Telomeres are particularly complex and repetitive, and thus very have been linked to ageing (McEachern characterized, though a great deal is to be gained from it, as telomeres to assemble telomeric regions from short-read data. Due to this are very similar. Existing assembly algorithms thus frequently fail repeats within itself, but often telomeres on different chromosomes regions, which are one of the primary challenges in assembly. for better algorithmic use of the data, especially for repetitive permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Recent advances in high-throughput sequencing (HTS) technologies have dramatically reduced the cost of producing reference genomes for new species and characterizing whole-genome variations in multiple individuals of a population. However, the assemblies produced by current algorithms are often incomplete. Alkan et al. report that a de novo shotgun assembly of the human genome using short-reads is 16% shorter than the reference assembled using more laborious methods, and that < 1% of segmental duplications are represented. Indeed, it is well recognized that there is room for better algorithmic use of the data, especially for repetitive regions, which are one of the primary challenges in assembly. 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 ageing. 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.

Genome assembly is the challenge of piecing together reads to reconstruct the original genome. Reads are obtained from various technologies, with varying read length, error rates and coverage. Sanger-chemistry reads range in length from around 500 to 1000 bases. Newer technologies such as Illumina Complete Genomics (Fermin et al., 2010), Helicos (Harris et al., 2009), 454 Life Sciences (Margulies et al., 2005), SOLiD (McKeeman et al., 2007) and Ion Torrent (Irving et al., 2010) provide reads at vastly lower costs for greater throughput, but at the expense of length. Initial improvements in assembly from short-read data focused on how to process the sheer quantity of data and how to detect overlaps. 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., 2009), and subsequent assemblers including SOAPdenovo Li et al. (2010), ALLPATHS 2 (MacCallum et al., 2008), ABySS (Simpson et al., 2009), and Cortex (Rabbi et al., 2011).

Many HTS platforms produce paired-end or mate-pair reads, which we collectively refer to as read-pairs. The paired nature of these reads constitutes a powerful source of information, significantly facilitating genome assembly. Improved use of read-pair information lies at the heart of recent works such as ALLPATHS-LG (Gnerre et al., 2011), the PE-Assembler (Vayargine and Simons, 2011), and the Paired de Bruijn Graph (Medvedev et al., 2010), innovations of which are primarily in earlier use of short-insert read-pairs.

ALLPATHS-LG requires reads of length around 100 bases sequenced from short fragments of length ≈180 bp so that, on average, each read-pair overlaps by ~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 300–500bp in size. The PE-Assembler builds short stretches in non-repetitive regions first, similar to unitigs (see Section 4 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 Parish 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 long enough long-insert libraries can substantially improve assembly.
Recent innovations (Yeong et al. 2013) in library construction may bring such libraries into the mainstream, so it is timely to develop algorithms that take full advantage of such data.

In this article, 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, our algorithm was developed with the specific aim of assembling highly repetitive regions such as telomeres. 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 3.1 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 3.2.

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 3.3 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 chromosomes have a core repetitive component called X (=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 contiguous assemblies in the telomeric regions than other state-of-the-art de novo assembly algorithms, especially when longer long-insert libraries are used.

**2 TERMINOLOGY**

We adopt the following terms commonly used to describe the output of sequencing technologies and the resulting assemblies:

- **Read-pair:** refers to 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.

- **Mate:** refers to 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.

- **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.

**3 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 larger 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 Brujin 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 1. The algorithm proceeds by extending the end of the contig iteratively by a fixed amount, $N_{max}$ per iteration, as detailed in Section 3.1. 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 $N_{max}$ 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 3.4. 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.
The extension step consists of finding possible extensions of a given e-node; we are extending to the right. The reads in the read cloud are error-corrected, then used to construct a unitig graph. Each new e-node consists of new bases from the end of the previous e-node.

The reads in the read cloud are error-corrected, then used to construct a unitig graph. Each new e-node consists of new bases from the end of the previous e-node.

3.1 Iterative extension of assembly

The extension step is depicted in Figure 2. It begins by mapping all the reads in the read cloud to a unitig graph. Each new e-node consists of new bases from the end of the previous e-node.

In the following description, we assume without loss of generality that the reads are sorted in decreasing order of length, and that the insert length is a constant.

Let \( U \) be a unitig, and let \( S \) be its set of mates. The expected number of reads at position \( x \) behind unitig \( U \) is given by

\[
\exp(x) = h(x + 1),
\]

where \( h(x) \) is the expected insert size at position \( x \).

The expected number of left-reads in \( U \) is

\[
f_L(x) = \exp(x) - 1.
\]

We compute \( f_L(x) \) by summing the expected insert distribution over all gaps in the unitig graph.

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.

In our implementation, we choose this length, denoted \( S_0 \), 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 the standard deviation of the short-insert library. The set of left-mates associated with reads in unitig \( U \) is denoted \( M_L(U) \).

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. (c) Paths through the unitig graph correspond to possible extensions.

In the following description, we assume without loss of generality that the reads are sorted in decreasing order of length, and that the insert length is a constant.

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

Although 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 3. 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 \( M_L(U) \). (Fig. 3).

The expected number of reads at position \( x \) behind unitig \( U \) is given by

\[
f_L(x) = \exp(x) - 1.
\]

where \( h(x) \) is the expected insert size at position \( x \).

We compute \( f_L(x) \) by summing the expected insert distribution over all gaps in the unitig graph. The expected number of left-reads in \( U \) is

\[
f_L(x) = \exp(x) - 1.
\]

where \( h(x) \) is the expected insert size at position \( x \).

We compute \( f_L(x) \) by summing the expected insert distribution over all gaps in the unitig graph. The expected number of left-reads in \( U \) is

\[
f_L(x) = \exp(x) - 1.
\]

where \( h(x) \) is the expected insert size at position \( x \).

We compute \( f_L(x) \) by summing the expected insert distribution over all gaps in the unitig graph. The expected number of left-reads in \( U \) is

\[
f_L(x) = \exp(x) - 1.
\]

where \( h(x) \) is the expected insert size at position \( x \).

We compute \( f_L(x) \) by summing the expected insert distribution over all gaps in the unitig graph. The expected number of left-reads in \( U \) is

\[
f_L(x) = \exp(x) - 1.
\]

where \( h(x) \) is the expected insert size at position \( x \).

We compute \( f_L(x) \) by summing the expected insert distribution over all gaps in the unitig graph. The expected number of left-reads in \( U \) is

\[
f_L(x) = \exp(x) - 1.
\]

where \( h(x) \) is the expected insert size at position \( x \).

We compute \( f_L(x) \) by summing the expected insert distribution over all gaps in the unitig graph. The expected number of left-reads in \( U \) is

\[
f_L(x) = \exp(x) - 1.
\]

where \( h(x) \) is the expected insert size at position \( x \).

We compute \( f_L(x) \) by summing the expected insert distribution over all gaps in the unitig graph. The expected number of left-reads in \( U \) is

\[
f_L(x) = \exp(x) - 1.
\]

where \( h(x) \) is the expected insert size at position \( x \).

We compute \( f_L(x) \) by summing the expected insert distribution over all gaps in the unitig graph. The expected number of left-reads in \( U \) is

\[
f_L(x) = \exp(x) - 1.
\]
Telescoper uses all libraries simultaneously during assembly, rather than reads. This support measure is then multiplied by the short-insert score a linear function of the fraction of the extension that is covered by the right gather all read-pairs of which right reads map to the extension and left-reads libraries simultaneously. For there to be ambiguity in extension choice, there is the proposed extension supported by read-pair information from all other produced and pruned a set of extensions using just the short-insert library in other assembly algorithms, including ABySS (Simpson et al., 2009), ALLPATHS 2 (MacCallum et al., 2010), SGA (Simpson and Durbin, 2011), 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 favourably with other algorithms, including Celera (Myers et al., 2002), which was designed for Sanger reads.

4.1 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](http://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 100× and an insert distribution with mean 400 bp and variance 75 bp. The long-insert reads had coverage depth 20× 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 datasets 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. (2013), were sorted using DeLover into reads categorized as short-insert (0–400 bp fragments, mean 220 bases) or long-insert (1–5 kb, mean 2.3 kb). The reads varied in length from 30–100 bp. We truncated reads to 50 bases in order to provide algorithms with high-quality, uniform-length reads. We used coverage 120× for the short-insert data and 40× 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 dataset, 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.

4.2 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
Table 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.

<table>
<thead>
<tr>
<th>Assembler</th>
<th>Produced (kb)</th>
<th>Aligned (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N50</td>
<td>NG50</td>
</tr>
<tr>
<td>Telescoper</td>
<td>40.0</td>
<td>40.0</td>
</tr>
<tr>
<td>ABySS</td>
<td>31.0</td>
<td>31.0</td>
</tr>
<tr>
<td>ALLPATHS2</td>
<td>35.2</td>
<td>33.0</td>
</tr>
<tr>
<td>SOAPdenovo</td>
<td>25.0</td>
<td>24.0</td>
</tr>
<tr>
<td>Velvet</td>
<td>13.9</td>
<td>9.0</td>
</tr>
<tr>
<td>SGA</td>
<td>31.2</td>
<td>27.0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Assembler</th>
<th>Produced (kb)</th>
<th>Aligned (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N50</td>
<td>NG50</td>
</tr>
<tr>
<td>Telescoper</td>
<td>39.0</td>
<td>38.0</td>
</tr>
<tr>
<td>ABySS</td>
<td>12.1</td>
<td>8.0</td>
</tr>
<tr>
<td>ALLPATHS2</td>
<td>32.0</td>
<td>27.0</td>
</tr>
<tr>
<td>SOAPdenovo</td>
<td>25.0</td>
<td>21.0</td>
</tr>
<tr>
<td>Velvet</td>
<td>14.0</td>
<td>9.0</td>
</tr>
</tbody>
</table>

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 datasets. Furthermore, Figures 5a and 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 5b and that in Figure 5c 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 5c (see Supplementary Material), suggesting that Telescoper is robust to the complications of real data and that the observed good performance is not an artifact of the specific insert-size distribution used.

Table 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.

<table>
<thead>
<tr>
<th>Assembler</th>
<th>Aligned (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N50</td>
</tr>
<tr>
<td>Telescoper</td>
<td>34.5</td>
</tr>
<tr>
<td>ABySS</td>
<td>12.0</td>
</tr>
<tr>
<td>ALLPATHS2</td>
<td>26.3</td>
</tr>
<tr>
<td>SOAPdenovo</td>
<td>21.4</td>
</tr>
<tr>
<td>Velvet</td>
<td>11.8</td>
</tr>
</tbody>
</table>

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 et al., 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 1 while the results for the real data are shown in Table 2. These results are for the 32 telomeric regions, each 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 1 reducing the read length from 101 to 50 bp while keeping all other parameters the same worsened the performance of most algorithms, with ABySS being the most affected.

Figure 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.
of Telescoper in Figure 5b 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 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 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 colour, so more colours 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.

5 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.

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 datasets 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 Section 1. 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 article 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 will 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.

Fig. 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.
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.

ACKNOWLEDGMENTS

We thank Jasper Rine, Oliver Zill and Devin Scannell for their motivation of the problem and for useful discussions. We also thank Paul Jenkins and Jeremy Ma tin-Shepard for helpful discussions.

Funding: In part by NSF Graduate Research Fellowships (to M.B. and S.S.), an NDSEG Graduate Fellowship (to A.H.C.), and an NSF CAREER Grant DBI-0846015 (to Y.S.S.).

Conflict of Interest: none declared.

REFERENCES


Parrish,N. et al. (2011) Assembly of non-unique insert content using next-generation sequencing. BMC Bioinformatics, 12 (Suppl. 6), S3.


