Project Report

Fundamentals of Bioinformatics, 02-604,

Long-Read Genome Assembly

"A Case in Study in Python"

Group 13 (D), SMRTy Pants

Dylan Estep, Ethan Gaskin,
Akshat Gupta, Zhen Yang

Background

Deoxyribonucleic acid (DNA) is the set of genetic instructions guiding an organism's development, function, and environmental response (National Institutes of Health, 2015). The genome of an organism is the collection of all of the organism's DNA. Since Watson and Crick's discovery of DNA's structure and the primary genetic code (Watson & Crick, 1953), genome sequencing has become increasingly essential to biology. Determining the sequence of the genome is a difficult task, and is divided into two categories: *de novo* genome assembly and reference-based genome assembly.

De novo assembly describes a computational task where subsequences from a genome are assembled into longer contiguous sequences without the use of a reference genome (Khan et al., 2018). Subsequences from a genome are called reads and are generated via sequencing. Reads are generated as follows (Sung, 2017): (1) sample tissue/cells are collected from an organism, (2) DNA is isolated from these cells, (3) DNA is fragmented (via sonication or digestion), (4) fragments within a specific size range are selected, (5) read sequencing by amplification detection. As sequencing technologies have become cheaper and more advanced in recent decades, the need for computer algorithms and tools to utilize these sequences and generate high-quality assemblies has become more critical than ever (Baker, 2012).

Assemblers commonly employ two types of algorithms: greedy algorithms which target local optima (Greedy-SCS), and graph-based algorithms, such as De Bruijn graphs or overlap graphs to aim for global optima (Khan et al., 2018; Ben Langmead, Overlap Layout Consensus assembly). The commonly used overlap-layout-consensus (OLC) paradigm utilizes overlap graphs.

Additionally, different assemblers are typically tailored to different read technologies. Illumina-generated reads (short-read sequencing) are usually shorter (around 50 to 200 base pairs) with lower error rates (0.5-2%) compared to reads from PacBio or Oxford Nanopore (long-read sequencing), which generate longer reads with a much higher error rate, typically around 10 to 20% (Zhang et al., 2020). Long read sequencing methods are also known as single-molecule real-time sequencing technologies (SMRT). OLC based assembly methods tend to perform better for long reads than De Bruijn graphs, due to their ability to handle error prone long reads (Khan et al., 2018). Short read sequencing has two varieties: single read and paired-end reads, but long reads are single read (Freedman et al., 2020). Single read means sequencing the molecule from one end (5' to 3'), whereas paired-end means sequencing from both ends of a molecule.

Since there are repetitive regions throughout the genome with identical or nearly identical sequences occurring multiple times, short reads typically ranging from 50 to 200 base pairs in length may not be sufficient to uniquely identify and characterize these repetitive regions. Resolving highly repetitive regions in genomes with short reads presents a challenge. On the other hand, long reads have an advantage of resolving repetitive regions but with the caveat of high sequencing error rate. As methods advance, the error rate of long read methods is decreasing, but the cost of the technology is much greater than short read. Paired-end reads are highly valuable for dealing with repetitive regions and the scaffolding of contigs (Sung 2017). All this together makes hybrid approaches with single-end long reads and paired-end short reads ideal.

These differences make it necessary to have different assemblers capable of resolving repeats for short reads or managing the high error rates associated with long reads. As sequencing technologies advance, the potential to generate long reads with low error rates becomes the next frontier in assembling genomes with greater confidence.

Computational Problem

De Novo Genome Reconstruction Problem: Reconstruct a genome from error-prone, single-molecule DNA sequencing reads (long reads) without a reference genome

- Input: fastq file consisting of
    - N single-molecule sequencing reads
    - quality scores of base calls for each read
- Output: Long contiguous sequences (contigs) that are "highly likely" to be in the genome used to produce these reads

"Highly likely" is reflected by the following metrics (Sung 2017): total number of contigs, maximum contig length, combined contig length, N50 score, and N90 score. Combined contig length is the total of all contig lengths. N50 is the maximum contig length such that contigs of length equal to or longer than said length account for 50% of the combined contig length. N90 is similar but for 90%. Good assemblies will minimize the number of contigs while ideally making the maximum contig length, combined contig length, N50, and N90 as close as possible to the true genome size. The true genome size can be estimated without knowing the sequence in advance via various wet lab techniques (e.g. gel electrophoresis). For genomes with a reference assembly, we can compare the assembled contigs directly against the reference. *De novo* assembly on genomes with an existing reference can be valuable to detect novel mutations and repeats that are not represented by the reference.

Overlap Layout Consensus for Long Read Assembly

Given the increasing usefulness of long reads, we would like to perform a case study in long read assembly. As noted above, the OLC paradigm is commonly used for long read based genome assembly. The steps of OLC based assembly can be summarized as follows (Sung 2017):

OLC

1. Overlap: Detection of which reads overlap well under an overlap alignment scoring scheme. Ideally, every read will have an oriented pairing with one or more other reads that indicates a suffix of read i is a prefix of read j. This is the computational bottleneck of OLC (Chu et al., 2017).
2. Layout: Using the pairwise overlap detection, bundle reads into groups (layouts) so that reads coming from similar genomic regions are together.
3. Consensus: Overlay each read in the layouts via multiple sequence alignment (MSA) to produce a consensus for each layout. These are the contigs.

Finishing

4. Scaffolding: Align reads over the contigs to decide orientation and contig order (generally uses paired end short reads).
5. Gap filling: Align reads over gaps to try to fill in spaces between contigs.

Methods

Salmonella Dataset

We obtained data from a whole-genome long-read sequencing experiment from the bacterium *Salmonella enterica* subsp. *enterica* serotype Hadar, archived at the National Library of Medicine's Sequence Read Archive under run ID SRR27959541. The raw data consists of 106,084 reads generated by an Oxford Nanopore sequencing platform. This dataset does not contain paired-data for the last two steps of assembly, so normal scaffolding algorithms will not apply.

Quality Control

Long reads obtained from sequencing technologies may contain reads with undesirable properties, such as low base quality (Phred) scores and low GC content, or the read itself may be an outlier in terms of length. Hence, before beginning the assembly process, it is essential to preprocess the data to eliminate low-quality reads. In our project's Quality Control (QC) stage, we employed NanoFilt as a crucial tool to ensure the integrity and reliability of our long-read assembly process. NanoFilt is specialized software tailored for preprocessing Oxford Nanopore sequencing data (De Coster et al., 2018). It has an array of parameters that allow the user to trim unwanted reads from the dataset. Specifically, we used three parameters within NanoFilt to filter our reads. The first is the average read quality. We set this value to be 10. This indicates that we discard any reads with an average read quality score of less than 10, with this score indicating a base call accuracy of 90%. We also chose this value since we observed that any value above 10 was too stringent and led to dropout of a significant number of reads. The second parameter was the length of reads. Upon inspection of the length of the distribution of all reads, we set 200 as the minimum length of reads, with any read shorter than this threshold excluded. The final parameter was the GC content. We set this threshold to be 30%. This threshold was set empirically after examining the general GC content in various bacteria. These parameters eliminated low-quality reads, including outliers. We initially had approximately 106,000 reads in the data, which was reduced to approximately 101,000 reads post-filtering. A comparative length distribution before and after filtering can be seen in Figure 1.

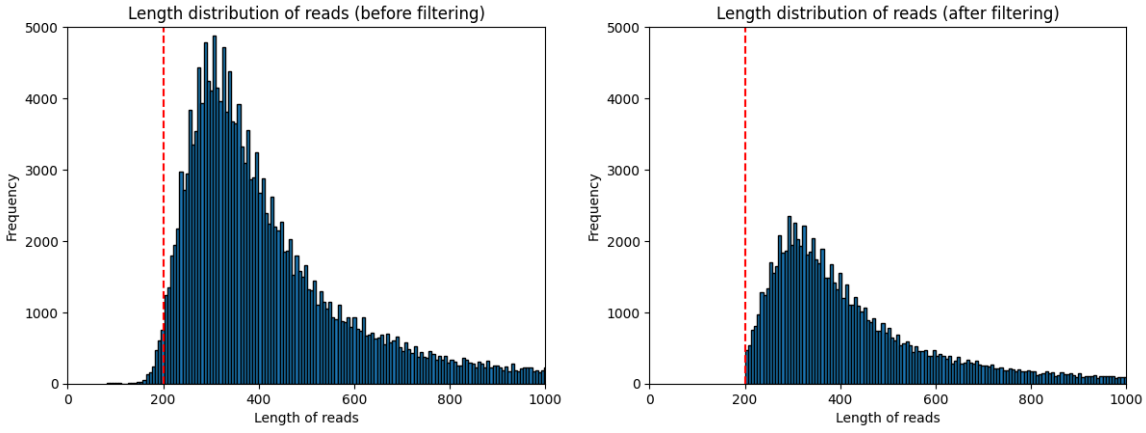

Figure 1: Length distribution before and after filtering and preprocessing using Nanofilt.

MHAP-based Overlap

The MinHash Alignment Process (MHAP) is an algorithm developed by Berlin et al. in 2015. It aims to overlap noisy, long reads by employing probabilistic, locality-sensitive hashing (LSH) techniques. Initially, for any given pair of reads S_1 and S_2 in the fastq file, each read is broken into its constituent k-mers. These k-mers are then hashed into integers, known as fingerprints $\Gamma(S_i)$, using H independent, identically distributed hash functions. In MHAP, the first hash (Γ_1) is generated using the MurmurHash3 hash function implemented in the Guava library, while subsequent fingerprints ($\Gamma_{2...H}$) are produced using an XORShift pseudo-random number generator. The minimum fingerprints from each hash function, referred to as minmers, are then stored in a list called a sketch, with a size of H corresponding to the number of hash functions used. The probability of $\Gamma(S_1)$ and $\Gamma(S_2)$ sharing the same min-mer is equivalent to the likelihood of a k-mer existing in at least one of the strings and in both strings. This probability is represented by the Jaccard similarity $J(S_1, S_2)$ (Broder, 1997), where $P[min\Gamma(S_1) = min\Gamma(S_2)] = J(S_1, S_2) = \frac{|\Gamma(S_1) \cap \Gamma(S_2)|}{|\Gamma(S_1) \cup \Gamma(S_2)|}$. The higher the Jaccard index, the more similar S_1 and S_2 are to each other.

The MinHash approach, as opposed to direct computation of the Jaccard similarity, is effective because it condenses reads from K integer fingerprints to smaller, randomized vectors of H fingerprints known as sketches, where H << K (Berlin et al., 2015). When calculating the Jaccard index, considering only H values for each read reduces read lookup costs in proportion to the size of the sketch, making it advantageous to keep H small for lower computational costs. However, Berlin et al., (2015) observed that sensitivity can be enhanced by increasing the sketch size, thereby reducing the expected error of the Jaccard estimate. Consequently, selecting the optimal H value is crucial due to its influence on accuracy and speed. From Berlin et al's simulations, H=1256 worked well in practice with k=16 for assembling the human genome.

Berlin et al. conducted multiple simulations for various combinations of H and k, involving unrelated sequences (Rand), reads overlapping by exactly 2 kbp (Olap), and reads mapped to a perfect reference (Map), see Figure 2. They found that when k = 10, it is common to encounter at least three min-mer matches by chance with a high probability of matching. However, when k = 16, at least 3 min-mer matches are sufficient to distinguish random matches from true matches, especially with larger values of H (H > 1000). Based on this evidence, we chose to set our H=1256. Berlin et al.'s MHAP used H=1256 to achieve a sensitivity of 0.89. We also selected k = 16 for our purposes. We verified from our own simulations that the estimate of the Jaccard index vs the true Jaccard index for these parameters was highly correlated, see Figure 3.

Additionally, by setting k = 16, MHAP can accurately detect 2 kbp overlaps from 10 kbp reads simulated from the human genome, even with an overlap error rate of 30%. However, considering that we are assembling a bacterial genome, which is significantly smaller than the human genome, the optimal k for our case may be smaller than 16.

To summarize, MHAP is an overlap detection algorithm that uses the MinHash approach which allows for quick numeric summaries of reads, which are highly correlated with underlying sequence identity and can

be quickly compared to determine whether two reads are good candidates for overlap alignment. The sensitivity of the algorithm is tunable by the size of k, the sketch size H, and the Jaccard index estimate threshold (Berlin et al., 2015).

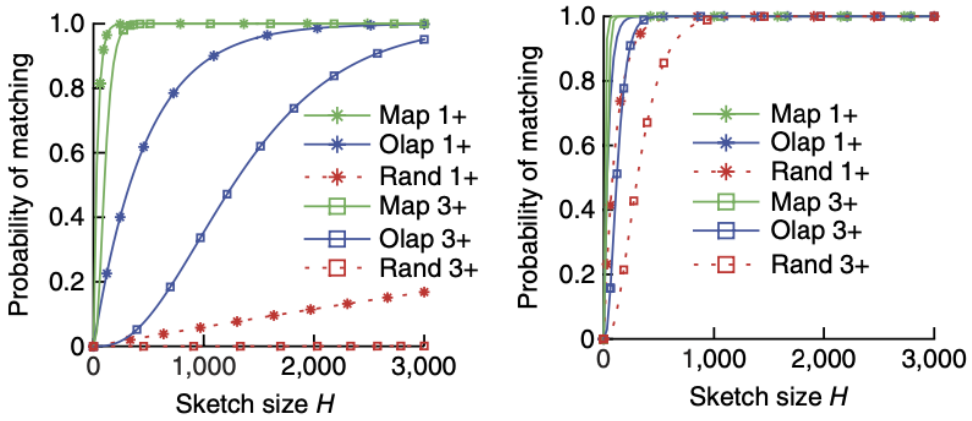

Figure 2. Probabilities of matching by sketch size for k = 10 (left) and k = 16 (right). From Berlin et al., 2015.

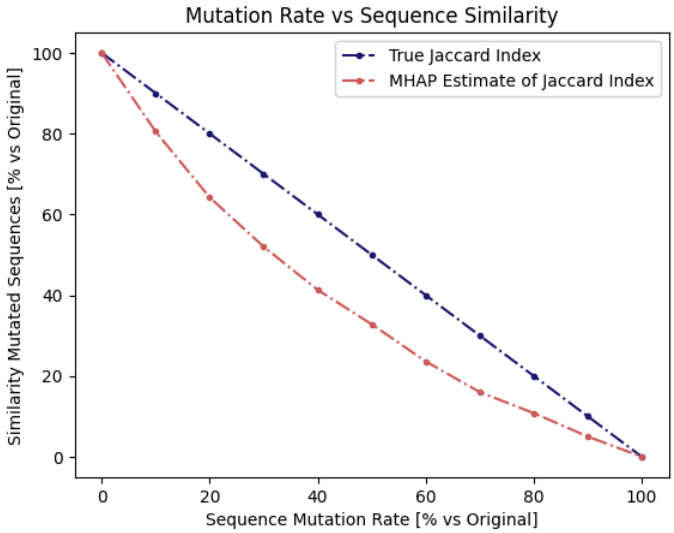

Figure 3. Comparison of MHAP-estimated Jaccard index with true Jaccard index for sequences of varying rates of difference/mutation.

MHAP Parallelization

Assume that MHAP with sketch size H tells us (on average) each read has Z candidate reads to overlap with. Let $N=1\times10^5$, H=1256, Z=1000. This will mean there are $O(NH) = 1.256\times10^8$ hashes to compute (for all N sketches), $O(N^2H^2) \approx 1.577536\times10^{15}/2$ calculations for the Jaccard matrix (N^2 elements, the symmetric nature allows this to be cut in half), $O(NZ) = 1\times10^8$ alignments will be necessary (2% of total $1\times10^5/2$ alignments). Clearly, this is a lot of computation. Although hashing in general is a quick, mathematical process, this is a lot of computation.

LSH has specifically been identified as a fast way to compare two sets of objects with statistical guarantees on the comparison, and has been shown to make intractable analyses into tractable analyses for large datasets (Dasgupta et al., 2011). Our implementation of LSH was somewhat naive since we computed the entirety of each read's sketch at once, which is slower and more memory intensive than (LSH) hashing reads into tables and tracking collisions. However, directly computing the sketches is a simple exercise and is embarrassingly parallel since computing the sketch of each read is independent of every other read. Thus, our implementation used Python's multiprocessing module to compute a table containing all the sketches of the reads at once (1×10^5*H int32). Similarly, pairwise comparison of each sketch is independent of other pairs of sketches (except symmetric comparison of i vs j is the same as j vs i), so we computed the Jaccard index estimates for each read in an embarrassingly parallel manner. This results in a 1×10^5*1×10^5 int32 symmetric matrix. Since this matrix is inconveniently large to store in one file, we stored the matrix row-wise, setting us up for the embarrassingly parallel step of checking candidate overlap pairs via alignments.

Error Correction

The high sequencing error rate from long-read methods necessitates additional effort to correct errors in the sequencing data, which manifest both as substitutions and indels with respect to the true template sequence. In our approach, we chose to implement the FALCON-sense consensus-based method introduced in the FALCON assembler (Chin et al., 2016), variants of which have also been incorporated into MHAP and Canu (Koren et al., 2017). The corrected sequence for each read in this method was obtained by performing fitting pairwise alignments of the read against all significantly overlapping reads, then generating a consensus from the resulting combined alignment. The fitting alignments, which permit penalty-free gaps preceding and following bases of the shorter read in the alignment, were performed using an extreme mismatch penalty to force deletion-insertion pairs where mismatches might otherwise occur. Following the lead of the implementation described by Sung 2017, we encoded each position in the pairwise alignments with tuples (p, d, b), where p represents the numerical position with respect to the template read, d represents the numerical order of a deletion in the template, and b represents either the (matching) symbol at the position or a gap symbol. After obtaining counts of these tuples and ordering them in ascending (p, d) order, consensus for the read was generated by outputting symbols b for those tuples (p, d, b) such that the count exceeds $\frac{1}{2}\sum_{x\in\{A,C,G,T,-\}} count(p, 0, x)$.

Following error-correction, we established directionality of edges in the overlap graph by comparing, for each pair of overlapping reads s and t, scores from the overlap alignments of a suffix of s to a prefix of t and of a suffix of t to a prefix of s. If the score from the former case was higher, the edge $s \rightarrow t$ was formed; otherwise, $t \rightarrow s$ was formed instead. A match reward of 3, mismatch penalty of 1, and gap penalty of 2 were used for these alignments. Fitting alignments used the same values except for the mismatch penalty, which was 1000.

Layout

The overlap graphs obtained from the previous steps tend to be large and messy, with each node having a high in-degree and out-degree. This makes it extremely difficult to identify contigs directly from the

overlap graph. Hence, we pruned the graph in this step while maintaining the inherent relationships between all nodes. In other words, we removed redundant edges in the graph via a method known as transitive reduction. In the mathematical field of graph theory, a transitive reduction of a directed graph D is another directed graph with the same vertices and as few edges as possible, such that for all pairs of vertices v, w a (directed) path from v to w in D, exists if and only if such a path exists in the reduction (Aho et al., 1972). This step allowed us to simplify the graph and then subsequently determine contigs. Figure 4 shows an overlap graph before and after transitive reduction.

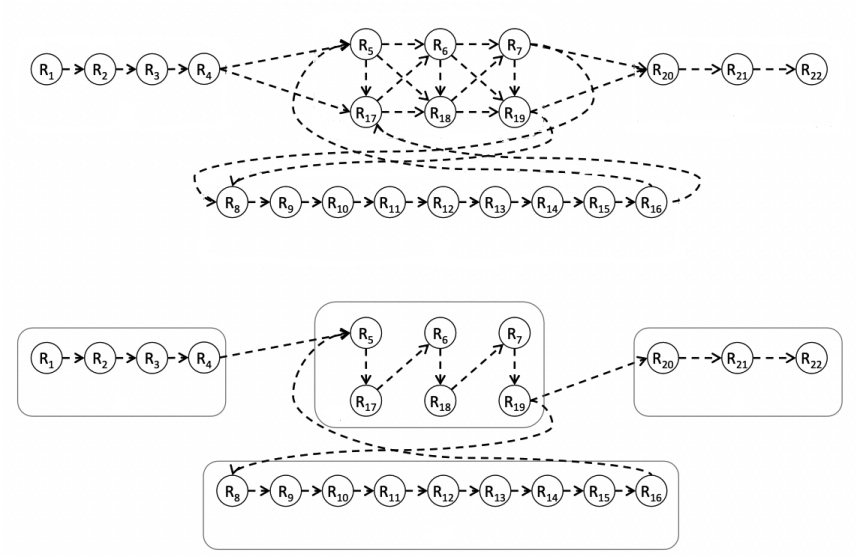

Figure 4: Example of an overlap graph before (above) and after (below) transitive reduction. An example of a redundant edge is (R_4, R_{17}) in the original graph. Since R_{17} can be accessed from R_4 via R_5, (R_4, R_{17}) is removed in the reduced graph. Adapted from Sung 2017.

Once we obtained the reduced graph, we identified contigs as maximal non-branching paths in the graph. Maximal non-branching paths are paths whose internal nodes are 1-in-1-out nodes and initial and final nodes are not 1-in-1-out nodes. In our case, each node represents a read, and these paths represent layouts. Each layout subsequently represents reads coming from one part of the genome. It is also important to note that the reduced graph may contain unresolvable spurious regions. Such paths must not be included in any layout and must be discarded as such. After reviewing examples of such unresolvable regions in the layout graph, we noticed that such repeats always occur in nodes that are part of a cycle in the reduced graph. Hence, we remove nodes and edges that are part of cycles before finding maximal non-branching paths (Figure 5).

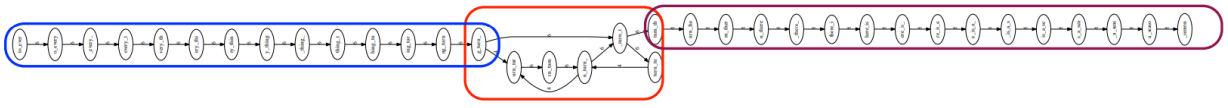

Figure 5: Overlap graph containing two transitively reduced paths corresponding to contigs and an unresolvable repeat cycle (red). From Langmead.

Consensus

To generate sequences for contigs obtained during the Layout step, we exported the reads within a contig in FASTA format, inputting this to Clustal Omega to produce a multiple sequence alignment (Sievers et al., 2011). This package performs a progressive alignment of the input sequences based on a guide tree constructed by the UPGMA algorithm with a distance matrix computed from pairwise *k*-tuple distances. These distances represent the sum, over all possible *k*-tuples (strings of length *k*), of the differences in the number of occurrences in two input sequences. For each position in the resulting multiple sequence alignment output by Clustal Omega, the most commonly occurring symbol among rows for which the position was internal to the corresponding read was output, generating the complete consensus sequence for the contig (see Figure 6).

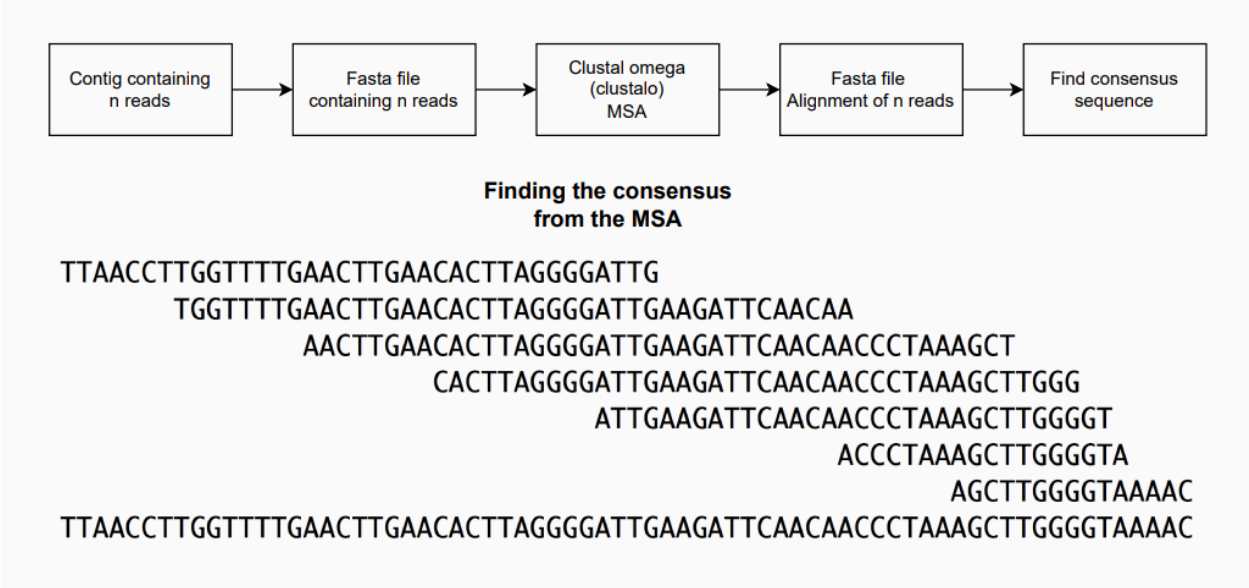

Figure 6. Flowchart of consensus-generation process and illustration of reading out consensus sequence from contig reads. Reads and consensus sequence from Bioinformatics Workbook.

Results

Test Code & Toy Datasets

Toy datasets and test code are essential for verifying that an algorithm behaves correctly. To this end, we attempted to develop test data/code to validate our implementation of the OLC paradigm for noisy long read data.

Computational demands of assembly algorithms are substantial. Performing assembly on data containing large numbers of reads, as in virtually any genuine sequencing experiment, requires an abundance of memory, computational resources, and time. With approximately 100,000 reads, after filtering, serving as

input for the assembly methods, we soon found that time constraints would be prohibitive without substantial efforts devoted to parallelization, as noted above, even utilizing Amazon Web Services resources. Through such efforts, we were able to successfully perform the MHAP phase of the overlapping procedure for the entirety of the genuine dataset, but had not cleared the alignment-heavy remainder of the Overlap bottleneck or further downstream phases of the pipeline. We thus found it necessary to construct "toy" datasets of smaller numbers of reads (on the order of thousands rather than hundreds of thousands) derived from short ground-truth genome sequences to observe the performance of the entire pipeline.

Given the need to generate test datasets, we created a toy genome that was 5000 nucleotides long and then generated long reads using this string. We generated three sets of long reads using different techniques, and our goal was to reassemble them using our algorithm and compare them to the original genome. The techniques we used to create long reads from a genome are as follows:

Toy Dataset 1: error-prone reads

We iterated through each position *i* in the genome and then generated a random number *k* between 250 and 750. Each read was a subset of the genome spanning indices *i* through *i+k.* We continued to generate reads until we surpassed the length of the genome. We employed this technique to generate overlapping reads and assess whether our algorithm could generate reliable contiguous sequences (contigs). To account for the error rate in long-read sequencing, we also induced a randomly applied 10% base substitution rate in each read and began the OLC assembly pipeline. This method generated 4275 reads.

Toy Dataset 2: error-free reads

We generated this dataset in the same manner as the one mentioned above. The only difference was that we did not introduce errors into any read after generation. The goal was to see if we could obtain reasonable contigs using reads that were 100% accurate. This method also generated 4275 reads.

Toy Dataset 3: error-free, overlaps consistent and emphasized

We generated a dataset from the same toy genome used for Datasets 1 and 2. The generation procedure *did not* introduce errors into the sequences, overlaps were created in a consistent and reproducible manner, and lastly the read depth was made large enough to emphasize the overlaps. Figure 7 describes the algorithm used to generate the dataset. Since the genome is 5000, this implies lenReadUni is 500 and lenReadOverlap is 100. We had 10 unique reads, and used a readDepth of 5 to emphasize the overlaps by the reads resulting in 50 total reads. Every read was length 600 except the 5 (duplicate) reads coming from genome indices [0,500).

Algorithm: GenerateDataset3
Input: *genome*, *readDepth*

```
n ← len(genome)
lenReadUni ← n × 0.10 # length of unique genome indices in read
lenOverlap ← n × 0.01 # length of overlap
reads ← empty list
for i ← 0 to n increment lenReadUni do
    if i == 0 then
        read = genome[i : i + lenReadUni]
    else
        read = genome[i − lenOverlap : i + lenReadUni]
    reads.append(read)
reads ← list containing every read in reads, duplicated readDepth times
```

Figure 7: Pseudo-code for generating dataset 3.

Toy Dataset 1: Results

We obtained ~2,700 contigs. The majority had length 2, with few contigs of length 3. This meant that the assembly had failed and that performing the consensus step for this set was futile. We hypothesized that this may have been due to the errors we induced in the reads, so we decided to test our assembler on perfect reads to ascertain if that was the case.

Toy Dataset 2: Results

We obtained ~6,700 contigs. In this case, we observed more contigs than reads, and the length distribution was similar to what we saw for Dataset 1. While we did observe some contigs of length 4 and 5, the distribution was still skewed in favor of length 2 and 3 contigs.

Toy Dataset 3: Results

We obtained 9 contigs from this dataset. The number of reads in each contig were as follows: [11, 5, 5, 2, 7, 5, 5, 11, 7]. And the length of each consensus sequence derived for each contig following MSA with *Clustal Omega* was: [680, 600, 600, 635, 681, 600, 600, 687, 668]. The overlap graph before and after transitive reduction are given in Figure 8, with the contigs circled in the latter. We noticed that read 19 was linked to read 10, the reverse of the expected edge direction based on how the data were generated. To investigate further why this was the case, we checked the overlap alignment scores for these two reads. We found that aligning the suffix of read 10 to the prefix of read 19 gave a score of 607 whereas aligning the suffix of read 19 to the prefix of read 10 gave a score of 610. It was curious to see that both these scores were very similar in alignments, even though generally aligning the prefix of some read i against the suffix of read j should not result in the same score as aligning the suffix of read i with the prefix of read j. However, since the score of aligning the suffix of read 19 to the prefix of read 10 was higher than its counterpart, the link in the graph was justified by OLC. We also noticed a similar behavior for other read pairs (e.g. reads 19/20) wherein the overlap alignment scores were similar in both directions. This may be attributed to the randomness of the current dataset and we hypothesize that this may also be the reason why contigs obtained after layout frequently end up shorter than anticipated. A more elaborate

statistical approach to determine the edge orientation in the overlap graph will be critical to improving the assembly pipeline. Additionally, the maximal non-branching paths break at node 19 (since 19 to 20 and 19 to 10) was justified by OLC since overlap alignment 10 with 20 (or 20 with 10) resulted in a highly similar score (597 and 608 respectively) compared with overlapping 19 to 20 and 19 to 10 (599 and 610). This is unexpected since 10 and 20 explicitly do not overlap at all: read 10 comes from *genome*[900,1500) whereas read 20 comes from *genome*[1900,2500). Given this unexpected similarity due to randomness, resolving this break is difficult and suggests we should use paired end reads in a scaffolding step to ensure that the contigs are in the proper order.

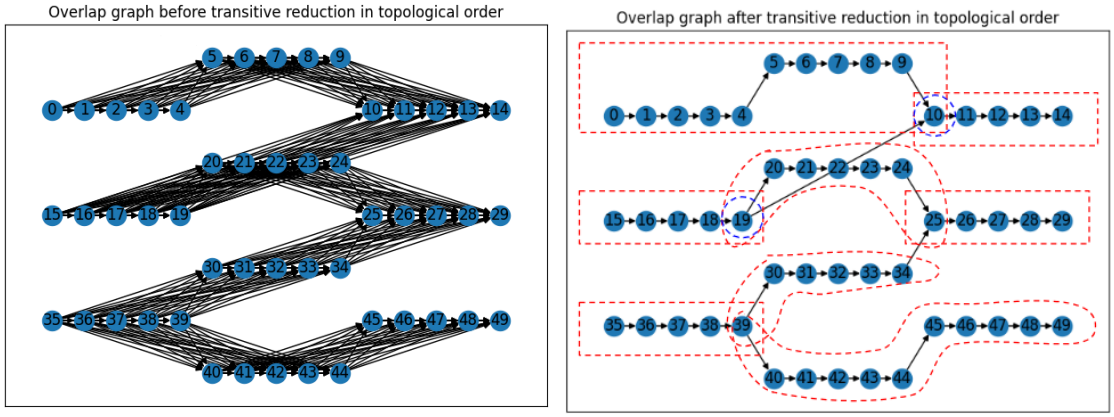

Figure 8: Overlap graph before and after transitive reduction. The reduced graph highlights contigs in red.

Salmonella Dataset: Results

For the *Salmonella* dataset, we have computed sketches for all the reads, with sketch size H=1256. This is a 2D numpy array of 1×10^6 * 1256 int32 values. By subsetting this, all sketch sizes less than 1256 were also obtainable. Using this matrix, we have computed the non-boolean Jaccard Index matrix (the raw Jaccard Index estimates) for the sketch size of H=1256. Using these results, we can generate an overlap graph for various Jaccard Index estimate thresholds. In Figure 9, we have shown the Number of Candidate Alignments provided by MHAP as a function of the Jaccard Index Threshold. The total number of alignments is computed as the number of elements in the upper triangle of the matrix excluding the main diagonal, which is equal to computing all unique pairwise alignments directly. The range of thresholds between 0.025 and ~0.12 give the range of a genuine tradeoff between computation saved (number of alignments performed) and the chance of missing overlapping reads. In the ideal scenario, a read will have *readDepth* sequences overlapping its suffix and *readDepth* sequences overlapping its prefix; if our data has 2 to 5 *readDepth* that would mean every read should have approximately 4 to 10 candidate reads to overlap with, so choosing thresholds yielding between (1×10^5 * 2) to (1×10^5 * 5) Number of Candidate Alignments would be appropriate. Going below the range will require too many alignments and going above will result in too few alignments. Lastly, even with a threshold of 0.025, we would be able to get significant cost savings using MHAP, performing only ~75% of the total pairwise alignments (Figure 9).

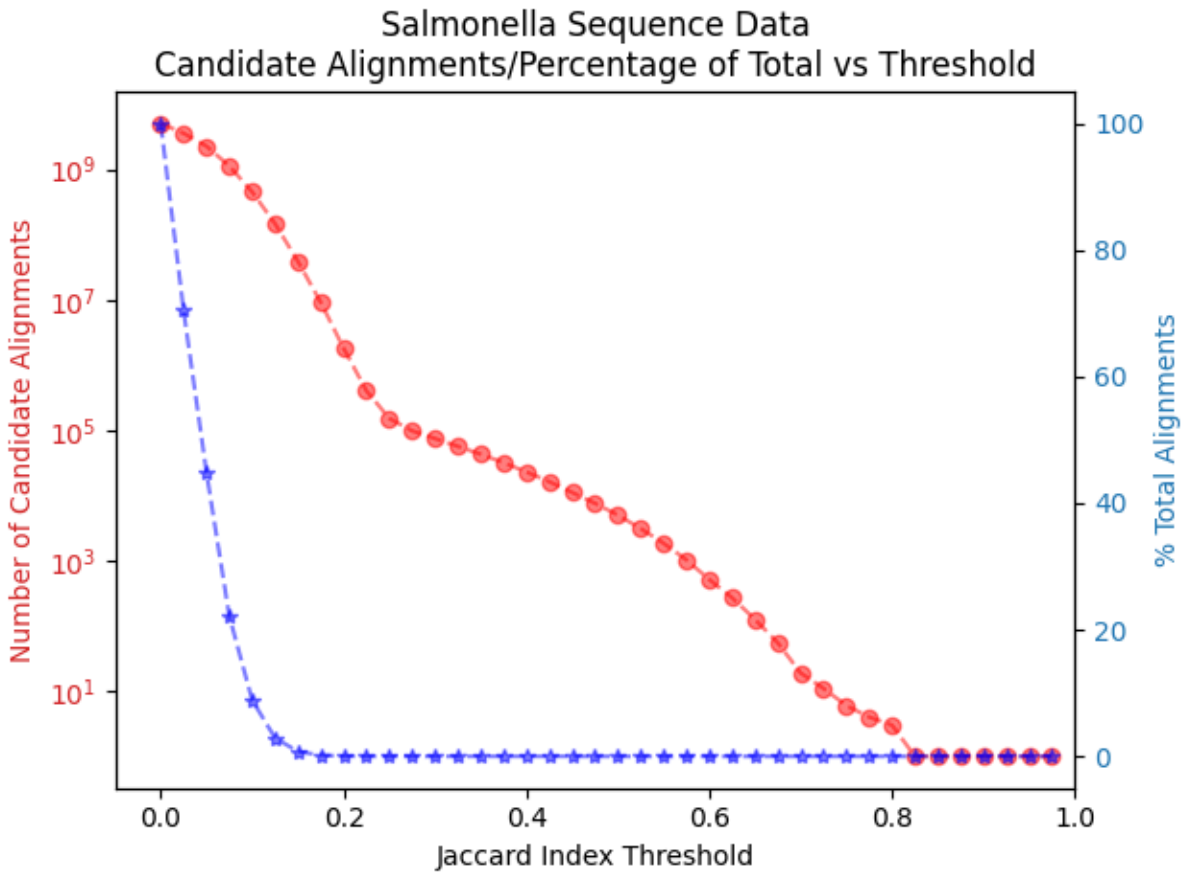

Figure 9. Plot of Number of Candidate Alignments vs Jaccard Index Threshold (left y-axis) and Percentage of Total Alignments vs Jaccard Index Threshold (right y-axis) for the *Salmonella* dataset. H=1256, k=16

Discussion & Conclusion

In this work, we have implemented an OLC pipeline for *de novo* assembly of genomes from error-prone long-read sequencing data. Our attempted assembly of the *Salmonella* Hadar genome from the input sequencing data did not proceed to completion due to the computational demands of assembly and the relatively unoptimized nature of our implementation, despite our efforts to parallelize execution of the Overlap phase, which has been identified as the bottleneck to OLC assembly (Chu et al., 2017). Nevertheless, all phases of the assembly were tested on smaller, synthetic datasets, ultimately to mixed results depending on the nature of the data. Execution of the pipeline even on the smaller datasets took many hours, reflecting the critical need for highly optimized implementations of genuine assembly pipelines. In hindsight, our decision to implement our pipeline in Python, even considering the relatively small number of reads in the sequencing experiment, our use of Amazon Web Services cloud computing, and parallelization efforts was rather naive. Clearly, the efficiency and low-level control afforded by compiled, rather than interpreted, programming languages, such as C or Rust, make these more attractive options for implementation of genome assembly.

Performance considerations notwithstanding, testing of our smaller synthetic datasets revealed significant limitations to the approach we implemented. For two of the three datasets constructed, our pipeline generated overwhelmingly small contigs, and in terms of contig length, the third one, while an improvement, was only marginally better. We speculate that our chosen method of establishing the direction of overlap graph edges may lack adequate specificity, as we elaborate upon below. Additionally, our pipeline's practical utility is also limited by its lack of consideration of read orientation and reverse complementarity – though the ability of our synthetic datasets, which were constructed from forward-direction reads only, to be successfully assembled would not be thwarted by this shortcoming, a genuine sequencing dataset certainly would be. It is also important to note that successful generation of contigs from an OLC pipeline does not complete a genome assembly; scaffolding or the resulting contigs' orientation is still required. A variety of scaffolding methods exist, including those dependent on paired-end read data or those guided by reference genomes, but these are beyond the scope of the work shown here.

When we compute the overlap graph, we perform 3 alignments for each pair of reads (A,B) deemed sufficiently similar by MHAP. The three alignments are fitting alignment between A and B, and overlap alignment in two directions - the suffix of A aligned to the prefix of B and suffix of B aligned to prefix of A. If the overlap alignment score exceeds the fitting alignment score, we add a directed edge between these two reads in the overlap graph; the directionality is based on which overlapping alignment scored higher (A to B or B to A). When comparing these scores, we did not set a threshold for the minimum alignment score required for an alignment to be considered reasonable. It is possible that the result from MHAP found had an accidental collision rather than a meaningful one. In other words, all three alignment scores may be very poor, meaning that no corresponding edges should be placed in the overlap graph irrespective of which alignment score is larger - alignment or overlap. Hence, setting a suitable statistical threshold to check for such cases may improve our results.

References

Aho, A. V., Garey, M. R., & Ullman, J. D. (1972). The transitive reduction of a directed graph. *SIAM Journal on Computing*, *1*(2), 131-137.

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., Pyshkin, A. V., Sirotkin, A. V., Vyahhi, N., Tesler, G., Alekseyev, M. A., & Pevzner, P. A. (2012, May). *Spades: A new genome assembly algorithm and its applications to single-cell sequencing*. Journal of computational biology : a journal of computational molecular cell biology. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3342519/

Baker, M. (2012). De novo genome assembly: what every biologist should know. *Nature methods*, *9*(4), 333-337.

Berlin, K., Koren, S., Chin, C. S., Drake, J. P., Landolin, J. M., & Phillippy, A. M. (2015). Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology*, *33*(6), 623-630.

Bioinformatics Workbook. Terminology. *Retrieved from bioinformaticsworkbook.org at https://bioinformaticsworkbook.org/introduction/dataTerminology.html.*

Broder, A. Z. & digital Systems Research Center. (1997). On the resemblance and containment of documents. In Digital Systems Research Center. digital Systems Research Center. https://www.cs.princeton.edu/courses/archive/spring13/cos598C/broder97resemblance.pdf

Chin, C. S., Peluso, P., Sedlazeck, F. J., Nattestad, M., Concepcion, G. T., Clum, A., ... & Schatz, M. C. (2016). Phased diploid genome assembly with single-molecule real-time sequencing. *Nature methods*, *13*(12), 1050-1054.

Chu, J., Mohamadi, H., Warren, R. L., Yang, C., & Birol, I. (2017). Innovations and challenges in detecting long read overlaps: an evaluation of the state-of-the-art. *Bioinformatics*, *33*(8), 1261-1270.

Dasgupta, A., Kumar, R., & Sarlós, T. (2011, August). Fast locality-sensitive hashing. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 1073-1081).

De Coster, W., D'hert, S., Schultz, D. T., Cruts, M., & Van Broeckhoven, C. (2018). NanoPack: visualizing and processing long-read sequencing data. *Bioinformatics*, *34*(15), 2666-2669.

Freedman, A. H., Gaspar, J. M., & Sackton, T. B. (2020). Short paired-end reads trump long single-end reads for expression analysis. *BMC bioinformatics*, *21*, 1-11.

Khan, A. R., Pervez, M. T., Babar, M. E., Naveed, N., & Shoaib, M. (2018). A comprehensive study of de novo genome assemblers: current challenges and future prospective. *Evolutionary Bioinformatics*, *14*, 1176934318758650.

Koren, S., Walenz, B. P., Berlin, K., Miller, J. R., Bergman, N. H., & Phillippy, A. M. (2017). Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, *27*(5), 722-736.

Langmead, Ben. Overlap Layout Consensus assembly. *Retrieved from cs.jhu.edu at https://www.cs.jhu.edu/~langmea/resources/lecture_notes/assembly_olc.pdf.*

National Institutes of Health. (2015). National Human Genome Research Institute. Talking glossary of genetic terms. *Retrieved from genome.gov at http://www.genome.gov/glossary*.

National Institutes of Health. Sequence Read Archive: Long-read whole genome sequencing of Salmonella Hadar (SRR27959541). *Retrieved from trace.ncbi.nlm.nih.gov at trace.ncbi.nlm.nih.gov/Traces/?view=run_browser&acc=SRR27959541.*

Sievers, F., Wilm, A., Dineen, D., Gibson, T. J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Söding, J., Thompson, J. D., & Higgins, D. G. (2011). Fast, scalable generation of high‐quality protein multiple sequence alignments using Clustal Omega. *Molecular systems biology*, *7*(1), 539.

Sung, W.-K. (2017). *Algorithms for next-generation sequencing*. Boca Raton, Florida: Chapman & Hall/CRC.

Watson, J. D., & Crick, F. H. (1953). Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, *171*(4356), 737-738.

Zhang, H., Jain, C., & Aluru, S. (2020). A comprehensive evaluation of long read error correction methods. *BMC genomics*, *21*, 1-15.