**Computational Biology 309: Lab 3 - Genome assembly of COVID-19 virus and *Carsonella ruddii bacterium***

Author: Yanni Guo

Partners: Xinduo Fan and Kairuo Yan

Date: October 28, 2021

**Introduction:**

Genome assembly is the process of putting short DNA sequences called reads that overlap each other to reconstruct the original genome (Compeau, and Pevzne, 2015). Assembling reads of genomes is important because biologists still do not have the technology to read an entire genome from start to end, and the location of the reads in the genome is unknown (Compeau and Pevzne, 2015). In addition, the assembly of reads can help analyze long DNA sequences and help with studies of diseases in patients (Compeau and Pevzne, 2015). Researchers can use existing software to assemble genomes. Some assembler software is based on the de Bruijn graph (Compeau, P., 2021). Maximal non-branching paths can be derived from the graph that helps to generate contigs. Contigs are overlapping DNA sequences and reconstruct the original DNA sequence (Compeau and Pevzne, 2015). This lab consists of two independent parts. In part one of the lab, we used an existing assembler called SPAdes assembler to sequence the SARS-CoV-2 virus. In the second part of the lab, we constructed functions to help researchers build a de Bruijn graph and contigs to rebuild the *Carsonella ruddii* bacterium genome.

To assemble genomes, biologists clone DNA molecules and break them down into random fragments. Then, the fragments are assembled to form contigs and use different maps to reconstruct the original DNA sequence (Compeau and Pevzne, 2015; Compeau, P., 2021).

To create a de Bruijn graph, the k-mers (DNA base pattern of a certain length) are converted to ((k-1)-mers) and are represented as nodes and directed edges will connect the nodes. Each node includes a prefix and a suffix of the k-mers, identical labeled nodes are glued together, and the edges are labeled with k-mers representing the overlap of the nodes it is connecting. There are no repeating nodes in the graph (Compeau and Pevzne, 2015). Thus, we created a function to read the FASTA file of *C. ruddii* reads to obtain its k-mers, and another function constructed a de Bruijn graph from the set of k-mers.

Maximal non-branching paths were derived from the de Bruijn graph. The paths are nodes with in-degrees and out-degrees equal to 1, except for the start and end nodes. The sequence of DNA bases that makes up the paths must be present in any possible assembly with a given k-mer. The strings that are spelled out in the paths correspond to a contig in the genome. Biologists must break the genomes into contigs because repeats of nodes would prevent a Eulerian path (Compeau and Pevzne, 2015). We attempted to find the maximal non-branching paths with a function that calculated the in-degrees and out-degrees of the nodes in the graph. We wrote another function that creates contigs from the set of paths.

After the contigs are found, the contigs are assembled to reconstruct the original genome. The N50 statistic is used to measure the quality of the assembly. N50 is the maximal contig length, where the total contig’s length is less than or equal to 50% of the entire length of all contigs. A higher N50 value indicates you have a small number of contigs with long lengths. A higher N50 value is preferred to have enough overlaps of k-mers and effective coverage of the genome (Narzisi, G. and Bud M., 2011). Thus, we created a function to get the highest N50 value with corresponding k value, contigs, and contigs’ lengths. Finally, the best set of contigs are used to find the ordering of the reads with paired-end reads alignment. So, we created a function that determined the order of reads and executed it as a new FASTA file.

This lab report will discuss how the SARS-Cov2 virus genome and the *Carsonella ruddii* bacterium genome were assembled, analyze the results, and discuss how our best set of contigs can reconstruct the *C. ruddii* genome.

**Methods:**

We used SPAdes to assemble the SARS-Cov2 genome for part 1 of the lab. We wrote seven different functions aside from the *main* function to reconstruct the *C. ruddii* genome for part 2 of the lab. The process of using SPAdes is discussed in this method section. Each function’s parameters, variables, and algorithm will be explained to answer the lab’s questions and overall goal.

For the first part of the lab, we used the SPAdes assembler and ran the software on the Australian service of Galaxy (Compeau, P., 2021). All analyses on the assembly of the SARS-Cov2 genome were performed in the history of our Galaxy account. Next, we answered exercises one through nine by studying the components of the SARS-Cov2 sequencing data from the NCBI website (Run Browser : Browse: Sequence Read Archive : NCBI/NLM/NIH). Next, we assembled the SARS-Cov2 genome using SPAdes assembly and set the settings according to Compeau’s website (Compeau, P., 2021). After the assembly was executed, the output was a dataset containing the number of contigs the assembly produced and the lengths of each contig. Lastly, we obtained an image of the assembly graph by using the Bandage program in the SPAdes assembler.

For the second part of the lab, our program required the FASTA file of the *C. ruddii* genome, which included read labels and reads. The *readpairs()* function read the FASTA file and returned a list of reads and a list of k-mers. The function took two parameters: *name* is the FASTA file of the genome and *k* is the length of the k-mers. The SeqIO.parse() function separated the read labels and reads. The function moves horizontally along the reads sequences to create the k-mers, the number of bases in each k-mer is the value of k, and the function moves along the reads in increments of one to find new k-mers.

The *create\_deBruijn* function created a de Bruijn graph from a set of k-mers. The function takes one parameter: *kmers* (a list of strings), the set of k-mers from the reads. The de Bruijn graph is formatted as a dictionary where the keys are nodes ((k-1)-mers) and values as lists of nodes with at least one directed edge that connects to its key node. The function iterates over the set of k-mers if the k-mers are not in the k-mer set dictionary called *count* then the node prefix is all the base of the k-mer except for the last base (the first (k-1)-mer). If the node prefix was not in the graph dictionary, we created nodes in all three dictionaries with its initialization. The node suffix is defined as (the last (k-1)-mer) in current k-mer). The node suffix followed the same conditions as the prefix. The node suffix is appended to the list with “node prefix” as the key representing the next nodes in the graph. After documenting the edge in the graph, we added the in-degrees for the suffix to the out-degree for the prefix node. The nodes are recorded to prevent repeats of nodes in order to find a maximal-non branching path. The function has three outputs: a graph dictionary with nodes and edges, a dictionary for in-degrees of each node in the graph, and a dictionary for out-degrees of each node in the graph.

Next, we created the GetMaximalNonbranchingPaths() function to find the maximal non-branching paths in the de Bruijn graph, where the in-degrees and out-degrees of internal nodes equal one. The three parameters are *graph*: dictionary with keys as nodes and values as following nodes, *inDegrees*: a dictionary where keys are nodes and values as the node’s in-degree value = 1, *outDegrees*: a dictionary where keys are nodes and values as the node’s out-degree value = 1. The paths list recorded the maximal non-branching path, and the nodes11 dictionary recorded the set of all nodes that have both in-degree and out-degree equal to 1. In the first for loop, the function iterates over nodes in the graph, not in the nodes11 dictionary. Each iteration found one path, and each path had one start node. The current node was defined as the node that followed the start node and was not the start node. The function searched for current nodes and skipped over nodes in the nodes11 dictionary and nodes that did not have a following node to prevent repetitions of nodes. While the current node was in the nodes11 dictionary, the next node added to the path was the node that was the first value of the graph dictionary. Then the next node became the current node and reiterated in the for loop. Each path was checked and added to the path list. In the second for loop, the function iterated over the nodes11 dictionary to find nodes that did not connect to the graph’s nodes (isolated circuits) and add the circuit to the path. The current node was defined similarly to the first for loop. While the current node was not the start node, then the current node was added to the path, and the next node that followed the current node was added to the path. The output is a list of maximal non-branching paths in the graph and is made up of k-mers.

After obtaining the paths, the *constructContigs()* function created contigs from the set of paths of the previous function. This function took one parameter: *paths*, which is a containing that contains ordered ((k-1)-mers) that form the maximal non-branching path in the de Bruijn graph. The function iterated over the *paths* to create contigs by using all the bases of the start node, then continuously added the last base of the nodes following the start node. Finally, the function returned a list made of contigs.

The *calculateN50()* function created contigs from the set of reads with different k values and used the N50 statistics to measure the quality of the assembly. One parameter was taken in the function: *l,* which is a list of reads. Each list consisted of the contigs and the function moved along the bases horizontally. The function first calculated 50% of the sum of all the contigs’ lengths and sorted the lengths in order. Then the function moved down the list to find the contig length that made up 50% of the total length of contigs. The output is the N50 value.

Lastly, the *orderingContigs()* function determined an ordering of the contigs by using paired-end reads in the original FASTA file. The function took two parameters: *contigs* is a list of *contigs*, and *reads* is a string that are the reads of the original FASTA file. The function examined each contig and its pair-read. For each for loop, the function searched for the paired-read number of each contig that was in a different contig and searched for the “x” value (either 1 or 2) from the read labels (read n/x) in other contigs. If the paired-reads number is in the same contig as itself, assemble then ignore the contig. Different contigs that have the same paired-read number and x value can be assembled. The output was a list of pairs in a new FASTA file.

In the main() function, all previously mentioned functions are called when given the FASTA file for the C\_ruddii bacterium. The printed results are the best N50, best K value, number of contigs, and the length of each contig.

**Results:**

In part one of the labs, we answered exercise questions 1 to 9 to understand genome assembly for the SARS-Cov2 genome by using the SPAdes assembler. This section displays the results for the SARS-Cov2 genome assembly and interprets the results.

The answers to questions 1 and 2 are stored in Table 1. We used the SARS-Cov2 genome data from the NCBI (“Run Browser : Browse : Sequence Read Archive : NCBI/NLM/NIH”) to answer the questions. The table shows that this genome is small, has low GC % content, and has a very high % coverage of reads. The small genome would have fewer reads to assemble together, is less stable due to the low GC content, and has several overlaps of bases at a certain location with the SARS-Cov genome (Compeau, and Pevzner, 2015; Kalyanaraman, A., 2011).

|  |  |
| --- | --- |
| Number of bases in the SARS-Cov2 genome | 186.5Mbp |
| GC content of reads | 38.2% |
| Coverage of reads | 98.6% |

**Table 1:** Answers for exercise questions 1 and 2. Values correspond to the SARS-Cov2 read file, including the number of bases, GC content, and read coverage.

The response to question 3 is put in Table 2. There are two reads and each read has a read label. The 15213.2 read is shorter because short reads are cheaper to produce but longer reads have more sequences to be used in genome assembly. Observing the read-pairs at certain read spots helps generate the k-mers and nodes to make the de Bruijn graph. The read labels help with assembling the contigs derived from the graph (Compeau and Pevzne, 2015).

|  |  |
| --- | --- |
| First read-pair at spot #15213 | >gnl|SRA|SRR11528307.15213.1MN01288:4:000H32WJK:1:11101:22090:17095 |
| First read at spot #15213 | CAACAAGGCCAAACTGTCACTAAGAAATCTGCTGCTGAGGCTTCTAAGAAGCCTCGGCAAAACGTACTGCCACTAAAGCATACAATGTAACACAAGCTTTCGGCAGACGTGGTCCAGAACAAACCCAAGGAAATTTTGGGGACCAGGA |
| Second read-pair at spot spot #15213 | >gnl|SRA|SRR11528307.15213.2 MN01288:4:000H32WJK:1:11101:22090:17095 |
| Second read at spot spot #15213 | TCAATATGCTTATTCAGCAAAATGACTTGAT |

**Table 2:** Answers for exercise question 3. First and third row are data for the read-pair at spot #15213, and the second and fourth rows are data for the reads of the read-pairs.

For question 4, the Phred score can predict the chance that a base is incorrect and help researchers decide which read to keep in the genome assembly. So the Pred scores can help determine reliable maximal non-branching paths in the de Bruijn graph and contigs. In addition, strategy can be used to prevent repeats of nodes in the graph (Compeau and Pevzne, 2015). Question 5: I think Illumina has four steps to consider to determine the quality score of a given base. The steps include sample preparation, cluster generation, sequencing, and data analysis. Question 6: the histogram shows the Phred quality scores from 2 to 37 of the whole dataset, and each quality score bin is the value for the length of the reads. I think the quality scores are bad because the histogram has a left-skew instead of a bell-shaped curve that would signify a normal distribution. So the Phred quality scores do not appear to be evenly distributed.

Table 3 shows the quality scores of reading “8595/1” to answer question 7. The quality scores are good because the majority of them are 37. The quality scores that should be of worry are the nucleotides with quality scores of 14, 28, or 32.

|  |  |
| --- | --- |
| Quality scores | 14, 28, 32, and 37 |

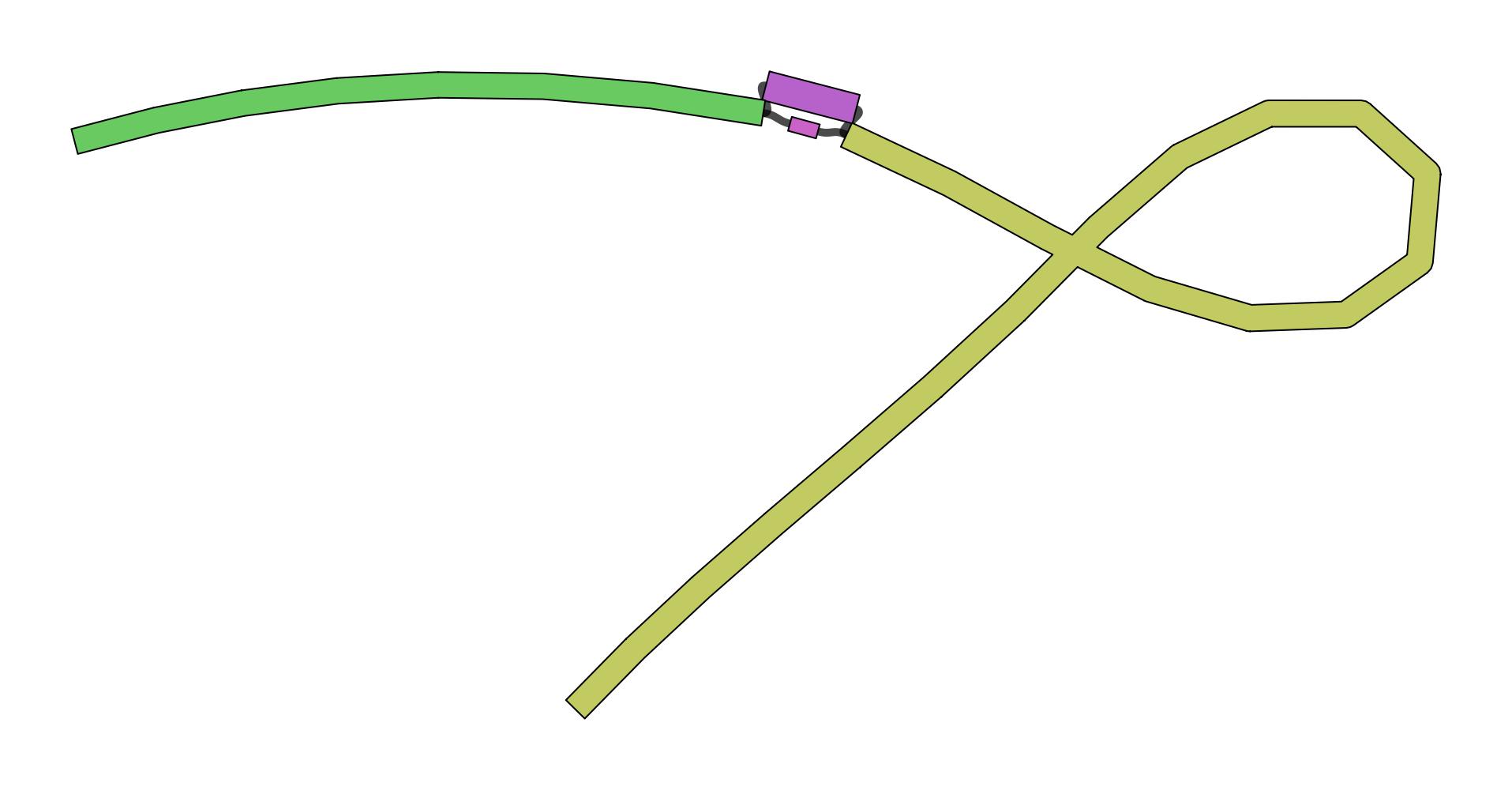
**Table 3:** The table shows quality scores of reading “8595/1” of the SARS-CoV2 genome. Data answers question 7.

Table 4 shows the information on the contigs produced by the SPAdes assembler to answer question 8. I think the coverage in this context is the length of the reads that match the DNA sequence at a specific genome location. The assembly of the contigs forms a scaffold of the original SARS-CoV2 genome. A higher coverage indicates that the assembly of reads aligns with the majority of the original read and the quality of the assembly is good (Compeau and Pevzne, 2015).

|  |  |
| --- | --- |
| Number of contigs in the assembly | Two contigs |
| Length of contig1 | 147 |
| Length of contig 2 | 29,600 |

**Table 4:** The table shows the information of contigs the SPAdes assembler produced for the SARS-CoV2 genome. Data answers question 8.

Figure 1 shows the assembly graph of the SARS-CoV2 genome. The green and yellow segments are the two contigs, the purple segment is the gap between them, and they are connected to form a scaffold. Contig 2 is the yellow-coded contig, and contig 1 is the green-coded contig. The scaffold is a reconstructed section of the entire genome. The contigs are assembled with their paired-end results (Compeau and Pevzne, 2015). I think due to the gap, the contigs could be hiding errors of bases in the assembly. Perhaps there are repeats of nodes or errors in the ordering of the contigs (Compeau and Pevzne, 2015). I think contig 2 exists near read-pair 8595/1 of the genome (Compeau and Pevzne, 2015).



**Figure 1:** Diagram showed the genome assembly of SARS-Cov2 using SPAdes assembler. The assembly was made of two contigs connected by pairing-end sequences and overall created a scaffold.

For part two of the lab, we used the *C. ruddii* bacterialgenome to test our functions of creating a de Bruijn graph and reconstructing its genome. We chose this bacterium because it is a small genome with only 160,000 base pairs (Katsir, Leron, et al., 2018), and the FASTA file was composed of error-free read-pairs. So it will be easier to analyze short read segments of the bacterial DNA sequence, create a graph from the reads, construct contigs from the graph, and assemble all the contigs to reconstruct the genome (Compeau and Pevzne, 2015). We used the *readpairs()* function to read the *C. ruddii* read FASTA file and obtained a list of reads and a list of k-mers from the file. The length of each k-mer was determined by the value of k. The list of reads and k-mers will be used in the following functions to create a de Bruijn graph and help us reach the overall goal of reconstructing the *C. ruddii* genome.

The list of reads and k-mers were used in the *create\_deBruijn()* function to create the de Bruijn graph. We obtained three dictionaries from this function: a graph dictionary with keys as nodes and values as edges that connect to the keys, a dictionary where the keys are nodes and values as the number of in-degree the node has, and a dictionary where the keys are the nodes and the values as the number of out-degree the node has.

Using the three dictionaries from the *create\_deBruijn()* function, the *GetMaximalNonbranchingPaths()* function found maximal non-branching paths from the de Bruijn graph and returned the paths as a list. In the introduction and method, each of the paths corresponds to a contig that can be assembled. So our *constructContigs()* returned a list of contigs from the list of paths in the previous function. Next, the *calculateN50()* function used the N50 statistics to evaluate the quality of the contig assembly and returned an N50 value for the function’s assembly. As mentioned in the introduction and method, we wanted a high N50 value which signified we had small numbers of long-lengthed contigs in the assembly to get enough overlaps (Compeau and Pevzne, 2015). After obtaining the N50 value, the *orderingContigs()* function used the list of contigs and reads of the original FASTA file of the *C. ruddii* genome to order the contigs with paired-end reads and executed it in a new FASTA file called “lab3\_result.fasta”. Table 5 shows the information on the contigs and the quality of their assembly with an N50 value. The high N50 value implies the genome assembly that our function produced has high quality and has a large % coverage value (Compeau and Pevzne, 2015). When k = 35 we got the highest N50 value. There are 16 contigs and the lengths of each contig are in table 5.

|  |  |
| --- | --- |
| Highest N50 value | 25,419 |
| K-value | 35 |
| Number of contigs | 16 |
| Length of each contig | [43, 46, 232, 607, 1147, 1175, 1686, 2898, 5621, 10674, 11784, 11863, 14378, 25419, 26425, 48930] |

**Table 5:** The table showed the maximum N50 value of contig assembly with the corresponding K value (between 20 and 40). The number of contigs in the assembly and each contig’s length.

**Discussion:**

Overall, this lab had two independent parts on strategies to assemble genomes. In the first part, we studied the SARS-CoV2 genome dataset to obtain its reads and PHRED and used the SPAdes assembler to reconstruct the genome. Then, we viewed the assembly in an assembler graph (Figure 1) which showed two contigs and a gap between them. Contig 1 was 147bp, and contig 2 was 29,600bp. In part two of the lab, we constructed functions to build a de Bruijn graph and found the best set of contigs to assemble the *C. ruddii* genome. The new assembly of contigs was exported as a FASTA file. We used our functions to determine the maximum N50 value of our program’s assembly: 25419, the k-value: 35, the number of contigs in the assembly: 16, and the lengths of each contig.

We hypothesized that finding other best sets of contigs and genome assembly can be completed by varying the value of k and the length of the reads (Junemann, S, et al., 2014). The k-value determined the length of the k-mers used to create the nodes and directed edges of the de Bruijn graph, so variation in the k-value would change the graph’s structure (Junemann, S et al., 2014). Biologists would derive different paths from the graph, resulting in different contigs and lengths, N50 value, coverage, and order of contigs (Junemann, S. et al., 2014). Thus, we would result in a different genome assembly. The study concluded that the best assembler does not exist, and different sets of data require different types of assemblers (Junemann, S. et al., 2014).

I think both the SPAdes assembler and our functions could produce a good quality assembly of genomes. Because figure 1 shows two contigs with long lengths that reconstruct the SARS-CoV2 genome. Table 5 showed that the assembly of the *C. ruddii* genome produced by our function had a high N50 value with a large k-value and a small number of contigs with long contig lengths. However, the resulting assembly of the SPAdes assembler and our program only offers one of the possible genome assemblies. Other possible assemblies of contigs can be formed from different values of k and read lengths.

**Bibliography:**

Compeau, and Pevzner. Bioinformatics Algorithms: An Active Learning Approach, Active

Learning Publishers, 2015

Compeau, Phillip. “SARS-COV-2 Software Assignment: Genome Assembly and Annotation:

Phillip Compeau.” *Phillip Compeau, Carnegie Mellon University*, 6 Aug.

2021,compeau.cbd.cmu.edu/online-education/sars-cov-2-software-assignments/covid-19-

genome-assembly-assignment/.

Jünemann, Sebastian, et al. "GABenchToB: a genome assembly benchmark tuned on bacteria

and benchtop sequencers." *PloS one* 9.9 (2014): e107014.

Kalyanaraman, Ananth. *Genome Assembly*  In Padua D. (eds) Encyclopedia of Parallel

Computing. Springer, Boston, MA., 2011,

<https://doi.org/10.1007/978-0-387-09766-4_402>

Katsir, Leron et al. “Genome Sequence of "*Candidatus* Carsonella ruddii" Strain BT from the

Psyllid *Bactericera trigonica*.” *Genome announcements* vol. 6,4 e01466-17. 25 Jan.

2018, doi:10.1128/genomeA.01466-17

Narzisi, Giuseppe, and Bud Mishra. “Comparing De Novo Genome Assembly: The Long and

Short of It.” *PLoS ONE*, vol. 6, no. 4, 2011, doi:10.1371/journal.pone.0019175.

“Run Browser : Browse : Sequence Read Archive : NCBI/NLM/NIH.” *National Center for*

*Biotechnology Information*, U.S. National Library of Medicine,

trace.ncbi.nlm.nih.gov/Traces/sra/?run=SRR11528307.