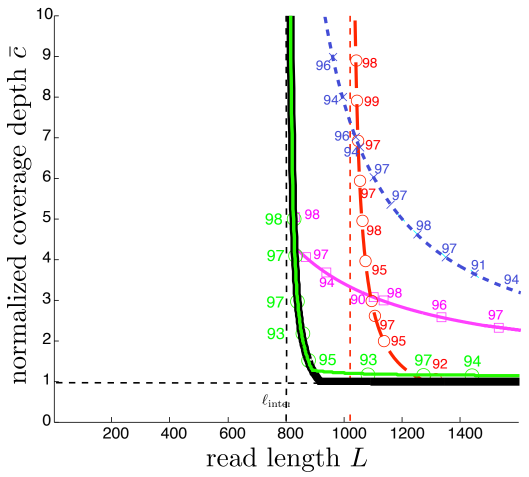
On the other hand, when ℓ_{triple} is comparable or larger than $\ell_{\text{interleaved}}$, then MultiBridging has a gap in the coverage depth to the lower bound (see for example Fig. 4.3). If we further assume that the longest triple repeat is dominant, then this gap can be calculated to be a factor of $3 \cdot \frac{\log 3\epsilon^{-1}}{\log \epsilon^{-1}} \approx 3.72 \quad \text{for } \epsilon = 10^{-2}$. This gap occurs only within the critical window where the repeat-bridging constraint is active. Beyond the critical window, the coverage constraint dominates and MultiBridging is optimal. Further details are provided in the appendices.



(a) S. Aureus



(b) R. sphaeroides



(c) hc14

Figure 4.9: Simulation results for each of the GAGE reference genomes. Each simulated (N, L) point is marked with the number of correct reconstructions (e.g. 93, 98, 95) on 100 simulated read sets. All four algorithms (Greedy, DeBruijn, SimpleBridging, and MultiBridging) were run on *S. Aureus*, *R. sphaeroides* and hc14. Note that MultiBridging is very close to the lower bound on all 3 datasets.

4.5 Simulations and complexity

In order to verify performance predictions, we implemented and ran the algorithms on simulated error-free reads from sequenced genomes. For each algorithm, we sampled (N, L) points predicted to give $< 5\%$ error, and recorded the number of times correct reconstruction was achieved out of 100 trials. Fig. 4.9 shows results for the three GAGE reference sequences.

We now estimate the run-time of MultiBridging. The algorithm has two phases: the

K-mer graph formation step, and the repeat resolution step. The K-mer graph formation runtime can be easily bounded by $O((L-K)NK)$, assuming $O(K)$ look-up time for each of the $(L-K)N$ K-mers observed in reads. This step is common to all K-mer graph based algorithms, so previous works to decrease the practical runtime or memory requirements are applicable.

The repeat resolution step depends on the repeat statistics and choice of K. It can be loosely bounded as $O\Big(\sum_{\ell=K}^{L} L \sum_{\substack{\text{max repeats } x\\ \text{of length } \ell}} d_x\Big)$. The second sum is over distinct maximal repeats x of length ℓ and d_x is the number of (not necessarily maximal) copies of repeat x. The bound comes from the fact that each maximal repeat of length $K < \ell < L$ is resolved via exactly one bridged X-node, and each such resolution requires examining at most the Ld_x distinct reads that contain the repeat. We note that $\sum_{\ell=K}^{L} L \sum_{\substack{\text{max repeats } x\\ \text{of length } \ell}} d_x < L\sum_{\ell=K}^{L} a_\ell$, and the latter quantity is easily computable from our sufficient statistics.

For our data sets, with appropriate choice of K, the bridging step is much simpler than the K-mer graph formation step: for *R. sphaeroides* we use $K = 40$ to get $\sum_{\ell=K}^{L} La_\ell = 412$; in contrast, $N > 22421$ for the relevant range of L. Similarly, for hc14, using $K = 300$, $\sum_{\ell=K}^{L} La_\ell = 661$ while $N > 733550$; for *S. Aureus*, $\sum_{\ell=K}^{L} La_\ell = 558$ while $N > 8031$.

4.6 Conclusion

The notion of *optimal shotgun assembly* is not commonly discussed in the literature. One reason is that there is no universally agreed-upon metric of success. Another reason is that most of the optimization-based formulations of assembly have been shown to be NP-hard, including Shortest Common Superstring [Gallant et al., 1980], Kececioglu and Myers [1995], De Bruijn Superwalk [Pevzner et al., 2001], Medvedev et al. [2007], and Minimum s-Walk on the string graph [Myers, 2005], [Medvedev et al., 2007]. Thus, it would seem that optimal assembly algorithms are out of the question from a computational perspective. What we show in this paper is that if the goal is complete reconstruction, then one can define a clear notion of optimality, and moreover there is a computationally efficient assembly algorithm (MultiBridging) that is near optimal for a wide range of DNA repeat statistics. So while the reconstruction problem may well be NP-hard, typical instances of the problem seem much easier than the worst-case, a possibility already suggested by Nagarajan and Pop [2009].

The MultiBridging algorithm is near optimal in the sense that, for a wide range of repeat statistics, it requires the minimum read length and minimum coverage depth to achieve complete reconstruction. However, since the repeat statistics of a genome to be sequenced are usually not known in advance, this minimum required read length and minimum required coverage depth may also not be known in advance. In this context, it would be useful for the MultiBridging algorithm to *validate* whether its assembly is correct. More generally, an interesting question is to seek algorithms which are not only optimal in their data requirements but also provide a measure of confidence in their assemblies.

How realistic is the goal of complete reconstruction given current-day sequencing technologies? The minimum read lengths L_{crit} required for complete reconstruction on the datasets we examined are typically on the order of $500 - 3000$ base pairs (bp). This is substantially longer than the reads produced by Illumina, the current dominant sequencing technology, which produces reads of lengths 100-200bp; however, other technologies produce longer reads. PacBio reads can be as long as several thousand base pairs, and as demonstrated by Koren et al. [2012], the noise can be cleaned by Illumina reads to enable near-complete reconstruction. Thus our framework is already relevant to some of the current cutting edge technologies. To make our framework more relevant to short-read technologies such as Illumina, an important direction is to incorporate mate-pairs in the read model, which can help to resolve long repeats with short reads. Other extensions to the basic shotgun sequencing model:

Heterogenous read lengths: This occurs in some technologies where the read length is random (e.g. Pacbio) or when reads from multiple technologies are used. Generalized Ukkonen's conditions and the sufficient conditions of MultiBridging extend verbatim to this case, and only the computation of the bridging probability (4.3) has to be slightly modified.

Non-uniform read coverage: Again, only the computation of the bridging probability has to be modified. One issue of interest is to investigate whether reads are sampled less frequently from long repeat regions. If so, our framework can quantify the performance hit.

Double strand: DNA is double-stranded and consists of a length-G sequence $\mathbf{u}$ and its reverse complement $\tilde{\mathbf{u}}$. Each read is either sampled from $\mathbf{u}$ or $\tilde{\mathbf{u}}$. This more realistic scenario can be mapped into our single-strand model by defining $\mathbf{s}$ as the length-$2G$ concatenation of $\mathbf{u}$ and $\tilde{\mathbf{u}}$, transforming each read into itself and its reverse complement so that there are $2N$ reads. Generalized Ukkonen's conditions hold verbatim for this problem, and MultiBridging can be applied, with the slight modification that instead of looking for a single Eulerian path, it should look for two Eulerian paths, one for each component of the sequence graph after repeat-resolution. An interesting aspect of this model is that, in addition to interleaved repeats on the single strand $\mathbf{u}$, *reverse complement repeats* on $\mathbf{u}$ will also induce interleaved repeats on the sequence $\mathbf{s}$.

Chapter 5

Conclusion

There are several challenging properties of the assembly problem: handling the quantity of data, creating an accurate read-graph given low coverage and errors, and resolving repetitive regions. Our emphasis is on the resolution of repeats. In this dissertation, we have proposed two novel algorithms for assembly, and we have analyzed theoretically some necessary and sufficient conditions for perfect assembly in the presence of repeats.

Both of our assembly algorithms use paired reads to generate assemblies compatible with the data. Our approaches avoid the following requirements, which are typical in other assembly algorithms: (1) the requirement that there must be a unique appropriate-length path between the reads in a pair in order for the read-pair to be used for repeat resolution, and (2) the requirement that the reads must fall into non-repetitive contigs in order to be used for repeat resolution. As a result, our algorithm is able to use individual long-insert reads in places where other algorithms may not.

Our two algorithms take different approaches, each with their own set of advantages and disadvantages. Telescoper, our iterative algorithm, assembles somewhat aggressively, and so can make mistakes if there are multiple solutions and the choice among them is ambiguous according to its criteria. However, when the reads have high coverage and contain few errors, and when the long insert is longer than the longest repeats, it can often unambiguously assemble where other algorithms cannot. Piper, the maximum likelihood approach, should eventually have superior handling of rare read errors and low coverage, due to both its more principled likelihood calculation, and to the fact that it is designed to handle uncertainties in the data by explicitly listing the most likely alternative solutions along with ranking of their relative likelihood. Its primary downside is that it does not yet scale well. On large datasets with combinatorial explosion, Piper may fail to run to completion.

The requirements for either a Telescoper-like algorithm to succeed, or a Piper-like algorithm to succeed are fairly different from those of other assemblers, as evidenced by their different domains of success relative to other assembly algorithms. From this perspective, they are a positive contribution to the field, and it would be interesting to explore the sufficient conditions under which they assemble correctly.

The fact that Piper does well on simulated data of sufficiently small size, even in regions where other assembly algorithms do not assemble, suggests that at least in some regions of the genome, the problem is an algorithmic problem, rather than one arising from a lack of information.

We have also theoretically analyzed the fundamental limits of assembly in the presence of repeats in terms of the repeat statistics, number of reads, and length of reads for single end reads. We conclude that for realistic repeat statistics, for almost any values of read length and read number for which it is *fundamentally possible* to perform perfect assembly with high probability, such perfect assembly can be performed with high probability by a simple algorithm.

An incredibly appealing open line of work is to identify the fundamental limits of paired-end assembly, and to suggest a simple, computationally tractable algorithm that can assemble correctly for realistic genome sequences and their read-pair parameters. Going further, the final aim would be to go from theoretically-motivated algorithms to robust versions employable on real data, thereby bridging the gap between theoretical analysis and practical tool design.

Bibliography

C. A. Albers, G. Lunter, D. G. MacArthur, G. McVean, W. H. Ouwehand, and R. Durbin. Dindel: Accurate indel calls from short-read data. *Genome Research*, 21(6):961–973, Jun 2011. ISSN 1088-9051. doi: 10.1101/gr.112326.110. URL `http://dx.doi.org/10.1101/gr.112326.110`.

C. Alkan, S. Sajjadian, and E. E. Eichler. Limitations of next-generation genome sequence assembly. *Nature Methods*, 8(1):61–65, Jan 2011. ISSN 1548-7105. doi: 10.1038/nmeth.1527. URL `http://dx.doi.org/10.1038/nmeth.1527`.

P. N. Ariyaratne and W.-k. Sung. PE-Assembler: de novo assembler using short paired-end reads. *Bioinformatics*, 27(2):167–174, Jan 2011. ISSN 1460-2059. doi: 10.1093/bioinformatics/btq626. URL `http://dx.doi.org/10.1093/bioinformatics/btq626`.

Arthropod Genomic Consortium. i5k Insect and other Arthropod Genome Sequencing Initiative. `http://www.arthropodgenomes.org/wiki/i5K`, 2014. Last accessed April 21, 2014.

I. Astrovskaya, B. Tork, S. Mangul, K. Westbrooks, I. Mndoiu, P. Balfe, and A. Zelikovsky. Inferring viral quasispecies spectra from 454 pyrosequencing reads. *BMC Bioinformatics*, 12(Suppl 6):S1, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-s6-s1. URL `http://dx.doi.org/10.1186/1471-2105-12-S6-S1`.

J. A. Bailey and E. E. Eichler. Primate segmental duplications: crucibles of evolution, diversity and disease. *Nature Reviews Genetics*, 7(7):552–564, Jul 2006. ISSN 1471-0064. doi: 10.1038/nrg1895. URL `http://dx.doi.org/10.1038/nrg1895`.

J. A. Bailey, A. M. Yavor, H. F. Massa, B. J. Trask, and E. E. Eichler. Segmental Duplications: Organization and Impact Within the Current Human Genome Project Assembly. *Genome Research*, 11(6):1005–1017, 2001.

M. Baker. De novo genome assembly: what every biologist should know. *Nature Methods*, 9(4):333–337, Mar 2012. ISSN 1548-7105. doi: 10.1038/nmeth.1935. URL `http://dx.doi.org/10.1038/nmeth.1935`.

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, et al. SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *Journal of Computational Biology*, 19(5):455–477, May 2012. ISSN 1557-8666. doi: 10.1089/cmb.2012.0021. URL `http://dx.doi.org/10.1089/cmb.2012.0021`.

A. K. Björklund, S. Light, R. Sagit, and A. Elofsson. Nebulin: A Study of Protein Repeat Evolution. *Journal of Molecular Biology*, 402(1):38–51, Sep 2010. ISSN 0022-2836. doi: 10.1016/j.jmb.2010.07.011. URL `http://dx.doi.org/10.1016/j.jmb.2010.07.011`.

M. Boetzer, C. V. Henkel, H. J. Jansen, D. Butler, and W. Pirovano. Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, 27(4):578–579, Feb 2011. ISSN 1460-2059. doi: 10.1093/bioinformatics/btq683. URL `http://dx.doi.org/10.1093/bioinformatics/btq683`.

K. R. Bradnam, J. N. Fass, A. Alexandrov, P. Baranay, M. Bechner, I. Birol, S. Boisvert, J. A. Chapman, G. Chapuis, R. Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *Giga Sci*, 2(1):10, 2013. ISSN 2047-217X. doi: 10.1186/2047-217x-2-10. URL `http://dx.doi.org/10.1186/2047-217X-2-10`.

G. Bresler, M. Bresler, and D. Tse. Optimal assembly for high throughput shotgun sequencing. *BMC Bioinformatics*, 14(Suppl 5):S18, 2013.

M. Bresler, S. Sheehan, A. H. Chan, and Y. S. Song. Telescoper: de novo assembly of highly repetitive regions. *Bioinformatics*, 28(18):i311–i317, Sep 2012. ISSN 1460-2059. doi: 10.1093/bioinformatics/bts399. URL `http://dx.doi.org/10.1093/bioinformatics/bts399`.

M. Bresler, P. Jin, R. Pandya, and Y. S. Song. Piper: likelihood-based assembly (tentative title). In preparation, 2014.

M. J. Chaisson, D. Brinza, and P. A. Pevzner. De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Research*, 19(2):336–346, Dec 2008. ISSN 1088-9051. doi: 10.1101/gr.079053.108. URL `http://dx.doi.org/10.1101/gr.079053.108`.

E. Check Hayden. Is the $1,000 genome for real? *Nature*, Jan 2014. ISSN 1476-4687. doi: 10.1038/nature.2014.14530. URL `http://dx.doi.org/10.1038/nature.2014.14530`.

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, Feb 2013. ISSN 1460-2059. doi: 10.1093/bioinformatics/bts723. URL `http://dx.doi.org/10.1093/bioinformatics/bts723`.

P. Cliften. Finding Functional Features in Saccharomyces Genomes by Phylogenetic Footprinting. *Science*, 301(5629):71–76, Jul 2003. ISSN 1095-9203. doi: 10.1126/science.1084337. URL `http://dx.doi.org/10.1126/science.1084337`.

P. E. C. Compeau, P. A. Pevzner, and G. Tesler. How to apply de Bruijn graphs to genome assembly. *Nature Biotechnology*, 29(11):987–991, Nov 2011. ISSN 1546-1696. doi: 10.1038/nbt.2023. URL `http://dx.doi.org/10.1038/nbt.2023`.

Computational Research and Development Group, Broad Institute of MIT and Harvard. Discovar. `http://www.broadinstitute.org/software/discovar/blog`, 2014. Last accessed January 2014.

A. Dayarian, T. P. Michael, and A. M. Sengupta. SOPRA: Scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics*, 11(1):345, 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-345. URL `http://dx.doi.org/10.1186/1471-2105-11-345`.

A. L. Delcher. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research*, 30(11):2478–2483, Jun 2002. ISSN 1362-4962. doi: 10.1093/nar/30.11.2478. URL `http://dx.doi.org/10.1093/nar/30.11.2478`.

P. Diehl. BGI Plans to Sequence the World. `http://biotech.about.com/od/investinginbiotech/a/Bgi-Plans-To-Sequence-The-World.htm`, 2013. Last accessed September 2013.

R. Drmanac, A. B. Sparks, M. J. Callow, A. L. Halpern, N. L. Burns, B. G. Kermani, P. Carnevali, I. Nazarenko, G. B. Nilsen, G. Yeung, et al. Human Genome Sequencing Using Unchained Base Reads on Self-Assembling DNA Nanoarrays. *Science*, 327(5961):78–81, Jan 2010. ISSN 1095-9203. doi: 10.1126/science.1181498. URL `http://dx.doi.org/10.1126/science.1181498`.

D. Earl, K. Bradnam, J. St. John, A. Darling, D. Lin, J. Fass, H. O. K. Yu, V. Buffalo, D. R. Zerbino, M. Diekhans, et al. Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Genome Research*, 21(12):2224–2241, Dec 2011. ISSN 1088-9051. doi: 10.1101/gr.126599.111. URL `http://dx.doi.org/10.1101/gr.126599.111`.

J. Gallant, D. Maier, and J. Astorer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20(1):50–58, Feb 1980. ISSN 0022-0000. doi: 10.1016/0022-0000(80)90004-5. URL `http://dx.doi.org/10.1016/0022-0000(80)90004-5`.

S. Gao, W.-K. Sung, and N. Nagarajan. Opera: Reconstructing Optimal Genomic Scaffolds with High-Throughput Paired-End Sequences. *Journal of Computational Biology*, 18(11):1681–1691, Nov 2011. ISSN 1557-8666. doi: 10.1089/cmb.2011.0170. URL `http://dx.doi.org/10.1089/cmb.2011.0170`.

Genome 10K Community of Scientists. Genome 10K: A Proposal to Obtain Whole-Genome Sequence for 10,000 Vertebrate Species. *J Hered.*, 100(6):659–674, 2009.

M. Ghodsi, C. M. Hill, I. Astrovskaya, H. Lin, D. D. Sommer, S. Koren, and M. Pop. De novo likelihood-based measures for comparing genome assemblies. *BMC Res Notes*, 6(1): 334, 2013. ISSN 1756-0500. doi: 10.1186/1756-0500-6-334. URL `http://dx.doi.org/10.1186/1756-0500-6-334`.

S. Gnerre, I. MacCallum, D. Przybylski, F. J. Ribeiro, J. N. Burton, B. J. Walker, T. Sharpe, G. Hall, T. P. Shea, S. Sykes, et al. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, 108(4):1513–1518, Jan 2011. ISSN 1091-6490. doi: 10.1073/pnas.1017351108. URL `http://dx.doi.org/10.1073/pnas.1017351108`.

A. A. Gritsenko, J. F. Nijkamp, M. J. T. Reinders, and D. D. Ridder. GRASS: a generic algorithm for scaffolding next-generation sequencing assemblies. *Bioinformatics*, 28(11): 1429–1437, Jun 2012. ISSN 1460-2059. doi: 10.1093/bioinformatics/bts175. URL `http://dx.doi.org/10.1093/bioinformatics/bts175`.

T. D. Harris, P. R. Buzby, H. Babcock, E. Beer, J. Bowers, I. Braslavsky, M. Causey, J. Colonell, J. DiMeo, J. W. Efcavitch, et al. Single-Molecule DNA Sequencing of a Viral Genome. *Science*, 320(5872):106–109, Apr 2008. ISSN 1095-9203. doi: 10.1126/science.1150427. URL `http://dx.doi.org/10.1126/science.1150427`.

X. Huang. CAP3: A DNA Sequence Assembly Program. *Genome Research*, 9(9):868–877, Sep 1999. ISSN 1088-9051. doi: 10.1101/gr.9.9.868. URL `http://dx.doi.org/10.1101/gr.9.9.868`.

J. Huddleston, S. Ranade, M. Malig, F. Antonacci, M. Chaisson, L. Hon, P. H. Sudmant, T. A. Graves, C. Alkan, M. Y. Dennis, et al. Reconstructing complex regions of genomes using long-read sequencing technology. *Genome Research*, 24(4):688–696, Apr 2014. ISSN 1088-9051. doi: 10.1101/gr.168450.113. URL `http://dx.doi.org/10.1101/gr.168450.113`.

M. Hunt, C. Newbold, M. Berriman, and T. D. Otto. A comprehensive evaluation of assembly scaffolding tools. *Genome Biology*, 15(3):R42, 2014. ISSN 1465-6906. doi: 10.1186/gb-2014-15-3-r42. URL `http://dx.doi.org/10.1186/gb-2014-15-3-r42`.

R. M. Idury and M. S. Waterman. A New Algorithm for DNA Sequence Assembly. *Journal of Computational Biology*, 2(2):291–306, Jan 1995. ISSN 1557-8666. doi: 10.1089/cmb.1995.2.291. URL `http://dx.doi.org/10.1089/cmb.1995.2.291`.

Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2):226–232, Jan 2012. ISSN 1546-1718. doi: 10.1038/ng.1028. URL `http://dx.doi.org/10.1038/ng.1028`.

D. B. Jaffe, T. Sharpe, S. Yin, N. Weisenfeld, B. Lau, L. Williams, D. Tabbaa, A. Gnirke, C. Russ, C. Nusbaum, and I. MacCallum. Highly accurate determination of indels (and

SNPs) from human resequencing data with an assembly-based approach. Unpublished talk, May 2013.

JGI. JGI - Project List. `http://genome.jgi.doe.gov/genome-projects`, 2014. Last accessed April 21, 2014.

J. D. Kececioglu and E. W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1-2):7–51, Feb 1995. ISSN 1432-0541. doi: 10.1007/bf01188580. URL `http://dx.doi.org/10.1007/BF01188580`.

M. Kellis, N. Patterson, M. Endrizzi, B. Birren, and E. S. Lander. Sequencing and comparison of yeast species to identify genes and regulatory elements. *Nature*, 423(6937):241–254, May 2003. ISSN 0028-0836. doi: 10.1038/nature01644. URL `http://dx.doi.org/10.1038/nature01644`.

M. Kellis, B. W. Birren, and E. S. Lander. Proof and evolutionary analysis of ancient genome duplication in the yeast Saccharomyces cerevisiae. *Nature*, 428(6983):617–624, Apr 2004. ISSN 1476-4679. doi: 10.1038/nature02424. URL `http://dx.doi.org/10.1038/nature02424`.

S. Koren, M. C. Schatz, B. P. Walenz, J. Martin, J. T. Howard, G. Ganapathy, Z. Wang, D. A. Rasko, W. R. McCombie, E. D. Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature Biotechnology*, 30(7):693–700, Jul 2012. ISSN 1546-1696. doi: 10.1038/nbt.2280. URL `http://dx.doi.org/10.1038/nbt.2280`.

E. S. Lander and M. S. Waterman. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics*, 2(3):231–239, Apr 1988. ISSN 0888-7543. doi: 10.1016/0888-7543(88)90007-9. URL `http://dx.doi.org/10.1016/0888-7543(88)90007-9`.

J. Laserson, V. Jojic, and D. Koller. Genovo: De Novo Assembly for Metagenomes . *Journal of Computational Biology*, 18(3):429–443, Mar 2011. ISSN 1557-8666. doi: 10.1089/cmb.2010.0244. URL `http://dx.doi.org/10.1089/cmb.2010.0244`.

R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265–272, Feb 2010. ISSN 1088-9051. doi: 10.1101/gr.097261.109. URL `http://dx.doi.org/10.1101/gr.097261.109`.

R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, G. He, Y. Chen, Q. Pan, Y. Liu, et al. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *Giga Sci*, 1(1):18, 2012. ISSN 2047-217X. doi: 10.1186/2047-217x-1-18. URL `http://dx.doi.org/10.1186/2047-217X-1-18`.

I. MacCallum, D. Przybylski, S. Gnerre, J. Burton, I. Shlyakhter, A. Gnirke, J. Malek, K. McKernan, S. Ranade, T. P. Shea, et al. ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome Biology*, 10(10): R103, 2009. ISSN 1465-6906. doi: 10.1186/gb-2009-10-10-r103. URL `http://dx.doi.org/10.1186/gb-2009-10-10-r103`.

M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y.-J. Chen, Z. Chen, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, Jul 2005. ISSN 1476-4679. doi: 10.1038/nature03959. URL `http://dx.doi.org/10.1038/nature03959`.

M. J. McEachern, A. Krauskopf, and E. H. Blackburn. Telomeres and their control. *Annu. Rev. Genet.*, 34(1):331–358, Dec 2000. ISSN 1545-2948. doi: 10.1146/annurev.genet.34.1.331. URL `http://dx.doi.org/10.1146/annurev.genet.34.1.331`.

K. J. McKernan, H. E. Peckham, G. L. Costa, S. F. McLaughlin, Y. Fu, E. F. Tsung, C. R. Clouser, C. Duncan, J. K. Ichikawa, C. C. Lee, et al. Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding. *Genome Research*, 19(9):1527–1541, Sep 2009. ISSN 1088-9051. doi: 10.1101/gr.091868.109. URL `http://dx.doi.org/10.1101/gr.091868.109`.

P. Medvedev and M. Brudno. Maximum Likelihood Genome Assembly. *Journal of Computational Biology*, 16(8):1101–1116, Aug 2009. ISSN 1557-8666. doi: 10.1089/cmb.2009.0047. URL `http://dx.doi.org/10.1089/cmb.2009.0047`.

P. Medvedev, K. Georgiou, G. Myers, and M. Brudno. Computability of Models for Sequence Assembly. *Lecture Notes in Computer Science*, pages 289–301, 2007. ISSN 1611-3349. doi: 10.1007/978-3-540-74126-8_27. URL `http://dx.doi.org/10.1007/978-3-540-74126-8_27`.

P. Medvedev, S. Pham, M. Chaisson, G. Tesler, and P. Pevzner. Paired de Bruijn Graphs: A Novel Approach for Incorporating Mate Pair Information into Genome Assemblers. *Journal of Computational Biology*, 18(11):1625–1634, Nov 2011. ISSN 1557-8666. doi: 10.1089/cmb.2011.0151. URL `http://dx.doi.org/10.1089/cmb.2011.0151`.

S. Motahari, G. Bresler, and D. Tse. Information theory of DNA sequencing. 2012. URL `arXiv:1203.6233`.

E. W. Myers. Toward Simplifying and Accurately Formulating Fragment Assembly. *Journal of Computational Biology*, 2(2):275–290, Jan 1995. ISSN 1557-8666. doi: 10.1089/cmb.1995.2.275. URL `http://dx.doi.org/10.1089/cmb.1995.2.275`.

E. W. Myers. A Whole-Genome Assembly of Drosophila. *Science*, 287(5461):2196–2204, Mar 2000. ISSN 1095-9203. doi: 10.1126/science.287.5461.2196. URL `http://dx.doi.org/10.1126/science.287.5461.2196`.

E. W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(Suppl 2):ii79–ii85, Sep 2005. ISSN 1460-2059. doi: 10.1093/bioinformatics/bti1114. URL `http://dx.doi.org/10.1093/bioinformatics/bti1114`.

N. Nagarajan and M. Pop. Parametric Complexity of Sequence Assembly: Theory and Applications to Next Generation Sequencing. *Journal of Computational Biology*, 16(7): 897–908, Jul 2009. ISSN 1557-8666. doi: 10.1089/cmb.2009.0005. URL `http://dx.doi.org/10.1089/cmb.2009.0005`.

National Human Genome Research Institute, NIH. Human Genome Sequence Quality Standards. `http://www.genome.gov/10000923`, 2012. Last accessed December 12, 2012.

N. Parrish, F. Hormozdiari, and E. Eskin. Assembly of non-unique insertion content using next-generation sequencing. *BMC Bioinformatics*, 12(Suppl 6):S3, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-s6-s3. URL `http://dx.doi.org/10.1186/1471-2105-12-S6-S3`.

Y. Peng, H. C. M. Leung, S. M. Yiu, and F. Y. L. Chin. IDBA - A Practical Iterative de Bruijn Graph De Novo Assembler. *Lecture Notes in Computer Science*, pages 426–440, 2010. ISSN 1611-3349. doi: 10.1007/978-3-642-12683-3_28. URL `http://dx.doi.org/10.1007/978-3-642-12683-3_28`.

Z. Peng, Z. Zhao, N. Nath, J. L. Froula, A. Clum, T. Zhang, J.-f. Cheng, A. C. Copeland, L. A. Pennacchio, and F. Chen. Generation of Long Insert Pairs Using a Cre-LoxP Inverse PCR Approach. *PLoS ONE*, 7(1):e29437, Jan 2012. ISSN 1932-6203. doi: 10.1371/journal.pone.0029437. URL `http://dx.doi.org/10.1371/journal.pone.0029437`.

P. A. Pevzner. 1-Tuple DNA sequencing: computer analysis. *J Biomol Struct Dyn.*, 7(1): 63–67, 1989.

P. A. Pevzner. DNA physical mapping and alternating Eulerian cycles in colored graphs. *Algorithmica*, 13(1-2):77–105, Feb 1995. ISSN 1432-0541. doi: 10.1007/bf01188582. URL `http://dx.doi.org/10.1007/BF01188582`.

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. ISSN 1091-6490. doi: 10.1073/pnas.171285098. URL `http://dx.doi.org/10.1073/pnas.171285098`.

M. Pop. Hierarchical Scaffolding With Bambus. *Genome Research*, 14(1):149–159, Dec 2003. ISSN 1088-9051. doi: 10.1101/gr.1536204. URL `http://dx.doi.org/10.1101/gr.1536204`.

A. Rahman and L. Pachter. CGAL: computing genome assembly likelihoods. *Genome Biology*, 14(1):R8, 2013. ISSN 1465-6906. doi: 10.1186/gb-2013-14-1-r8. URL `http://dx.doi.org/10.1186/gb-2013-14-1-r8`.

J. M. Rothberg, W. Hinz, T. M. Rearick, J. Schultz, W. Mileski, M. Davey, J. H. Leamon, K. Johnson, M. J. Milgrew, M. Edwards, et al. An integrated semiconductor device enabling non-optical genome sequencing. *Nature*, 475(7356):348–352, Jul 2011. ISSN 1476-4687. doi: 10.1038/nature10242. URL `http://dx.doi.org/10.1038/nature10242`.

L. Salmela, V. Makinen, N. Valimaki, J. Ylinen, and E. Ukkonen. Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, 27(23):3259–3265, Dec 2011. ISSN 1460-2059. doi: 10.1093/bioinformatics/btr562. URL `http://dx.doi.org/10.1093/bioinformatics/btr562`.

S. L. Salzberg and J. A. Yorke. Beware of mis-assembled genomes. *Bioinformatics*, 21 (24):4320–4321, Dec 2005. ISSN 1460-2059. doi: 10.1093/bioinformatics/bti769. URL `http://dx.doi.org/10.1093/bioinformatics/bti769`.

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, et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome Research*, 22(3):557–567, Mar 2012. ISSN 1088-9051. doi: 10.1101/gr.131383.111. URL `http://dx.doi.org/10.1101/gr.131383.111`.

J. T. Simpson and R. Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Research*, 22(3):549–556, Mar 2012. ISSN 1088-9051. doi: 10.1101/gr.126953.111. URL `http://dx.doi.org/10.1101/gr.126953.111`.

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, Jun 2009. ISSN 1088-9051. doi: 10.1101/gr.089532.108. URL `http://dx.doi.org/10.1101/gr.089532.108`.

G. G. Sutton, O. White, M. D. Adams, and A. R. Kerlavage. TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects. *Genome Science and Technology*, 1 (1):9–19, Jan 1995. ISSN 1070-2830. doi: 10.1089/gst.1995.1.9. URL `http://dx.doi.org/10.1089/gst.1995.1.9`.

T. J. Treangen and S. L. Salzberg. Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nature Reviews Genetics*, Nov 2011. ISSN 1471-0064. doi: 10.1038/nrg3117. URL `http://dx.doi.org/10.1038/nrg3117`.

E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, Jan 1992. ISSN 0304-3975. doi: 10.1016/0304-3975(92)90143-4. URL `http://dx.doi.org/10.1016/0304-3975(92)90143-4`.

F. Van Nieuwerburgh, R. C. Thompson, J. Ledesma, D. Deforce, T. Gaasterland, P. Ordoukhanian, and S. R. Head. Illumina mate-paired DNA sequencing-library preparation using Cre-Lox recombination. *Nucleic Acids Research*, 40(3):e24–e24, Feb 2012. ISSN 1362-4962. doi: 10.1093/nar/gkr1000. URL `http://dx.doi.org/10.1093/nar/gkr1000`.

A. Voskoboynik, N. F. Neff, D. Sahoo, A. M. Newman, D. Pushkarev, W. Koh, B. Passarelli, H. C. Fan, G. L. Mantalas, K. J. Palmeri, et al. The genome sequence of the colonial chordate, Botryllus schlosseri. *eLife*, 2(0):e00569–e00569, Jan 2013. ISSN 2050-084X. doi: 10.7554/elife.00569. URL `http://dx.doi.org/10.7554/eLife.00569`.

R. L. Warren, G. G. Sutton, S. J. M. Jones, and R. A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4):500–501, Feb 2007. ISSN 1460-2059. doi: 10.1093/bioinformatics/btl629. URL `http://dx.doi.org/10.1093/bioinformatics/btl629`.

O. Zagordi, A. Bhattacharya, N. Eriksson, and N. Beerenwinkel. ShoRAH: estimating the genetic diversity of a mixed sample from next-generation sequencing data. *BMC Bioinformatics*, 12(1):119, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-119. URL `http://dx.doi.org/10.1186/1471-2105-12-119`.

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. ISSN 1088-9051. doi: 10.1101/gr.074492.107. URL `http://dx.doi.org/10.1101/gr.074492.107`.

Appendix A

Appendix: Assembly Using Iterative Extension

A.1 Data simulation

In our simulation study, we generated three distinct types of simulated reads: 50 bp short-reads, 101 bp short-reads, and Sanger-reads. Each type was generated with different error profiles.

50 bp data: For all the reads of length 50 bp, we simulated using a uniform coverage distribution. Errors and corresponding quality scores were generated by mimicking standard short-read data of ϕX174 virus, sequenced on the Illumina GA-II platform. Specifically, for each simulated read, a randomly sampled ϕX174 read provided error positions and quality scores. The average per-base error rate was 1.2%.

101 bp data: For all the reads of length 101 bp, we used the simulator simNGS (www.ebi.ac.uk/goldman-srv/simNGS), which uses Illumina templates for reads of this length.

Sanger-reads: The Sanger-reads were simulated using a read length distribution equal to that of the Sanger-reads from Kellis et al. [2003] and Cliften [2003]. This amounted to coverage depth of 6.5X, with mean read length 593 bp. Insert size was modeled as normally distributed with mean 5 kb and standard deviation 1 kb. Errors were introduced uniformly at a rate of 0.1%.

A.2 Additional data sets for benchmarking

In addition to the data sets D1–D3 described in the main text, we were interested in a closer look at how assembly performance is affected by the insert distribution and the type of read data. We simulated four additional data sets with shorter insert lengths for the long-insert reads, and with Sanger-reads. These data sets are as follows:

Simulated Data D4: was intended to mimic our real data set. It consisted of two libraries, one short-insert and one long-insert, each with reads of length 50 bp. The *short-insert short-reads* were the same as those in D2. The *long-insert short-reads* had coverage depth 20X, with an insert size normally distributed with mean 2.2 kb and variance 800 bp.

Simulated Data D5: is similar to data set D4 but with longer read lengths. It consisted of the *short-insert short-reads* in data D1 (101 bp) combined with a *long-insert short-read* library with reads of length 101 bp, coverage depth 40X, and an insert size with mean 2.2 kb and variance 500 kb.

Simulated Data D6: consisted of the *short-insert short-reads* in D2 and *Sanger-reads*.

Simulated Data D7: consisted of the *short-insert short-reads* in D1 and the *Sanger-reads* from D6.

A.3 Running the assembly algorithms

We compared Telescoper to six other assembly algorithms, ABySS [Simpson et al., 2009], ALLPATHS 2 [MacCallum et al., 2009], SGA [Simpson and Durbin, 2012], SOAPdenovo [Li et al., 2010], Velvet [Zerbino and Birney, 2008], and where appropriate, the Sanger-read assembler Celera [Myers, 2000]. Below we describe the general approach we used to run each algorithm, using the data set D7 as an example. We tried to provide the other algorithms with as much useful information as possible to optimize their results. Given our computational resources, we tried to optimize over parameter ranges for the more important parameters and the faster assemblers. We largely used SOAPdenovo to guide our parameter choices for other algorithms, since it was very fast to run. Whenever possible we selected the output file with the most continuous contigs (i.e., if an algorithm produced both a contigs file and a scaffolds file, we chose the scaffolds file). We were unable to run every algorithm on every data set, but included as many as we could to improve the larger evaluation picture.

ABySS:

```
abyss-pe k=64 n=10 name=Scer_sanger lib='short long'
    short='Scer_short_A.fasta Scer_short_B.fasta'
    long='Scer_sanger_A.fasta Scer_sanger_B.fasta' np=16
```

The main parameter for ABySS is k, which we were unable to set higher than 64, so we chose this value for D1, D5, and D7 (101 bp reads). We chose $k = 31$ for D2 and D6, and $k = 21$ for D3 (real data) based on the optimal parameters for SOAPdenovo. We used a lower k for real data due to the more complex error distribution. ABySS does not take in parameters for the insert distribution, but rather infers them (usually quite successfully) on its own. We chose `n=10` (recommended value in the manual) as the number of paired-reads needed to join two contigs, and `np=16` as the number of processors to use.

ALLPATHS:

```
ulimit -s 100000
RunAllPaths PRE=. RUN=rundir REFERENCE_NAME=Scer_sanger/ K=24
    DATA_SUBDIR=output SOURCE_DIR=./Scer_sanger/data/ GC_BIAS_CORRECTION=False
    MAXPAR=16 PARALLEL_BATCHES=16
```

The `ulimit` was set to give ALLPATHS 2 more stack space. The main parameter here is K, the k-mer size for building a de Brujin graph, which could not be set higher than 24 (which is what we selected) for D1, D5, and D7. For D2, D3, and D6 we used $K = 20$ since the short-reads were shorter. ALLPATHS 2 also takes an "expected_lib_stats" file, where we described the location of the data, the paired-end nature, and provided the expected separation and standard deviation of insert distribution. For simulated data, the `GC_BIAS_CORRECTION` module did not run correctly, but we did include this step for D3. ALLPATHS 2 requires quality scores, which we generated for the simulated data sets, and provided for D3. We allotted 16 cores to ALLPATHS 2. For the real data set D3, ALLPATHS 2 ran out of memory, so we thinned reads to half the coverage.

SGA:

```
sh runSGA.sh Scer_short Scer_sanger 75
```

For SGA we used a shell script to run the main modules. The first parameter is the short-insert library, the second is the long-insert library, and the third is the k-mer size. Due to the relatively long runtime of SGA, we were unable to optimize over a large range of k, so we used the parameter values given in SGA's example script sga-celegans.sh for 100 bp data: $k = 75$, and error correction parameter 41. This k is very similar to that of SOAPdenovo for D1, D5, and D7. SGA is designed for read of length 100 bp or greater and did not run our 50 bp data sets, so we did not include it for D2, D3, D4, or D6.

SOAPdenovo:

```
SOAPdenovo-127mer all -s data/Scer_sanger.config -o output/Scer_sanger -K 77
```

SOAPdenovo has three executables for $k \leq 127$, $k \leq 63$, and $k \leq 31$, which we used as appropriate. Since SOAPdenovo runs very quickly, we tried $k = 49, \cdots, 85$ (odd) to select the best k for the 101 bp data sets, as shown for D1 in Figure A.1 and Table A.1. We chose the k that produced the highest % Aligned + % Covered, although as shown, k's close to the optimal value produced very similar results. We also optimized SOAPdenovo for the 50 bp data using $k = 21, \cdots, 31$ (odd). From these optimization we chose $k = 77$ for D1, $k = 79$ for D5 and D7, $k = 31$ for D2, D4, and D6, and $k = 21$ for D3. These choices helped select k for other algorithms. While ideally every algorithm would be optimized over every parameter, in practice close values of k did not substantially affect the results. In the config file we specified the location of the data and the expected separation size, as well as the maximum read length for all reads. We also had to specify an `asm_flag` for each library which encodes the stages where the reads should be used. For most libraries we set this flag to 3, except for sanger libraries where we set it to 2. We specified that the short-insert libraries should be used first for scaffolding, for D1-D6, since this generally gave better results, and that the Sanger reads should be used first for D6-D7, without which the scaffolding step frequently failed.

Velvet:

```
velveth Scer_sanger/ 31 -shortPaired -fastq Scer_short.A.fastq Scer_short.B.fastq
    -longPaired -fasta Scer_sanger.fasta
velvetg Scer_sanger/ -ins_length 400 -cov_cutoff 33 -min_contig_lgth 200
    -ins_length_long 5000 -ins_length_sd 75 -ins_length_long_sd 1000
```

We were unable to compile Velvet for higher k than 31, so for all data sets besides D3 we used $k = 31$. For D3 we selected $k = 21$, following SOAPdenovo. We provided information about the insert distributions of the different libraries, and specified a `cov_cutoff` of 1/3 the expected coverage, for removing nodes of low coverage. A high coverage cutoff made the results worse, but as long as it was reasonably low the results were virtually identical.

Celera:

```
runCA -d Scer_sanger/ -p Scer_sanger Scer_sanger.frg
```

For Celera there are few parameters to select. The `frg` file encodes the data, so we wrote scripts to convert fasta and quality files into the `frg` format, providing insert distribution statistics.

Telescoper:

For each seed string $j = 1, \dots, 32$ (16 chromosomes, 2 telomeres each):

```
Telescoper -shortInsReads Scer_short.fasta -longInsReads Scer_sanger.fasta
    -mean 400 -std 75 -meanLong 5000 -stdLong 1000 -seedSource seed(j).fasta
    -outdir Scer_sanger/ -outname (j)
    -ErrCorrnK 20 -ErrFreeKneiHigh 31 -ErrCorrminoverlap 20
```

For each data set the appropriate mean and variance of the insert distributions were used. The rest of the parameters varied only by read length. Specifically, for 50 bp reads, the parameters used were: `ErrCorrnK = 20` for the k-mer used to detect overlapping reads for error correction, `ErrCorrminoverlap = 20` for the minimum overlap (possibly with errors) at which reads were considered neighbors, and

`ErrFreeK_neiHigh = 31` for the initial threshold for overlapping reads in the read-graph. For 101 bp reads these values were `ErrCorrnK = 21`, `ErrCorrminoverlap = 21`, and `ErrFreeKneiHigh = 51`. Other parameters are optional, and were not specified, so that default values were used.

A.4 Optimizing over k

As described above, we tried to choose parameters to optimize performance for the other algorithms. Table A.1 and Figure A.1 summarize the performance of SOAPdenovo for different values of the parameter k between 61 and 89. We chose the row in Table A.1 with the best value for % Aligned + % Covered, although we note that results do not vary as much as they do between algorithms.

	Produced (kb)				Aligned (kb)					
Assembler	N50	NG50	Max	Total	N50	NG50	Max	Total	%Aligned	%Covered
SOAP, K=61	25.0	23.0	39.0	1127	28.7	23.2	40.0	1051	93.2	81.0
SOAP, K=63	26.1	25.0	39.0	1129	27.7	23.2	40.0	1053	93.3	81.2
SOAP, K=65	28.8	25.0	39.0	1128	30.0	25.3	40.0	1054	93.4	81.2
SOAP, K=67	30.0	25.0	39.0	1130	30.1	25.7	40.0	1055	93.4	81.4
SOAP, K=69	28.4	25.0	39.0	1133	30.5	25.7	40.0	1057	93.3	81.5
SOAP, K=71	26.0	25.0	39.0	1130	26.6	25.4	40.0	1060	93.7	81.7
SOAP, K=73	28.0	25.0	39.0	1143	30.5	25.4	40.0	1063	92.9	82.0
SOAP, K=75	26.2	25.0	39.0	1150	26.9	25.4	40.0	1064	92.5	82.1
SOAP, **K=77**	25.0	24.0	39.0	1149	28.6	**24.6**	40.0	1068	**92.9**	**82.3**
SOAP, K=79	25.0	22.0	39.0	1155	25.6	22.8	40.0	1070	92.6	82.4
SOAP, K=81	26.2	25.0	39.0	1159	28.6	25.4	40.0	1069	92.3	82.3
SOAP, K=83	28.3	26.0	39.0	1165	30.5	26.2	40.0	1071	91.9	82.6
SOAP, K=85	28.5	26.0	39.0	1169	31.0	26.3	40.0	1072	91.6	82.7
SOAP, K=87	28.0	28.0	39.0	1170	30.6	26.6	40.0	1067	91.1	82.4
SOAP, K=89	26.0	23.0	39.0	1169	28.6	23.8	40.0	1067	91.2	82.4

Table A.1: Results for SOAPdenovo on D1 (101 bp reads), using a variety of different values of the k-mer size k.

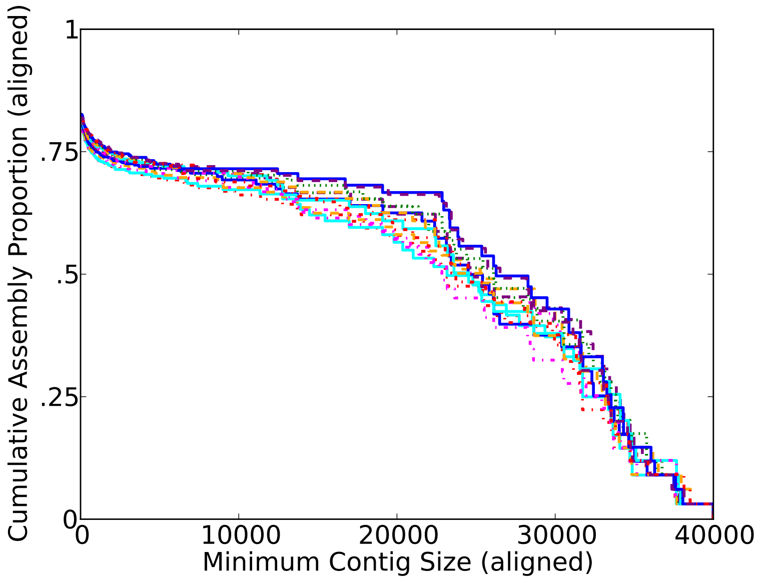


Figure A.1: The cumulative proportion of all aligned contigs exceeding the contig size indicated on the x-axis, for simulated data D1 (read length 101 bp). Results are for SOAPdenovo on a variety of k.

A.5 Telescoper implementation details

Read mapping: We employ a hash-based read mapping for computational efficiency. To accommodate sequencing errors, we allow up to δ mismatches in mapping left-mates to the *e-node*. Because we are interested in repetitive regions, the fact that a read-pair maps to one location need not preclude it mapping to another. Each time we map to an *e-node*, we use all reads, not just all unused reads.

Error correction: We perform error correction on the set of left-reads that map to a given *e-node* plus their associated right-reads. For each read R in the set, we find its neighbor reads (also in the set) meeting certain overlap criteria. Then, we perform error correction at each position of read R using the information of its neighbors' bases that overlap that position: If a given base call is confirmed by enough neighboring reads, or if no alternative base call has strong support, then no error correction is made; otherwise, the base is corrected to the majority vote. These conditions are similar to those used in ALLPATHS-LG's error correction algorithm [Gnerre et al., 2011].

Read-overlap graph construction: The coverage and error profiles are such that very rarely consecutive reads in the read-overlap graph overlap only by very small amount, such as 5 bp. Therefore, in introducing edges in the read-overlap graph, Telescoper proceeds by two passes: The first pass adds directed edges between reads overlapping by at least k bp. The second pass adds directed edges between the set of reads lacking outgoing edges and the set of reads lacking incoming edges; edges are simply added one at a time, starting with the most overlapping read pair, until either set of edge-less reads is empty. This step can be thought of as employing a partial greedy algorithm for overlapping reads and is motivated by the work of Motahari et al. [2012] showing that for non-repetitive sequence, the greedy algorithm is asymptotically optimal in terms of its coverage requirements. The sequence here is repetitive, of course, but with some probability the repetitive parts will have read-overlaps large enough to be added by the first pass.

A.6 Runtimes and memory requirements

We ran each algorithm on a Linux machine with 128 GB of memory, and 24 cores, although the algorithms' number of processors was set at 16 for parallelizable algorithms. The results are shown in Table A.2.

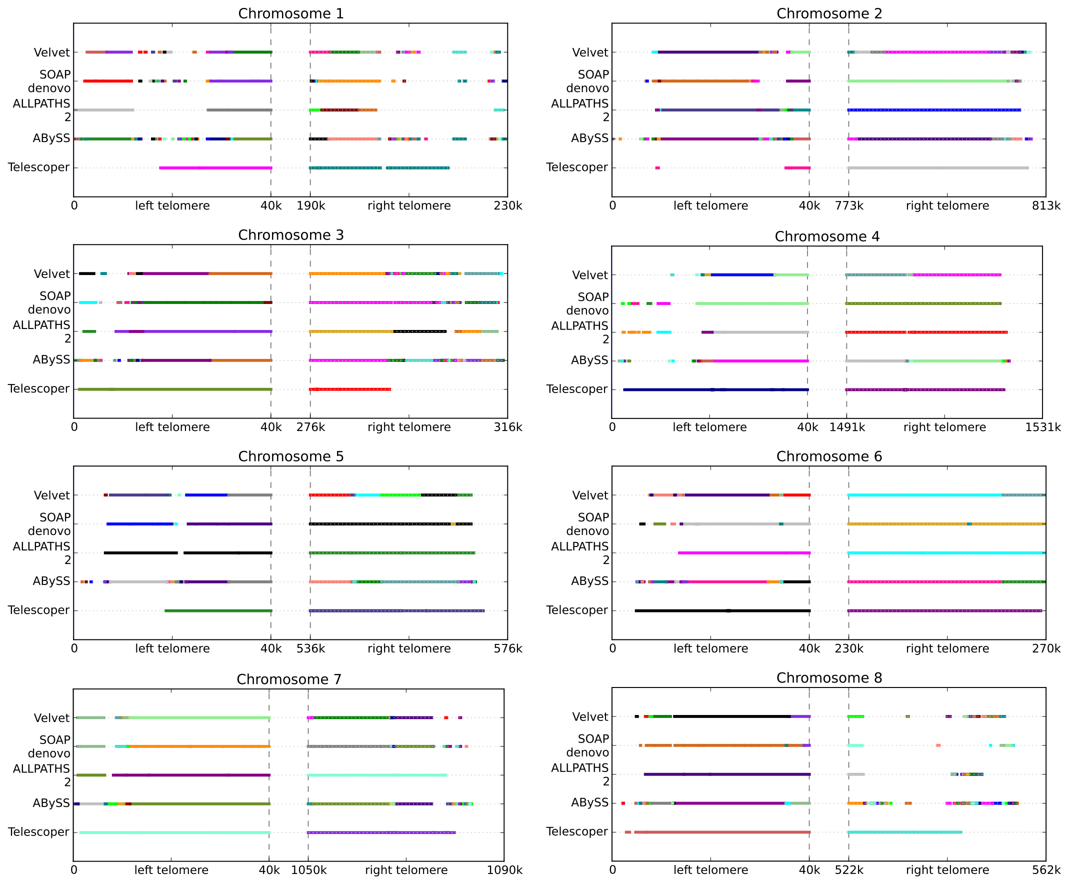
Telescoper's memory requirements are dominated by storing the reads and the hash table that indexes the reads by their first k-mer. This will scale asymptotically as $O(C \cdot G)$, where C is the coverage and G is the genome size. For the short-reads in data set D3, with coverage 100X and genome size $G = 12$ Mb, this consisted of 1.4 GB for the short reads, and 386 MB for the accompanying hash table. Future work to reduce the memory requirements may borrow ideas from memory-efficient algorithms such as SGA.

Assembler	D1: 10kb insert	D5: 2.2kb insert	D7: Sanger
ABySS 1.3.3	11m	9m	8m
ALLPATHS 2.2	87m	57m	40m
Celera 6.1	N/A	N/A	1m29s
SGA 6/3/2012	59m	31m	26m
SOAPdenovo 1.05	1m14s	1m35s	1m40s
Telescoper 0.9	35m	36m	36m
Velvet 1.2.06	3m18s	3m53s	2m36s

Table A.2: Runtime results for all algorithms on D1, D5, and D7, including version number.

A.7 Contig continuity plots for real yeast data D3

Below we include all the chromosome plots for D3 (real yeast data), where all the produced contigs have been aligned to the reference.



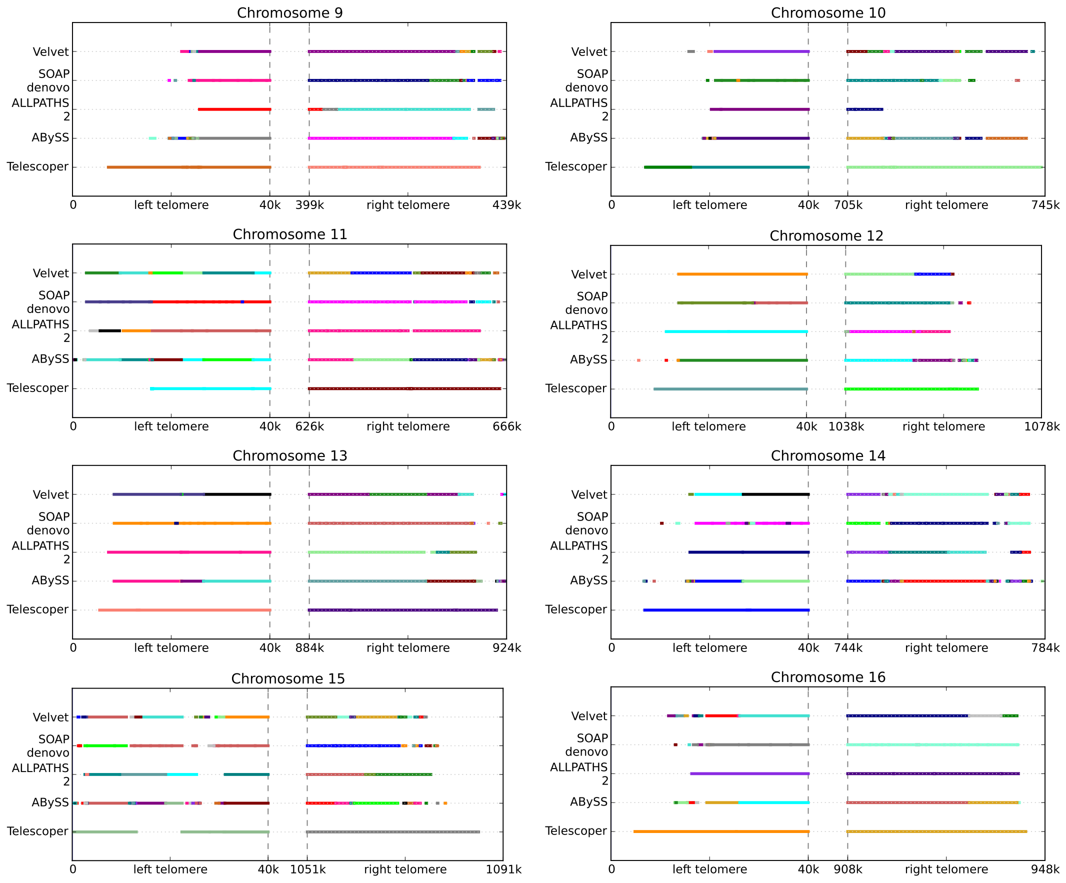
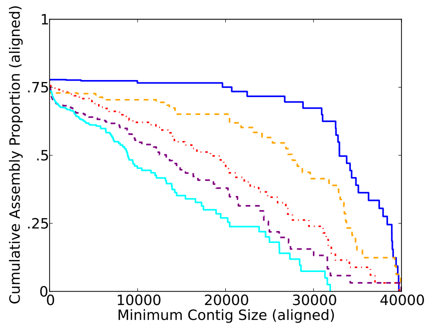
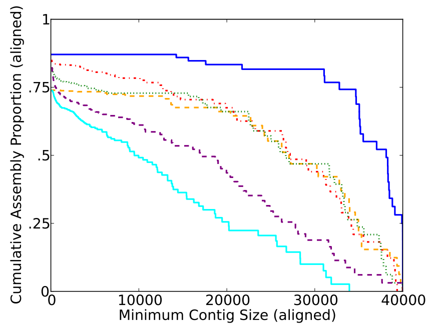


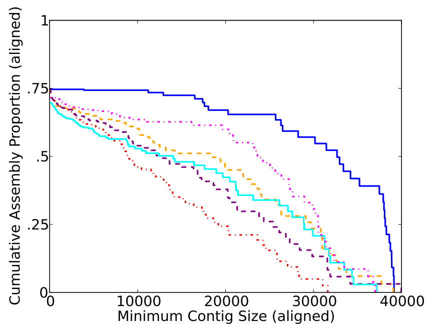
Figure A.2: Contig continuity results for real data D3. The left and right telomeric regions (separated by the dotted line) for two different chromosomes are shown, with the aligned contigs displayed for each assembly algorithm. Different colors represent different contigs in the produced assembly, so more colors per method indicates a larger number of contigs.



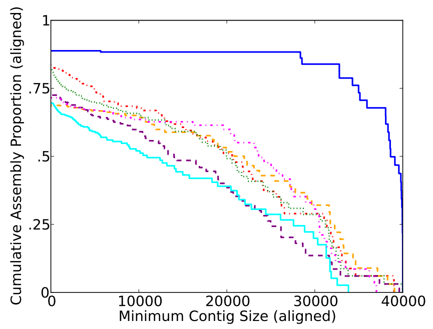
(a) D4: 50 bp reads, long-insert 2.2 kb



(b) D5: 101 bp reads, long-insert 2.2 kb



(c) D6: 50 bp short-reads with Sanger-reads



(d) D7: 101 bp short-reads with Sanger-reads

Figure A.3: The fraction of reference covered, for contigs exceeding the contig size indicated on the x-axis. These plots illustrate the continuity and completeness of different assemblies. For any given minimum contig length, Telescoper covered more of the reference. NG50 can be read from this graph as the x-coordinates at which each curve hits the 50% mark of bases output relative to the reference.

A.8 Results on additional data sets D4–D7

Results on additional data D4–D7 are shown in Figure A.3 and Table A.3. These results highlight different features.

Results on D4 show that Telescoper's weaker performance on the real data D3 as compared to its performance on simulated data sets D2 and D5 can largely be explained by the data parameters (shorter reads than D5, shorter long-insert distribution than D2), rather than being exclusively the result of the non-uniformities of real data.

Sanger reads (data D6 and D7) used in place of short-read long-insert libraries have a surprisingly modest effect on performance; the tradeoff between coverage and continuity of reads is fairly balanced. This is somewhat less true for other leading algorithms, which are not designed to run on Sanger reads. None of the algorithms other than Telescoper had higher NG50 than the Sanger-read assembler Celera.

Telescoper's %Aligned is somewhat low for some of the data sets. This occurs when Telescoper is incorrect in determining which extension to pursue after a repetitive region. Figure A.2 shows that after irresolvable repeats Telescoper frequently produces no aligned output, whereas other assemblers do. For instance, on the left telomere of chromosome 2, no algorithm is able to scaffold the interior contig to the rest, but the other algorithms produce the sequence further towards the end of the telomere as a separate contig. If Telescoper pursues a false extension path following the irresolvable choice, then not only is its %Covered heavily penalized, but so is its %Aligned. Better identification of ambiguity in extension paths would allow Telescoper to produce multiple contigs instead of single contigs, and thereby improve its %Aligned and %Covered, with only a slight potential decrease in NG50. The problem is exacerbated for the Sanger read data sets, for which the long-insert coverage is lower, so that one extension path might appear better than another by chance.

	Produced (kb)				Aligned (kb)					
Assembler	N50	NG50	Max	Total	N50	NG50	Max	Total	%Aligned	%Covered
Telescoper	36.0	35.0	39.0	1032	34.8	32.9	39.7	1003	97.2	77.8
ABySS	21.1	19.0	39.0	1136	22.8	19.4	39.7	1024	90.1	77.3
ALLPATHS2	33.0	28.0	52.0	993	32.9	27.6	40.0	935	94.0	72.9
SOAPdenovo	18.9	13.0	39.0	994	20.3	13.3	40.0	966	97.2	74.9
Velvet	13.0	8.0	31.0	968	14.2	8.9	31.9	946	97.6	73.6

(a) Results for simulated data D4: 50 bp reads, long-insert 2.2 kb

	Produced (kb)				Aligned (kb)					
Assembler	N50	NG50	Max	Total	N50	NG50	Max	Total	%Aligned	%Covered
Telescoper	39.1	38.0	40.0	1164	38.3	38.1	40.0	1114	95.7	87.1
ABySS	29.0	29.0	39.0	1240	29.3	29.0	39.4	1200	96.7	85.3
ALLPATHS2	33.4	30.0	50.0	987	33.4	27.3	40.0	947	96.0	73.9
SGA	32.1	26.0	39.0	1111	32.2	26.7	40.0	1076	96.7	82.1
SOAPdenovo	19.3	17.0	39.0	1175	20.5	17.2	40.0	1090	92.7	82.9
Velvet	13.9	9.0	33.0	964	14.4	9.5	33.9	948	98.3	73.8

(b) Results for simulated data D5: 101 bp reads, long-insert 2.2kb

	Produced (kb)				Aligned (kb)					
Assembler	N50	NG50	Max	Total	N50	NG50	Max	Total	%Aligned	%Covered
Telescoper	38.0	38.0	39.0	1021	37.5	32.7	39.2	955	93.5	74.6
ABySS	12.0	8.0	31.0	1096	13.7	8.9	31.6	965	88.1	75.0
ALLPATHS2	24.0	18.0	39.0	894	24.1	18.5	39.1	888	99.2	68.5
SOAPdenovo	18.6	13.0	39.0	1009	18.9	13.3	40.0	979	97.0	74.9
Velvet	26.0	20.0	75.0	1009	23.1	14.0	37.2	895	88.6	69.6
Celera	27.5	24.0	41.0	930	27.6	23.8	37.0	917	98.5	71.5

(c) Results for simulated data D6: 50 bp short-reads with Sanger-reads

	Produced (kb)				Aligned (kb)					
Assembler	N50	NG50	Max	Total	N50	NG50	Max	Total	%Aligned	%Covered
Telescoper	40.0	40.0	41.0	1245	39.6	38.6	40.0	1158	92.9	88.8
ABySS	22.0	21.0	39.0	1194	22.8	21.8	39.0	1154	96.6	83.6
ALLPATHS2	29.0	21.0	39.0	886	29.7	22.0	39.1	880	99.3	68.7
SGA	21.0	20.0	39.0	1110	22.6	20.1	40.0	1078	97.1	82.1
SOAPdenovo	20.7	14.0	39.0	936	20.7	14.0	39.6	928	99.1	72.5
Velvet	24.0	17.0	59.0	1000	21.4	11.0	33.8	895	89.5	69.6
Celera	27.5	24.0	41.0	930	27.6	23.8	37.0	917	98.5	71.5

(d) Results for simulated data D7: 101 bp short-reads with Sanger-reads

Table A.3: Summary of results based on simulated data from 32 telomeric regions each of length 40 kb. "%Aligned" is the ratio of Total Aligned to Total Produced, while "%Covered" is the fraction of the telomeric regions covered by contigs. Celera is excluded from both (a) and (b) since it is a Sanger-read assembler. SGA is excluded from (a) and (c) since it did not run on 50 bp data.

Appendix B

Appendix: Information-Theoretic Requirements

B.1 Supplementary Material

In this supplementary material, we display the output of our pipeline for 9 datasets (in addition to hc19, whose output is in the introduction, and the GAGE datasets R. sphaeroides, S. Aureus, and hc14). For each dataset we plot

$$\log(1 + a_\ell),$$

the log of one plus the number of repeats of each length ℓ. From the repeat statistics a_m, $b_{m,n}$, and c_m, we produce a feasibility plot. The thick black line denotes the lower bound on feasible N, L, and the green line is the performance achieved by MultiBridging.

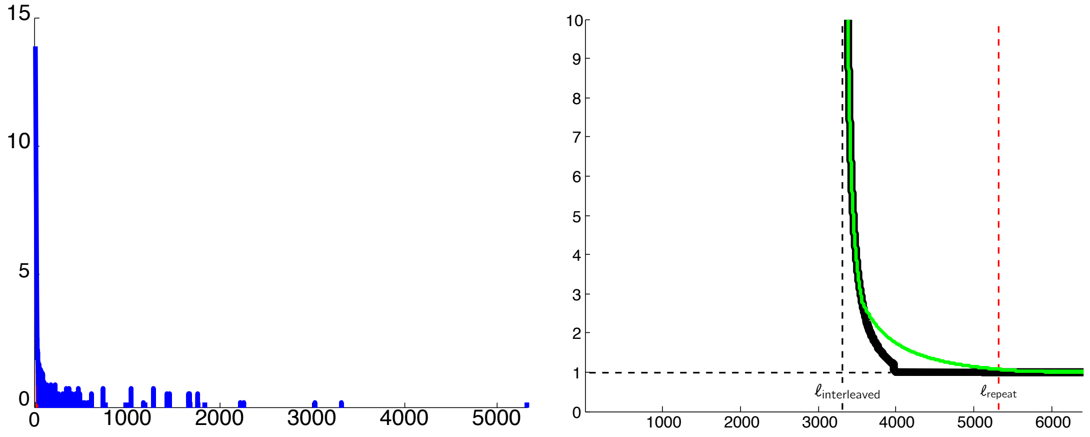


Figure B.1: Lactofidus. $G = 2,078,001$, $\ell_{\text{triple}} = 3027$, $\ell_{\text{interleaved}} = 3313$, $\ell_{\text{repeat}} = 5321$.

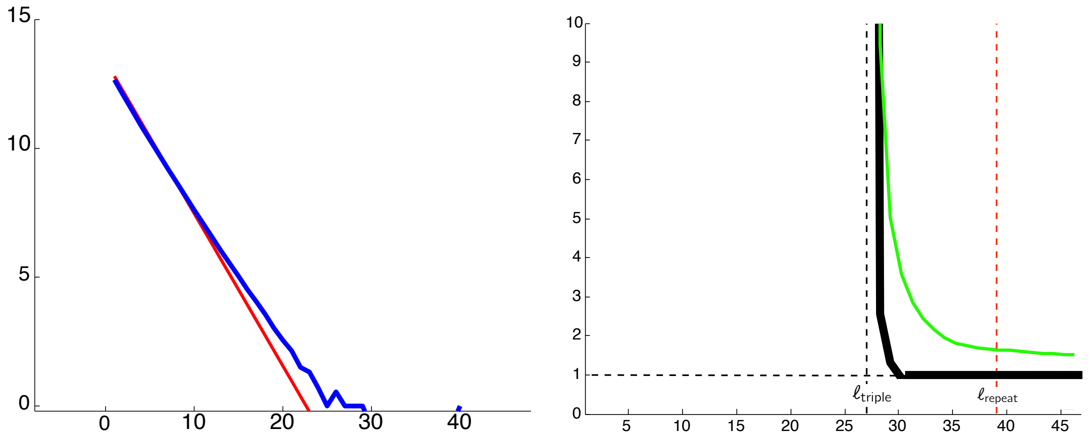


Figure B.2: Buchnera. $G = 642,122$, $\ell_{\text{triple}} = 27$, $\ell_{\text{interleaved}} = 23$, $\ell_{\text{repeat}} = 39$.

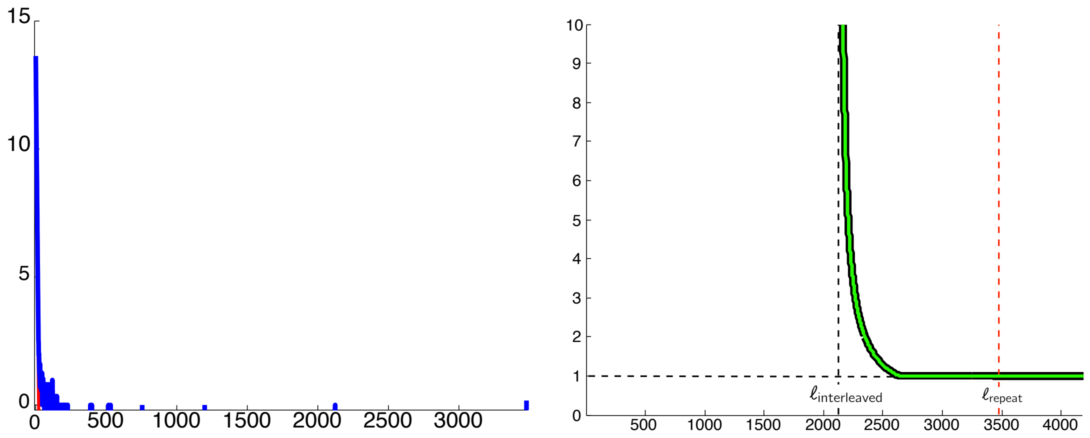


Figure B.3: Heli51. $G = 1,589,954$, $\ell_{\text{triple}} = 219$, $\ell_{\text{interleaved}} = 2122$, $\ell_{\text{repeat}} = 3478$.

B.2 Lower bounds on coverage depth

The lower bounds are based on a generalization of Ukkonen's condition to shotgun sequencing, as described in Theorem 1. The proof of Theorem 1 follows by a straightforward modification to the argument in Ukkonen [1992] and is omitted here.

Theorem 1. *Given a DNA sequence* $\mathbf{s}$ *and a set of reads, if there is a pair of interleaved repeats or a triple repeat whose copies are all unbridged, then there is another sequence* $\mathbf{s}'$ *of the same length under which the likelihood of observing the reads is the same.*

Lower bound due to interleaved repeats

In this section we derive a necessary condition on N and L in order that the probability of correct reconstruction be at least $1 - \epsilon$.

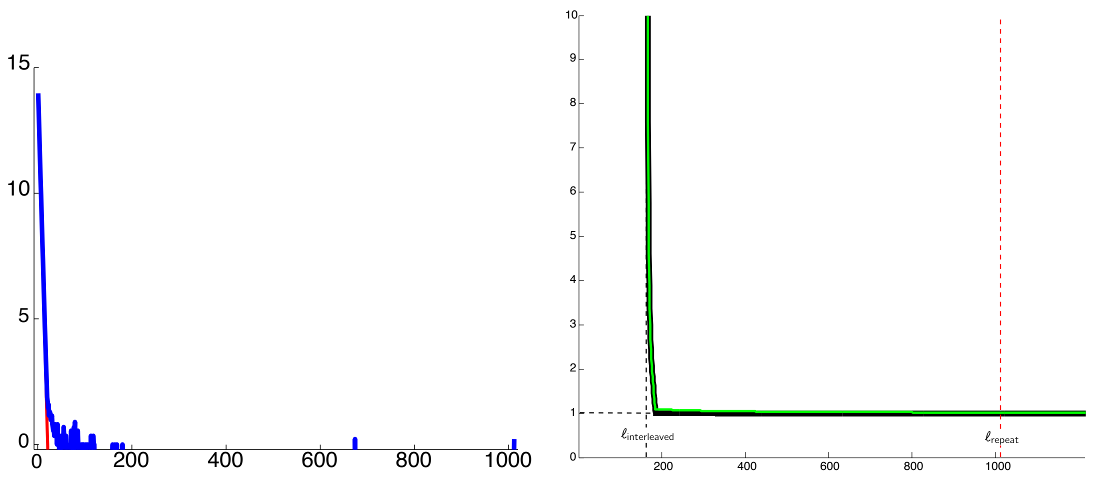


Figure B.4: Salmonella. $G = 2,215,568$, $\ell_{\text{triple}} = 112$, $\ell_{\text{interleaved}} = 163$, $\ell_{\text{repeat}} = 1011$.

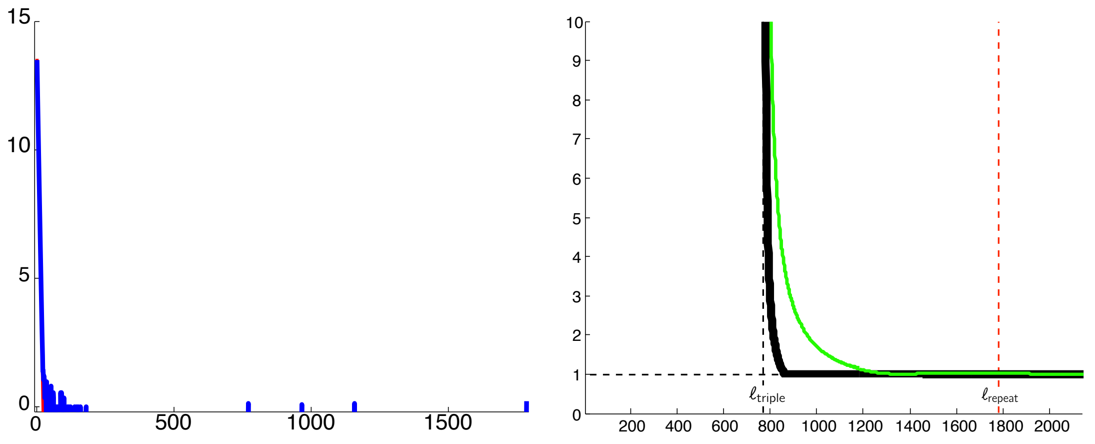


Figure B.5: Perkinsus marinus. $G = 1,440,372$, $\ell_{\text{triple}} = 770$, $\ell_{\text{interleaved}} = 92$, $\ell_{\text{repeat}} = 1784$.

Recall that a pair of repeats, one at positions t_1, t_3 with $t_1 < t_3$ and the second at positions t_2, t_4 with $t_2 < t_4$, is *interleaved* if $t_1 < t_2 < t_3 < t_4$ or $t_2 < t_1 < t_4 < t_3$. From the DNA we may extract a (symmetric) matrix of interleaved repeat statistics b_{mn}, the number of pairs of interleaved repeats of lengths m and n.

We proceed by fixing both N and L and checking whether or not unbridged interleaved repeats occur with probability higher than ϵ. We will break up repeats into 2 categories: repeats of length at least $L-1$ (these are always unbridged), and repeats of length less than $L-1$ (these are sometimes unbridged). We assume that $L > \ell_{\text{interleaved}} + 1$, or equivalently $b_{ij} = 0$ for all $i, j \geq L-1$, since otherwise there are (with certainty) unbridged interleaved repeats and Ukkonen's condition is violated.

First, we estimate the probability of error due to interleaved repeats of lengths $i < L-1$ and $j \geq L-1$. The repeat of length j is too long to be bridged, so an error occurs if the

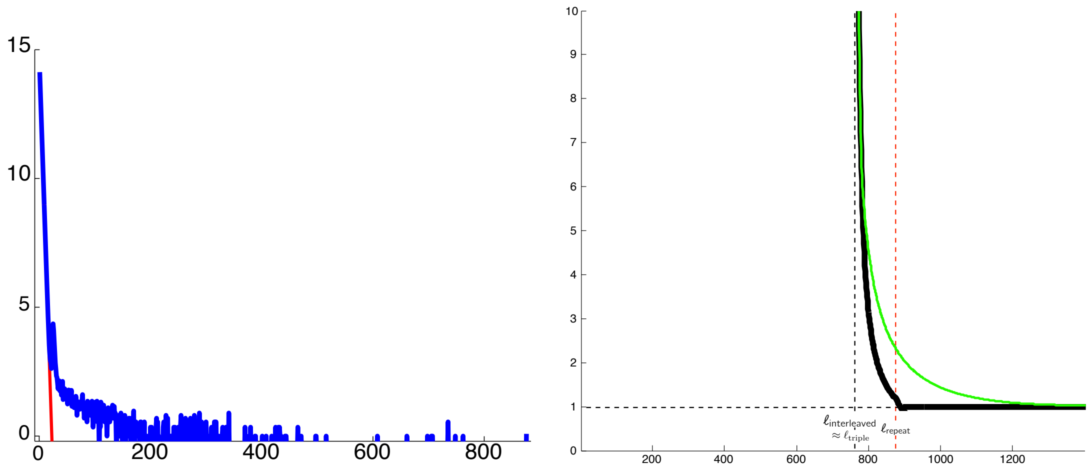


Figure B.6: Sulfolobus islandicus. $G = 2,655,198$, $\ell_{\text{triple}} = 734$, $\ell_{\text{interleaved}} = 761$, $\ell_{\text{repeat}} = 875$.

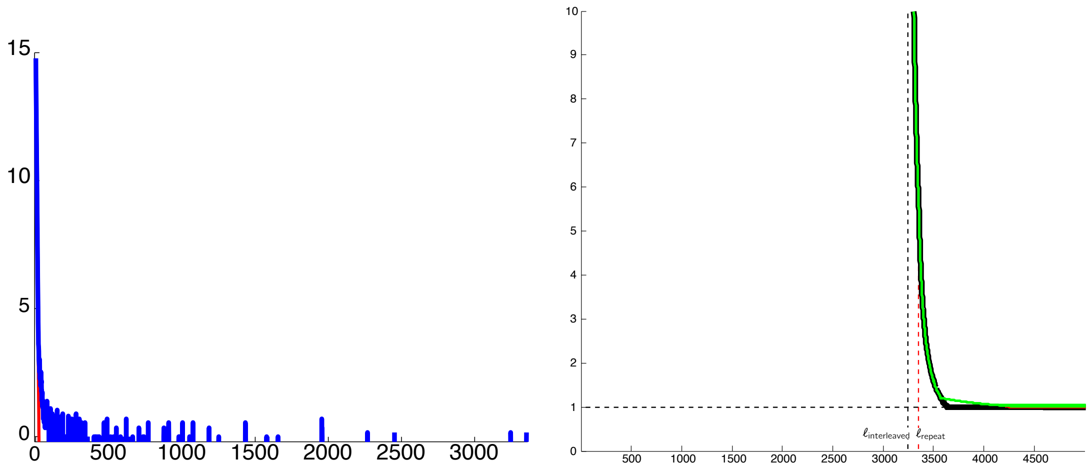


Figure B.7: Ecoli536. $G = 4,938,920$, $\ell_{\text{triple}} = 2267$, $\ell_{\text{interleaved}} = 3245$, $\ell_{\text{repeat}} = 3353$.

repeat of length i is unbridged. For a repeat, as long as the two copies' locations are not too nearby[1], each copy is bridged independently and hence the probability that both copies of the repeat of length i are unbridged is $p_i^{\text{unbridged}} = e^{-2\frac{N}{G}(L-i-1)}$. (Recall that a repeat is unbridged if *both* copies are unbridged.)

[1]More precisely, for the two copies of a a repeat of length ℓ to be bridged independently requires that no single read can bridge them both. This means their locations t and $t+d$ must have separation $d \geq L - \ell - 2$.

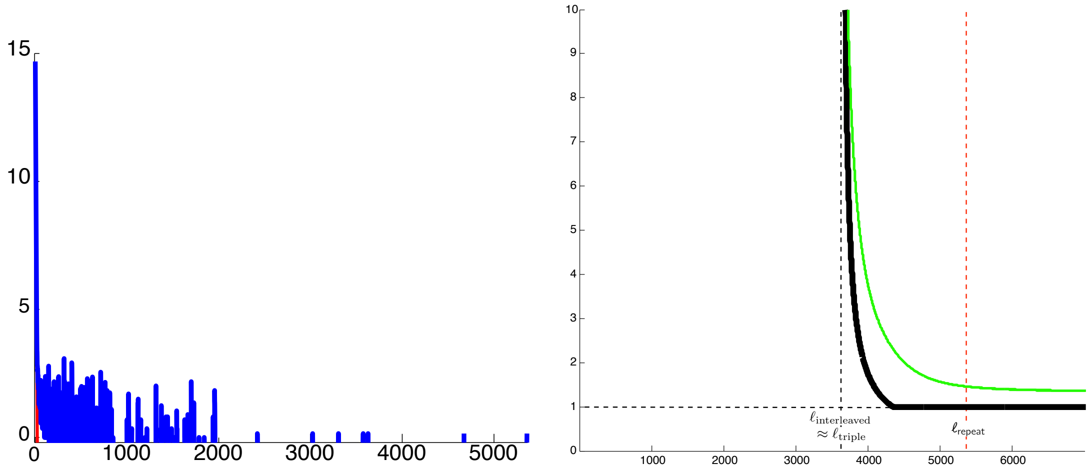


Figure B.8: Yesnina. $G = 4,504,254$, $\ell_{\text{triple}} = 3573$, $\ell_{\text{interleaved}} = 3627$, $\ell_{\text{repeat}} = 5358$.

A union bound estimate[2] gives a probability of error

$$P_{\text{error}} \approx \frac{1}{2} \sum_{\substack{m<L-1 \\ n\geq L-1}} b_{mn} e^{-2\lambda(L-m-1)} . \tag{B.1}$$

Requiring the error probability to be less than ϵ and solving for L gives the necessary condition

$$L \geq \frac{1}{2\lambda} \log \frac{\gamma_1}{2\epsilon} = \frac{G}{2N} \log \frac{\gamma_1}{2\epsilon} , \tag{B.2}$$

where $\gamma_1 := \sum_{\substack{m<L-1 \\ n\geq L-1}} b_{mn} e^{2(N/G)(m+1)}$ is a simple function of the interleaved repeat statistic b_{mn}.

We now estimate the probability of error due to interleaved repeat pairs in which both repeats are shorter than $L-1$. In this case only one repeat of each interleaved repeat pair must be bridged. Again a union bound estimate gives

$$P_{\text{error}} \approx \frac{1}{2} \sum_{m,n<L-1} b_{mn} e^{-2\lambda(L-m-1)} e^{-2\lambda(L-n-1)} .$$

Requiring the error probability to be less than ϵ gives the necessary condition

$$L \geq \frac{1}{4\lambda} \log \frac{\gamma_2}{2\epsilon} = \frac{G}{4N} \log \frac{\gamma_2}{2\epsilon} , \tag{B.3}$$

where $\gamma_2 := \sum_{m,n<L-1} b_{mn} e^{2(N/G)(m+n+2)}$ and similarly to γ_1 is computed from b_{mn}.

[2]The union bound on probabilities gives an *upper bound*, so its use here is only an approximation. To get a rigorous lower bound we can use the inclusion-exclusion principle, but the difference in the two computations is negligible for the data we observed. For ease of exposition we opt to present the simpler union bound estimate.

Lower bound due to triple repeats

We translate the generalized Ukkonen's condition prohibiting unbridged triple repeats into a condition on L and N. Let c_m denote the number of triple repeats of length m. Then a union bound estimate gives

$$\mathbb{P}(\mathcal{E}) \approx \frac{1}{2}\sum_m c_m e^{-3\lambda(L-m-1)} \,. \tag{B.4}$$

Requiring $\mathbb{P}(\mathcal{E}) \leq \epsilon$ and solving for L gives

$$L \geq \frac{1}{3\lambda}\log\frac{\gamma_3}{2\epsilon} = \frac{G}{3N}\log\frac{\gamma_3}{2\epsilon}\,, \tag{B.5}$$

where $\gamma_3 := \sum_m c_m e^{3(N/G)(m+1)}$.

Remark 7. *As discussed here and in Section 4.3, if the DNA sequence is not covered by the reads or there are unbridged interleaved or triple repeats, then reconstruction is not possible. But there is another situation which must be ruled out. Without knowing its length a priori, it is impossible to know how many copies of the DNA sequence are actually present: if the sequence* $\mathbf{s}$ *to be assembled consists of multiple concatenated copies of a shorter sequence, rather than just one copy, the probability of observing any set of reads will be the same. Since it is unlikely that a true DNA sequence will consist of the same sequence repeated multiple times, we assume this is not the case throughout the paper. Equivalently, if* $\mathbf{s}$ *does consist of multiple concatenated copies of a shorter sequence, we are content to reconstruct a single copy. If available, knowledge of the approximate length of* $\mathbf{s}$ *would then allow to reconstruct.*

B.3 Proofs for algorithms

Proof of Theorem 2 (Greedy)

The greedy algorithm's underlying data structure is the overlap graph, where each node represents a read and each (directed) edge $(\mathbf{x}, \mathbf{y})$ is labeled with the overlap $\text{ov}(\mathbf{x}, \mathbf{y})$ (defined as the the length of the shared prefix/suffix) between the incident nodes' reads. For a node $\mathbf{v}$, the in-degree [out-degree] is the number of edges in the graph directed towards [away from] $\mathbf{v}$. The greedy algorithm is described as follows.

Algorithm 2 GREEDY. Input: reads $\mathcal{R}$. Output: sequence $\hat{\mathbf{s}}$.

```
1. For each read with sequence $\mathbf{x}$, form a node with label $\mathbf{x}$.
Greedy steps 2-3:
2. Consider all pairs of nodes $\mathbf{x}_1, \mathbf{x}_2$ in $\mathbb{G}$ satisfying $d_{\text{out}}(\mathbf{x}_1) = d_{\text{in}}(\mathbf{x}_2) = 0$, and add an edge $(\mathbf{x}_1, \mathbf{x}_2)$ with largest value $\text{ov}(\mathbf{x}_1, \mathbf{x}_2)$.
3. Repeat Step 2 until no candidate pair of nodes remains.
Finishing step:
4. Output the sequence corresponding to the unique cycle in $\mathbb{G}$.
```

Theorem 2. *Given a sequence* $\mathbf{s}$ *and a set of reads,* GREEDY *returns* $\mathbf{s}$ *if every repeat is bridged.*

Proof. We prove the contrapositive. Suppose GREEDY makes its first error in merging reads $\mathbf{r}_i$ and $\mathbf{r}_j$ with overlap $\text{ov}(\mathbf{r}_i, \mathbf{r}_j) = \ell$. Now, if $\mathbf{r}_j$ is the successor to $\mathbf{r}_i$, then the error is due to incorrectly aligning the reads; the other case is that $\mathbf{r}_j$ is not the successor of $\mathbf{r}_i$. In the first case, the subsequence $\mathbf{s}_{t_j}^{\ell}$ is repeated at location $\mathbf{s}_{t_i+L-\ell}^{\ell}$, and no read bridges either repeat copy.

In the second case, there is a repeat $\mathbf{s}_{t_j}^{\ell} = \mathbf{s}_{t_i+L-\ell}^{\ell}$. If $\mathbf{s}_{t_i+L-\ell}^{\ell}$ is bridged by some read $\mathbf{r}_k$, then $\mathbf{r}_i$ has overlap at least $\ell + 1$ with $\mathbf{r}_k$, implying that read $\mathbf{r}_i$ has already found its successor before step ℓ (either $\mathbf{r}_k$ or some other read with even higher overlap). A similar argument shows that $\mathbf{s}_{t_j}^{\ell}$ cannot be bridged, hence there is an unbridged repeat. □

Proofs for K-mer algorithms

Background

We give some mathematical background leading to the proof of Theorem 3 (restated below).

Lemma 8. *Fix an arbitrary* K *and form the* K*-mer graph from the* $(K+1)$*-spectrum* $\mathcal{S}_{K+1}$*. The sequence* $\mathbf{s}$ *corresponds to a unique cycle* $\mathcal{C}(\mathbf{s})$ *traversing each edge at least once.*

To prove the lemma, note that all $(K+1)$-mers in $\mathbf{s}$ correspond to edges and adjacent $(K+1)$-mers in $\mathbf{s}$ are represented by adjacent edges. An induction argument shows that $\mathbf{s}$ corresponds to a cycle. The cycle traverses all the edges, since each edge represents a unique $(K+1)$-mer in $\mathbf{s}$.

In both SBH and shotgun sequencing the number of times each edge e is traversed by $\mathcal{C}(\mathbf{s})$ (henceforth called the *multiplicity* of e) is unknown a priori, and finding this number is part of the reconstruction task. Repeated $(K+1)$-mers in $\mathbf{s}$ correspond to edges in the K-mer graph traversed more than once by $\mathcal{C}(\mathbf{s})$, i.e. having multiplicity greater than one. In order to estimate the multiplicity, previous works seek a solution to the so-called Chinese Postman Problem (CPP), in which the goal is to find a cycle of the shortest total length traversing every edge in the graph (see e.g. Pevzner [1989], Idury and Waterman [1995], Pevzner et al. [2001], Medvedev and Brudno [2009]). It is not obvious under what conditions the CPP solution *correctly* assigns multiplicities in agreement with $\mathcal{C}(\mathbf{s})$. For our purposes, as we will see in Theorem 3, the multiplicity estimation problem can be sidestepped (thereby avoiding solving CPP) through a modification to the K-mer graph.

Ignoring the issue of edge multiplicities for a moment, Pevzner [1995] showed for the SBH model that if the edge multiplicities are known with multiple copies of each edge included according to the multiplicities, and moreover Ukkonen's condition is satisfied, then there is a *unique Eulerian cycle* in the K-mer graph and the Eulerian cycle corresponds to the original sequence. (An Eulerian cycle is a cycle traversing each edge exactly once.) Pevzner's algorithm is thus to find an Eulerian cycle and read off the corresponding sequence. Both steps can be done efficiently.

Lemma 9 (Pevzner [1995]). *In the SBH setting, if the edge multiplicities are known, then there is a unique Eulerian cycle in the K-mer graph with $K = L - 1$ if and only if there are no unbridged interleaved repeats or unbridged triple repeats.*

Most practical algorithms (e.g. Idury and Waterman [1995], MacCallum et al. [2009], Zerbino and Birney [2008]) condense unambiguous paths (called unitigs by Myers [2000] in a slightly different setting) for computational efficiency. The more significant benefit for us, as shown in Theorem 3, is that if Ukkonen's condition is satisfied then condensing the graph obviates the need to estimate multiplicities. Condensing a K-mer graph results in a graph of the following type.

Definition 10 (Sequence graph). *A* sequence graph *is a graph in which each node is labeled with a subsequence, and edges $(\mathbf{u}, \mathbf{v})$ are labeled with an* overlap $a_{\mathbf{uv}}$ *such the subsequences $\mathbf{u}$ and $\mathbf{v}$ overlap by $a_{\mathbf{uv}}$ (the overlap is not necessarily maximal). In other words, an edge label $a_{\mathbf{uv}}$ on $e = (\mathbf{u}, \mathbf{v})$ indicates that the $a_{\mathbf{uv}}$-length suffix of $\mathbf{u}$ is equal to the $a_{\mathbf{uv}}$-length prefix of $\mathbf{v}$.*

The sequence graph generalizes both the overlap graph used by Greedy in Section 4.4 (nodes correspond to reads, and edge overlaps are *maximal* overlaps) as well as the K-mer algorithms discussed in this section (nodes correspond to K-mers, and edge overlaps are $K - 1$).

In order to speak concisely about concatenated sequences in the sequence graph, we extend the notation $\mathbf{s}_t^\ell$ (denoting the length-ℓ subsequence of the DNA sequence $\mathbf{s}$ starting at position t) which was introduced in Section 4.3; we abuse notation slightly, and write $\mathbf{s}_t^{\text{end}}$ to indicate the subsequence of $\mathbf{s}$ starting at position t and having length so that its end coincides with the end of $\mathbf{s}$.

We will perform two basic operations on the sequence graph. For an edge $e = (\mathbf{u}, \mathbf{v})$ with overlap $a_{\mathbf{uv}}$, *merging* $\mathbf{u}$ and $\mathbf{v}$ along e produces the concatenation $\mathbf{u}_1^{\text{end}}\mathbf{v}_{a_{\mathbf{uv}}+1}^{\text{end}}$. *Contracting* an edge $e = (\mathbf{u}, \mathbf{v})$ entails two steps (c.f. Fig. 4.6): first, merging $\mathbf{u}$ and $\mathbf{v}$ along e to form a new node $\mathbf{w} = \mathbf{u}_1^{\text{end}}\mathbf{v}_{a_{\mathbf{uv}}+1}^{\text{end}}$, and, second, edges to $\mathbf{u}$ are replaced with edges to $\mathbf{w}$, and edges from $\mathbf{v}$ are replaced by edges from $\mathbf{w}$. We will only contract edges $(\mathbf{u}, \mathbf{v})$ with $d_{\text{out}}(\mathbf{u}) = d_{\text{in}}(\mathbf{v}) = 1$.

The condensed graph is defined next.

Definition 11 (Condensed sequence graph)**.** *The* condensed sequence graph *replaces unambiguous paths by single nodes. Concretely, any edge $e = (u, v)$ with $d_{out}(u) = d_{in}(v) = 1$ is contracted, and this is repeated until no candidate edges remain.*

For a path $\mathcal{P} = \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_q$ in the original graph, the corresponding path in the condensed graph is obtained by contracting an edge $(\mathbf{v}_i, \mathbf{v}_{i+1})$ whenever it is contracted in the graph, replacing the node $\mathbf{v}_1$ by $\mathbf{w}$ whenever an edge $(\mathbf{u}, \mathbf{v}_1)$ is contracted to form $\mathbf{w}$, and similarly for the final node $\mathbf{v}_q$. It is impossible for an intermediate node $\mathbf{v}_i$, $2 \leq i < q$, to be merged with a node outside of $\mathcal{P}$, as this would violate the condition $d_{\text{out}}(u) = d_{\text{in}}(v) = 1$ for edge contraction in Defn. 11.

In the condensed sequence graph $\mathbb{G}$ obtained from a sequence $\mathbf{s}$, nodes correspond to subsequences via their labels, and paths in $\mathbb{G}$ correspond to subsequences in $\mathbf{s}$ via merging the constituent nodes along the path. If the subsequence corresponding to a node $\mathbf{v}$ appears twice or more in $\mathbf{s}$, we say that $\mathbf{v}$ corresponds to a repeat. Conversely, subsequences of length $\ell \geq K$ in $\mathbf{s}$ correspond to paths $\mathcal{P}$ of length $\ell - K + 1$ in the K-mer graph, and thus by the previous paragraph also to paths in the condensed graph $\mathbb{G}$.

We record a few simple facts about the condensed sequence graph obtained from a K-mer graph.

Lemma 12. *Let $\mathbb{G}_0$ be the K-mer graph constructed from the $(K + 1)$-spectrum of $\mathbf{s}$ and let $\mathcal{C}_0 = \mathcal{C}_0(\mathbf{s})$ be the cycle corresponding to $\mathbf{s}$. In the condensed graph $\mathbb{G}$, let $\mathcal{C}$ be the cycle obtained from $\mathcal{C}_0$ by contracting the same edges as those contracted in $\mathbb{G}_0$.*

1. *Edges in $\mathbb{G}_0$ can be contracted in any order, resulting in the same graph $\mathbb{G}$, so the condensed graph is well-defined. Similarly $\mathcal{C}$ is well-defined.*

2. *The cycle $\mathcal{C}$ in $\mathbb{G}$ corresponds to $\mathbf{s}$ and is the unique such cycle.*

3. *The cycle $\mathcal{C}$ in $\mathbb{G}$ traverses each edge at least once.*

Theorem 3. *Let $\mathcal{S}_{K+1}$ be the $(K+1)$-spectrum of $\mathbf{s}$ and $\mathbb{G}_0$ be the K-mer graph constructed from $\mathcal{S}_{K+1}$, and let $\mathbb{G}$ be the condensed sequence graph obtained from $\mathbb{G}_0$. If Ukkonen's condition is satisfied, i.e. there are no triple repeats or interleaved repeats of length at least K, then there is a unique Eulerian cycle $\mathcal{C}$ in $\mathbb{G}$ and $\mathcal{C}$ corresponds to $\mathbf{s}$.*

Proof. We will show that if Ukkonen's condition is satisfied, the cycle $\mathcal{C} = \mathcal{C}(\mathbf{s})$ in $\mathbb{G}$ corresponding to $\mathbf{s}$ (constructed in Lemma 12) traverses each edge exactly once in the condensed K-mer graph, i.e. $\mathcal{C}$ is Eulerian. Arguments by Pevzner [1995] show that if there are multiple Eulerian cycles then Ukkonen's condition is violated, so it is sufficient to prove that $\mathcal{C}$ is Eulerian. As noted in Lemma 12, $\mathcal{C}$ traverses each edge at least once, and thus it remains only to show that $\mathcal{C}$ traverses each edge *at most* once.

To begin, let $\mathcal{C}_0$ be the cycle corresponding to $\mathbf{s}$ in the original K-mer graph $\mathbb{G}_0$. We argue that every edge $(\mathbf{u}, \mathbf{v})$ traversed twice by $\mathcal{C}_0$ in the K-mer graph $\mathbb{G}_0$ has been contracted in the condensed graph $\mathbb{G}$ and hence in $\mathcal{C}$. Note that the cycle $\mathcal{C}_0$ does not traverse any node three times in $\mathbb{G}_0$, for this would imply the existence of a triple repeat of length K, violating the hypothesis of the Lemma. It follows that the node $\mathbf{u}$ cannot have two outgoing edges in $\mathbb{G}_0$ as $\mathbf{u}$ would then be traversed three times; similarly, $\mathbf{v}$ cannot have two incoming edges. Thus $d_{\text{out}}(\mathbf{u}) = d_{\text{in}}(\mathbf{v}) = 1$ and, as prescribed in Defn. 11, the edge $(\mathbf{u}, \mathbf{v})$ has been contracted. □

Proofs for SimpleBridging

Since bridging reads extend one base to either end of a repeat, it will be convenient to use the following notation for extending sequences: Given an X-node $\mathbf{v}$ with an incoming edge $(\mathbf{p}, \mathbf{v})$ and an outgoing edge $(\mathbf{v}, \mathbf{q})$, let

$$\mathbf{v}^{\to\mathbf{q}} = \mathbf{v}\,\mathbf{q}^1_{a_{\mathbf{vq}}+1}, \quad \text{and} \quad {}^{\mathbf{p}\to}\mathbf{v} = \mathbf{p}^1_{\text{end}-a_{\mathbf{pv}}}\mathbf{v}\,. \tag{B.6}$$

Here $\mathbf{v}^{\to\mathbf{q}}$ denotes the subsequence $\mathbf{v}$ appended with the single next base in the merging of $\mathbf{v}$ and $\mathbf{q}$ and ${}^{\mathbf{p}\to}\mathbf{v}$ the subsequence $\mathbf{v}$ prepended with the single previous base in the merging of $\mathbf{p}$ and $\mathbf{v}$. For example, if $\mathbf{v}$ = ATTC, $\mathbf{p}$ = TCAT, $a_{\mathbf{pv}} = 2$, $\mathbf{q}$ = TTCGCC, and $a_{\mathbf{vq}} = 3$, then $\mathbf{v}^{\to\mathbf{q}}$ = ATTCG, ${}^{\mathbf{p}\to}\mathbf{v}$ = CATTC, and ${}^{\mathbf{p}\to}\mathbf{v}^{\to\mathbf{q}}$ = CATTCG.

The idea is that a bridging read is consistent with only one pair ${}^{\mathbf{p}\to}\mathbf{v}$ and $\mathbf{v}^{\to\mathbf{q}}$ and thus allows to match up edge $(\mathbf{p}, \mathbf{v})$ with $(\mathbf{v}, \mathbf{q})$. This is recorded in the following lemma.

Lemma 13. *Suppose $\mathcal{C}$ corresponds to a sequence $\mathbf{s}$ in a condensed sequence graph $\mathbb{G}$. If a read $\mathbf{r}$ bridges an X-node $\mathbf{v}$, then there are unique edges $(\mathbf{p}, \mathbf{v})$ and $(\mathbf{v}, \mathbf{q})$ such that ${}^{\mathbf{p}\to}\mathbf{v}$ and $\mathbf{v}^{\to\mathbf{q}}$ are adjacent in $\mathbf{r}$.*

SimpleBridging is described as follows.

Algorithm 3 SimpleBridging. Input: reads $\mathcal{R}$, parameter K. Output: sequence $\hat{\mathbf{s}}$.

```
K-mer steps 1-3:
1. For each subsequence $\mathbf{x}$ of length $K$ in a read, form a node with label $\mathbf{x}$.
2. For each read, add edges between nodes representing adjacent $K$-mers in the read.
3. Condense the graph as described in Defn. 11.
4. Bridging step: See Fig. 4.7. While there exists an X-node $\mathbf{v}$ with $d_{\text{in}}(\mathbf{v}) = d_{\text{out}}(\mathbf{v}) = 2$
   bridged by some read $\mathbf{r}$: (i) Remove $\mathbf{v}$ and edges incident to it. Add duplicate nodes $\mathbf{v}_1, \mathbf{v}_2$.
   (ii) Choose the unique $\mathbf{p}_i$ and $\mathbf{q}_j$ s.t. ${}^{\mathbf{p}_i\to}\mathbf{v}$ and $\mathbf{v}^{\to\mathbf{q}_j}$ are adjacent in $\mathbf{r}$ and add edges $(\mathbf{p}_i, \mathbf{v}_1)$
   and $(\mathbf{v}_1, \mathbf{q}_j)$. Choose the unused $\mathbf{p}_i$ and $\mathbf{q}_j$, add edges $(\mathbf{p}_i, \mathbf{v}_2)$ and $(\mathbf{v}_2, \mathbf{q}_j)$. (iii) Condense
   the graph.
5. Finishing step: Find an Eulerian cycle in the graph and return the corresponding sequence.
```

Proofs for MultiBridging

In this subsection we recall Theorem 6 stating sufficient conditions for correct reconstruction, and derive the corresponding required coverage depth and read length to meet a target probability of correct reconstruction. The subsection concludes with a proof that the sufficient conditions are correct.

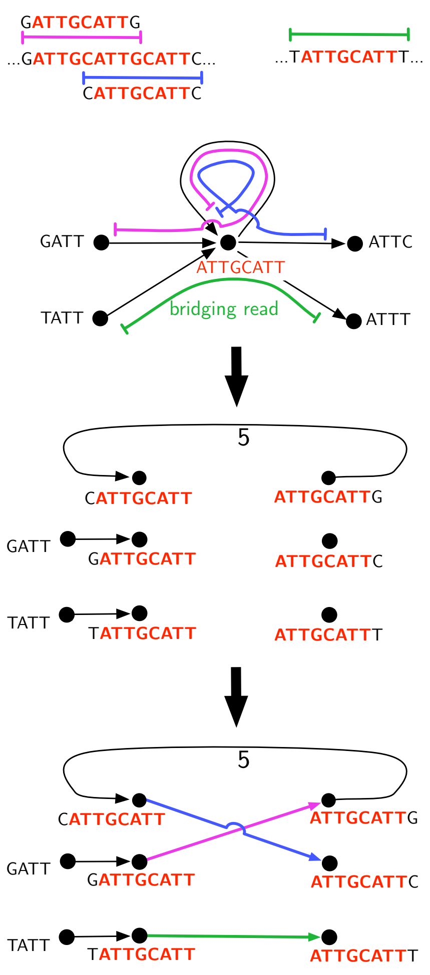


Figure B.9: Resolution of X-node with a self-loop.

Theorem 6. *The algorithm* MULTIBRIDGING *reconstructs the sequence* $\mathbf{s}$ *if:*

(a) all interleaved repeats are bridged

(b) all triple repeats are ***all-bridged***

(c) the sequence is covered by the reads.

Remark 14. *Unlike the previous K-mer algorithms,* DEBRUIJN *and* SIMPLEBRIDGING*, it is unnecessary to specify a parameter K for* MULTIBRIDGING*. Implicitly* MULTIBRIDGING *uses $K = 1$, which makes the condition that reads overlap by K equivalent to coverage of the genome.*

Figure 4.2 plots the performance of MULTIBRIDGING, obtained by solving for the relationship between G, N, L, and ϵ in order to satisfy the conditions of Lemma 6. We first perform the requisite calculations, and then prove the Lemma.

Condition (a) is already dealt with in (B.2) and (B.3), and Condition (c) amounts to the requirement that $\frac{N}{N_{\text{LW}}} \geq 1$.

We turn to Condition (b) that all triple repeats are all-bridged. Let c_m denote the number of triple repeats of length m. A union bound estimate over triple repeats for the event that one such triple repeat fails to be all-bridged gives

$$P_{\text{error}} \approx \sum_m 3 \cdot c_m e^{-\lambda(L-m-1)^+}\,, \tag{B.7}$$

and requiring $P_{\text{error}} \leq \epsilon$ and solving for L yields

$$L \geq \frac{1}{\lambda} \log \frac{\gamma_3}{\epsilon} = \frac{G}{N} \log \frac{\gamma_3}{\epsilon}\,, \tag{B.8}$$

where $\gamma_3 := \sum_m 3c_m e^{(N/G)\cdot(m+1)}$ is computed from the triple repeat statistics c_m.

In order to understand the cost of all-bridging triple repeats, compared to simply bridging one copy as required by our lower bound, it is instructive to study the effect of the single longest triple repeat. Setting $c_{\ell_{\text{triple}}} = 1$ and $c_m = 0$ for $m \neq \ell_{\text{triple}}$ makes $\gamma_3 = 3e^{(N/G)\cdot(\ell_{\text{triple}}+1)}$ in (B.8) and

$$L \geq L_3^{\text{all}} := \ell_{\text{triple}} + 1 + \frac{G}{N} \log 3\epsilon^{-1}\,. \tag{B.9}$$

Bridging the longest triple repeat, as shown in Section B.2, requires

$$L \geq L_3 := \ell_{\text{triple}} + 1 + \frac{G}{3N} \log \epsilon^{-1}\,. \tag{B.10}$$

Solving for N in equations (B.10) and (B.9) gives

$$N_3 \geq \frac{G}{3} \cdot \frac{\log \epsilon^{-1}}{L - \ell_{\text{triple}} - 1} \tag{B.11}$$

$$N_3^{\text{all}} \geq G \cdot \frac{\log \epsilon^{-1} + \log 3}{L - \ell_{\text{triple}} - 1}\,. \tag{B.12}$$

The ratio is

$$\frac{N_3^{\text{all}}}{N_3} = 3 \cdot \frac{\log 3\epsilon^{-1}}{\log \epsilon^{-1}} \approx 3.72 \quad \text{for } \epsilon = 10^{-2}\,. \tag{B.13}$$

This means that if the longest triple repeat is dominant, then for L slightly larger than ℓ_{triple}, MultiBridging needs a coverage depth approximately 3.72 times higher than required by our lower bound.

The remainder of this subsection is devoted to the proving Lemma 6.

We will use $m_{\mathcal{C}}(\mathbf{v})$ to denote the multiplicity (traversal count) a cycle $\mathcal{C}$ assigns a node $\mathbf{v}$. The multiplicity $m_{\mathcal{C}}(\mathbf{v})$ is also equal to the number of times the subsequence $\mathbf{v}$ appears in the sequence corresponding to $\mathcal{C}$. For an edge e, we can similarly let $m_{\mathcal{C}}(e)$ be the number of times $\mathcal{C}$ traverses the edge. The following key lemma relates node multiplicities with the existence of X-nodes.

Lemma 15. *Let $\mathcal{C}$ be a cycle in a condensed sequence graph $\mathbb{G}$, where $\mathbb{G}$ itself is not a cycle, traversing every edge at least once. If $\mathbf{v}$ is a node with maximum multiplicity at least 2, i.e. $m_{\mathcal{C}}(\mathbf{v}) = \max_{u\in\mathbb{G}} m_{\mathcal{C}}(\mathbf{u}) \geq 2$, then $\mathbf{v}$ is an X-node. As a consequence, if $m_{\mathcal{C}}(\mathbf{v}) \geq 3$ for some $\mathbf{v}$, i.e. $\mathcal{C}$ traverses some node at least three times, then $m_{\mathcal{C}}(\mathbf{u}) \geq 3$ for some X-node $\mathbf{u}$.*

Proof. Let $\mathbf{v}$ be a node with maximum multiplicity $m_{\mathcal{C}}(\mathbf{v}) = \max_{\mathbf{u}\in\mathbb{G}} m_{\mathcal{C}}(\mathbf{u})$. We will show that $\mathbf{v}$ is an X-node, i.e. $d_{\text{out}}(\mathbf{v}) \geq 2$ and $d_{\text{in}}(\mathbf{v}) \geq 2$.

We prove that $d_{\text{out}}(\mathbf{v}) \geq 2$ by supposing that $d_{\text{out}}(\mathbf{v}) = 1$ and deriving a contradiction. Denote the outgoing edge from $\mathbf{v}$ by $e = (\mathbf{v}, \mathbf{u})$, where $\mathbf{u}$ is distinct from $\mathbf{v}$ since otherwise $\mathbb{G}$ is a cycle. If $d_{\text{in}}(\mathbf{u}) \geq 2$, then $\mathbf{u}$ must be traversed more times than $\mathbf{v}$, contradicting the maximality of $m_{\mathcal{C}}(\mathbf{v})$, and if $d_{\text{in}}(\mathbf{u}) = 1$, then the existence of the edge e contradicts the fact that $\mathbb{G}$ is condensed. The argument showing that $d_{\text{in}}(\mathbf{v}) \geq 2$ is symmetric to the case $d_{\text{in}}(\mathbf{v}) \geq 2$. □

Proof of Lemma 6. We assume that all triple repeats are all-bridged, that there are no unbridged interleaved repeats, and that all reads overlap their successors by at least 1 base pair. We wish to show that MultiBridging returns the original sequence.

Consider the condensed sequence graph $\mathbb{G}_0$ constructed in steps 1-3 of MultiBridging. Suppose all X-nodes that are either all-bridged or correspond to bridged 2-repeats have been resolved according to repeated application of the procedure in step 4 of MultiBridging, resulting in a condensed sequence graph $\mathbb{G}$. We claim that 1) $\mathbf{s}$ corresponds to a cycle $\mathcal{C}$ in $\mathbb{G}$ traversing every edge at least once, 2) $\mathcal{C}$ is Eulerian, and 3) $\mathcal{C}$ is the unique Eulerian cycle in $\mathbb{G}$.

Proof of Claim 1. Let $\mathbb{G}_n$ be the graph after n resolution steps, and suppose that $\mathcal{C}_n$ is a cycle in $\mathbb{G}_n$ corresponding to the sequence $\mathbf{s}$ and traversing all edges. We will show that there exists a cycle $\mathcal{C}_{n+1}$ in $\mathbb{G}_{n+1}$ corresponding to $\mathbf{s}$ and traversing all edges, and that $\mathbb{G}_t = \mathbb{G}$ for a finite t, so by induction, there exists a cycle $\mathcal{C}$ in $\mathbb{G}$ corresponding to $\mathbf{s}$ and traversing all edges. The base case $n = 0$ was shown in Lemma 8. Moving on to arbitrary $n > 0$, let $\mathbf{v}$ be an X-node in $\mathbb{G}_n$ labeled as in Fig. 4.6. The X-node resolution step is constructed precisely to preserve the existence of a cycle corresponding to $\mathbf{s}$. Each traversal of $\mathbf{v}$ by the cycle $\mathcal{C}_n$ assigns an incoming edge $(\mathbf{p}_i\mathbf{v})$ to an outgoing edge $(\mathbf{v}, \mathbf{q}_j)$, and the resolution step correctly determines this pairing by the assumption on bridging reads.

Note that all X-nodes in the graph $\mathbb{G}_{n+1}$ continue to correspond to repeats in $\mathbf{s}$. The process terminates: let $\mathcal{L}(\mathbb{G}_i) = \sum_{\mathbf{v}\in\mathbb{G}_i} m_{\mathcal{C}_i}(\mathbf{v})\mathbf{1}_{m_{\mathcal{C}_i}(\mathbf{v})>1}$ and observe that $\mathcal{L}(\mathbb{G}_i)$ is strictly decreasing in i. Thus $\mathbf{s}$ corresponds to a cycle $\mathcal{C}$ in $\mathbb{G}$ traversing each edge at least once.

Proof of Claim 2. We next show that $\mathcal{C}$ is an Eulerian cycle. If $\mathbb{G}$ is itself a cycle, and $\mathbf{s}$ is not formed by concatenating multiple copies of a shorter subsequence (assumed not to be the case, see discussion at end of Section 4.3), then $\mathcal{C}$ traverses $\mathbb{G}$ exactly once and is an Eulerian cycle. Otherwise, if $\mathbb{G}$ is not a cycle, then we may apply Lemma 15 to see that any node with $m_{\mathcal{C}}(\mathbf{v}) \geq 3$ implies the existence of an X-node $\mathbf{u}$ with $m_{\mathcal{C}}(\mathbf{u}) \geq 3$. Node $\mathbf{u}$ must be all-bridged, by hypothesis, which means that an additional X-node resolution step can be applied to $\mathbb{G}$, a contradiction. Thus each node $\mathbf{v}$ in $\mathbb{G}$ has multiplicity $m_{\mathcal{C}}(\mathbf{v}) \leq 2$.

We can now argue that no edge $e = (\mathbf{u}, \mathbf{v})$ is traversed twice by $\mathcal{C}$ in the condensed sequence graph $\mathbb{G}$, as it would have been contracted. Suppose $m_{\mathcal{C}}(e) \geq 2$. The node $\mathbf{u}$ cannot have two outgoing edges as this implies $m_{\mathcal{C}}(\mathbf{u}) \geq 3$; similarly, $\mathbf{v}$ cannot have two incoming edges. Thus $d_{\text{out}}(\mathbf{u}) = d_{\text{in}}(\mathbf{v}) = 1$, but by Defn. 11 the edge $e = (\mathbf{u}, \mathbf{v})$ would have been contracted.

Proof of Claim 3. It remains to show that there is a *unique* Eulerian cycle in $\mathbb{G}$. All X-nodes in $\mathbb{G}$ must be unbridged 2-X-nodes (correspond to 2-repeats in $\mathbf{s}$), as all other X-nodes were assumed to be bridged and have thus been resolved in $\mathbb{G}$.

We will map the sequence $\mathbf{s}$ to another sequence $\mathbf{s}'$, allowing us to use the characterization of Lemma 9 for SBH with known multiplicities. Denote by $\mathbb{G}'$ the graph obtained by relabeling each node in $\mathbb{G}$ by a single unique symbol (no matter the original node label length), and setting all edge overlaps to 0. Through the relabeling, $\mathcal{C}$ corresponds to a cycle $\mathcal{C}'$ in $\mathbb{G}'$, and let $\mathbf{s}'$ be the sequence corresponding to $\mathcal{C}'$. Writing $\mathcal{S}_2'$ for the 2-spectrum of $\mathbf{s}'$, the graph $\mathbb{G}'$ is by construction precisely the 1-mer graph created from $\mathcal{S}_2'$, and there is a one-to-one correspondence between X-nodes in $\mathbb{G}'$ and unbridged repeats in $\mathbf{s}'$. Through the described mapping, every unbridged repeat in $\mathbf{s}'$ maps to an unbridged repeat in $\mathbf{s}$, with the *order of repeats preserved*.

There are multiple Eulerian cycles in $\mathbb{G}$ only if there are multiple Eulerian cycles in $\mathbb{G}'$ since the graphs have the same topology, and by Lemma 9 the latter occurs only if there are unbridged interleaved repeats in $\mathbf{s}'$, which by the correspondence in the previous paragraph implies the existence of unbridged interleaved repeats in $\mathbf{s}$. □

Truncation estimate for bridging repeats (Greedy and MultiBridging)

The repeat statistics a_m and c_m used in the algorithm performance curves are potentially overestimates. This is because a large repeat family—one with a large number of copies f—will result in a contribution $\binom{f}{2} \approx f^2/2$ to a_m and $\binom{f}{3} \approx f^3/6$ to c_m.

We focus here on deriving an estimate for the required N, L for bridging all repeats with probability $1-\epsilon$. This upper bound reduces the sensitivity to large families of short repeats. The analogous derivation for all-bridging all triple-repeats is very similar and is omitted.

Suppose there are a_m repeats of length m. The probability that some repeat is unbridged is approximately, by the union bound estimate,

$$\mathbb{P}(\mathcal{E}) \approx \sum_m a_m e^{-2\lambda(L-m)} . \tag{B.14}$$

Requiring $\mathbb{P}(\mathcal{E}) \leq \epsilon$ and solving for L gives

$$L \geq \frac{1}{2\lambda} \log \frac{\gamma}{\epsilon} = \frac{G}{2N} \log \frac{\gamma}{\epsilon} , \tag{B.15}$$

where $\gamma := \sum_m a_m e^{2(N/G)m}$. Now, if a_m *overcounts* the number of repeats for small values of m, the bound in (B.15) might be loose. In order for each read to overlap the subsequent read by x nucleotides, with probability of failure $\epsilon/2$, it suffices to take

$$L \geq L_{\text{K-cov}}\Big(x, \frac{\epsilon}{2}\Big) := x + \frac{1}{\lambda} \log \frac{2N}{\epsilon} \,. \tag{B.16}$$

Thus, for any $x < L$, we may replace (B.15) by

$$L \geq \min_x \max \left\{ \frac{1}{2\lambda} \log \frac{2\gamma(x)}{\epsilon}, L_{\text{K-cov}}\left(x, \frac{\epsilon}{2}\right) \right\} , \tag{B.17}$$

where $\gamma(x) = \sum_{m>x} a_m e^{2(N/G)m}$, and obtain a *looser* bound.

B.4 Critical window calculations

Window size if $\ell_{\text{interleaved}} \gg \ell_{\text{triple}}$

We focus here on the bound due to interleaved repeats (rather than triple repeats, treated subsequently), and furthermore assume that the effect of the single largest interleaved repeat is dominant. In this case $\ell_{\text{interleaved}} = L_{\text{crit}} - 1$ is the length of the shorter of the pair of interleaved repeats, and let ℓ_1 be the length of the longer of the two. For $L_{\text{crit}} < L \leq \ell_1 + 1$, we are in the setting of (B.2) but with a redefined $\gamma_1 = e^{2(N/G)(L_{\text{crit}}-1)}$. Thus,

$$L \geq L_{\text{crit}} + \frac{G}{2N} \log \epsilon^{-1} , \tag{B.18}$$

and solving for N gives

$$N_{\text{repeat}} = \frac{G}{2} \frac{\log \epsilon^{-1}}{L - \ell_2 - 1} \tag{B.19}$$

Let L^* be the value of L at which the curve described by constraint (B.19) intersects the Lander-Waterman coverage value, i.e. $N_{\text{repeat}}(L^*) = N_{\text{LW}}(L^*) := N^*$. This is the minimum read length for which coverage of the sequence suffices for reconstruction.

We now solve for $\frac{L^*}{L_{\text{crit}}}$. First, the Lander-Waterman equation (4.2) at $N = N^*$ is

$$N^* = \frac{G}{L^*} \log \frac{N^*}{\epsilon} , \tag{B.20}$$

and setting equal the right-hand sides of (B.20) and (B.19) at $L = L^*$ gives

$$\frac{G}{L^*} \log \frac{N^*}{\epsilon} = \frac{G}{2} \frac{\log \epsilon^{-1}}{L^* - \ell_2 - 1} .$$

A bit of algebra yields

$$\frac{L^*}{L_{\text{crit}}} = \frac{2}{2 - x} , \tag{B.21}$$

where

$$x := \cdot\frac{\log \epsilon^{-1}}{\log N^* + \log \epsilon^{-1}}\,. \tag{B.22}$$

Since $x \leq \frac{1}{2}$, equation (B.21) implies $L^* \leq 2L_{\text{crit}}$, and combined with the obvious inequality $L^* \geq L_{\text{crit}}$, we have $L_{\text{crit}} \leq L^* \leq 2L_{\text{crit}}$. Thus

$$N_{\text{LW}}(2L_{\text{crit}}) \leq N^* \leq N_{\text{LW}}(L_{\text{crit}})\,, \tag{B.23}$$

and applying the Lander-Waterman fixed-point equation (4.2) yet again gives

$$\frac{G}{2L_{\text{crit}}}\log\frac{N_{\text{LW}}(2L_{\text{crit}})}{\epsilon} \leq N^* \leq \frac{G}{L_{\text{crit}}}\log\frac{N_{\text{LW}}(L_{\text{crit}})}{\epsilon}\,. \tag{B.24}$$

Writing this out gives

$$\begin{aligned}&\frac{\log \epsilon^{-1}}{\log\frac{G}{L_{\text{crit}}} + \log\log\frac{N_{\text{LW}}(L_{\text{crit}})}{\epsilon} + \log\epsilon^{-1}} \leq x\\ &\quad\leq \frac{\log \epsilon^{-1}}{\log\frac{G}{L_{\text{crit}}} - 1 + \log\log\frac{N_{\text{LW}}(2L_{\text{crit}})}{\epsilon} + \log\epsilon^{-1}}\,,\end{aligned}$$

and this can be relaxed to

$$\begin{aligned}&\frac{\log \epsilon^{-1}}{\log\frac{G}{L_{\text{crit}}} + \log\epsilon^{-1} + \log\log\frac{G}{\epsilon L_{\text{crit}}}} \leq x\\ &\quad\leq \frac{\log \epsilon^{-1}}{\log\frac{G}{L_{\text{crit}}} - 1 + \log\epsilon^{-1}}\,.\end{aligned} \tag{B.25}$$

Letting

$$r := \frac{\log\frac{G}{L_{\text{crit}}}}{\log\epsilon^{-1}}\,, \tag{B.26}$$

we have to a very good approximation

$$\frac{L^*}{L_{\text{crit}}} \approx \frac{2(r+1)}{2(r+1)-1}\,. \tag{B.27}$$

For $G \sim 10^8$, $L_{\text{crit}} \sim 1000$, and $\epsilon = 5\%$, we get $\log\frac{G}{L_{\text{crit}}} \approx 13.8$ and $\log\epsilon^{-1} \approx 3.0$, so $r \approx 4.6$ and

$$\frac{L^*}{L_{\text{crit}}} = \frac{2(r+1)}{2(r+1)-1} \approx 1.1\,.$$

From (B.26) we see that the relative size of $\log\epsilon^{-1}$ and $\log\frac{G}{L_{\text{crit}}}$ determines the size of the critical window. If in the previous example $\epsilon = 10^{-5}$, say, then $\frac{L^*}{L_{\text{crit}}}$ increases to 1.3. As ϵ tends to zero, r approaches zero as well and $\frac{L^*}{L_{\text{crit}}} \to 2$.

Window size if $\ell_{\text{triple}} \gg \ell_{\text{interleaved}}$

We now suppose the single longest triple repeat dominates the lower bound and estimate the size of the critical window. In this case $\ell_{\text{triple}} = L_{\text{crit}} - 1$ is the length of the longest triple repeat. Since we don't have matching lower and upper bounds for triple repeats, we separately compute the critical window size for each.

We start with the lower bound. For $L > L_{\text{crit}}$, the minimum value of N required in order to bridge the longest triple repeat is given by (B.11) and repeated here:

$$N_{\text{triples}} = \frac{G}{3} \cdot \frac{\log \epsilon^{-1}}{L - L_{\text{crit}}}\,. \tag{B.28}$$

As for the interleaved repeats case considered earlier, we let L^* be the value of L at which the curve described by constraint (B.28) intersects the Lander-Waterman coverage value, i.e. $N_{\text{triple}}(L^*) = N_{\text{LW}}(L^*) := N^*$. This is the minimum read length for which coverage of the sequence suffices for reconstruction.

A similar procedure as leading to (B.21) gives $L^*/L_{\text{crit}} = 3/(3-x)$ with x defined in (B.22). One can check that the estimates on x in (B.25) continue to hold, and we therefore get

$$\frac{L^*}{L_{\text{crit}}} \approx \frac{3(r+1)}{3(r+1)-1}\,. \tag{B.29}$$

For the same example as before, $G \sim 10^8$, $L_{\text{crit}} \sim 1000$, and $\epsilon = 5\%$, we get $r \approx 4.6$ and

$$\frac{L^*}{L_{\text{crit}}} = \frac{3(r+1)}{3(r+1)-1} \approx 1.06\,.$$

Changing ϵ to 10^{-5} makes $\frac{L^*}{L_{\text{crit}}} \approx 1.17$, and as ϵ (and hence also r) tends to zero, $\frac{L^*}{L_{\text{crit}}} \to \frac{3}{2}$.

The analogous computation for L^*/L_{crit} for the upper bound, as given by N_3^{all} in (B.11), yields

$$\frac{L^*}{L_{\text{crit}}} = \frac{r+1}{r + \frac{\log 3}{\log \epsilon^{-1}}} \approx 1.12\,, \tag{B.30}$$

for the example with $G \sim 10^8$, $L_{\text{crit}} \sim 1000$, and $\epsilon = 5\%$. The critical window size of the upper bound is about twice as large as that of the lower bound for typical values of G and L_{crit}, with ϵ moderate. But as $\epsilon \to 0$, we see from (B.30) that $L^*/L_{\text{crit}} \to \infty$, markedly different to the $L^*/L_{\text{crit}} \to \frac{3}{2}$ observed for the lower bound.