

Computational Problems for Bioinformatics

By Ting Ming Yin

Department of Computer Science

The University of Hong Kong

23/5/2019

Supervised by

Dr. S.M. Yiu

Abstract

This report introduces the background of DNA including the history and the basic structure of the DNA sequence to give a basic concept of DNA and explain why DNA is important to understand organisms. This project will introduce many DNA analysing tools and skills such as Linux and string matching algorithms which are commonly used by most of the biologists as they are efficient and convenient. The objective of this project is to compare the three different genome assemblers which are available for scientists to do genome assembly. The goal of this project is to compare the performance of the assemblers given a dataset of paired-end sequencing E.coli. The performance of them will be analysed by efficiency and accuracy. To facilitate the analysis, the time taken for assembling the genome will be recorded and the statics of the generated genome will be calculated. By comparing the results, the performance of the assemblers can be justified and other biologists can choose the suitable assemblers for genome assembly according to the source of dataset.

List of Figures

Fig. 3.1: “grep” function provided in the linux system. The matching patterns “ATTC” in DNA sequence are highlighted in red color.	7
Fig. 3.2: Combining the use of “grep” and “wc -l” to show the total number of lines matching the pattern “ATTC”. “-l” means counting the number of lines.	8
Fig. 3.3: Calculate the total number of GC bases appear in a chromosome	8
Fig. 3.4: A “.fastq” file showing the base qualities of the DNA sequence. The sequence is above the “+” symbol and below that is the corresponding base qualities represented by ASCII code.	9
Fig 3.5: An overlap graph with readings and directed edge.	12
Fig. 3.6: A simple overlap graph constructed from short readings.	13
Fig. 3.7: A solution of the greedy shortest common superstring. The result will be GTACGTACGAT.	13
Fig. 3.8: A figure showing the steps of the greedy algorithm. The red strings are the nodes chosen to be merged in each step. The length of the result is 7 base pairs.	14
Fig. 3.9: A figure showing the other way of solution of the greedy algorithm using the same readings. The length of the result is 9 base pairs which has 2 more base pairs than the optimal one.	14
Fig. 3.10: This figure shows which edges can be deleted in the layout phase.	15
Fig. 3.11: This figure is a complicated overlap graph before the layout phase.	15
Fig. 3.12: This figure shows the result of the figure 3.11 after the layout phase. The regions circled by the red and blue circle are the readings that can be merged into a sequence of string called contigs, which will be used to construct the original genome. The remaining part of the graph will be left as it formed a loop which is unresolvable.	16
Fig. 3.13: Divide the reading with k length into k-1 nodes.	16
Fig.3.14: Draw the De Bruijn graph according to the result of fig. 3.13.	17

Fig. 3.15: Change the directed edges into weighted edges by counting the number of edges.	18
Fig. 3.16: A graph showing the number of nodes occur in the De Bruijn graph with error-free sequencing.	19
Fig. 3.17: A graph comparing the number of nodes appear between an error-free sequencing and one with 0.1% error. There are more distinct nodes detected in 0.1% error sequencing.	20
Fig. 3.18: There are two possible solutions of this De Bruijn graph. It does not matter whether the purple path goes first or the green path goes first.	21
Fig. 5.1: The computation time for each assemblers to finish assembling the input dataset of pair-end E.coli library. ABySS takes 1956 seconds, Ray takes 5696 seconds and Edena takes 2144 seconds.	25
Fig. 5.2: The total number of contigs generated after the assembly process. ABySS has 108 contigs, Ray has 177 contigs and Edena has 148 contigs.	26
Fig. 5.3: The total length of contigs generated after the assembly process. ABySS produces contigs with total length ~4.58Mbp, Ray produces contigs with total length ~4.50Mbp and Edena produces contigs with total length ~4.58Mbp.	26
Fig. 5.4: The N50 length of contigs generated after the assembly process. ABySS produces contigs with N50 ~106Kbp, Ray produces contigs with N50 ~40.2Kbp and Edena produces contigs with N50 ~67.6Kbp.	27

Contents

Abstract.....	ii
List of Figures	iii
Contents.....	v
Chapter 1. Introduction	1
1.1 Project Background.....	1
1.2 Motivation	2
1.3 Contribution.....	2
1.4 Objective	3
1.5 Report Outline	3
Chapter 2. Literature Review	4
2.1 AN INTRODUCTION TO BIOINFORMATICS ALGORITHMS	4
Chapter 3. Methodology.....	6
3.1 Operating system.....	6
3.1.1 Linux.....	6
3.1.2 File Format	9
3.2 String Matching Algorithm.....	10
3.3 Assembly Algorithms	12
3.3.1 Overlap – Layout – Consensus (OLC) Assembly	12
3.3.2 De Bruijn Graph (DBG) Assembly.....	16
3.4 Conclusion.....	22
Chapter 4. Genome Assembler and E.coli Dataset	23
4.1 Genome Assembler.....	23
4.2 E.coli dataset.....	24
Chapter 5. Schedule and Milestone.....	25
5.1 Result	25
5.1.1 Efficiency Evaluation.....	25
5.1.2 Accuracy Evaluation	26
5.2 Comparison	27
5.2.1 Efficiency.....	27
5.2.2 Accuracy.....	28
5.3 Conclusion.....	29
Chapter 6. Conclusion.....	30

Chapter 7. Reference	31
----------------------------	----

Chapter 1. Introduction

DNA (deoxyribonucleic acid) is the most important material in almost all organisms on Earth. It contains only four bases - adenine (A), guanine (G), cytosine (C), and thymine (T) and forms a sequence which includes all the information of the organism. DNA is like a detailed blueprint of the specific organism which decides all features of every single cell. However, these vital messages are hidden inside the very long sequence, approximately 3 billion bases in human DNA, making it difficult for scientists to understand them. Moreover, more than 99 percent of DNA between two different individuals are identical and that 1 percent of difference it that which differentiates us from each other. Scientists need an efficient and convenient tool to analysis and investigate the DNA sequence.

1.1 Project Background

In 1860s, Johann Friedrich Miescher, a Swiss chemist, identified the first DNA molecule when doing a research on white blood cell [2]. However, due to the limitation of technology, the structure of DNA could not be solved. Until 1960s, with Rosalind Franklin, Francis Crick, James Watson, and Maurice Wilkins contributions, the DNA structure was finally solved, and sequencing DNA became a possible but difficult task. In the 21st century, DNA help scientists understand many unsolved problems before, like the cause of cancer, the origin of humanity or crime solving. It becomes an important field that scientists put many effort to solve the mystery inside the DNA sequence.

However, DNA sequence is impossible to be solved manually because the information stored in a cell is about one million A4 paper printed with both sides. The DNA sequence in one human cell can be 2 meters long and more than three billion nucleotides need to be mapped [1]. Therefore, scientists use computer and software tools to help them analyse DNA sequence and this technique is referred to called bioinformatics. Although computers enhance the analysis greatly, it is still a difficult task as the information stored in one cell is about 1.5 Gb large which means handling of big data is one important challenge. This project aims to solve the computational problems and reconstruct a complete genome using the appropriate tools.

1.2 Motivation

Bioinformatics is a combination of Biology and Computer Science to construct and analyse DNA sequence. I am interested in this field because of one important project, human genome project, which was initiated in 1990 and aims to map all the human genome and understand the hidden message between the sequences [3]. This project has made lots of contribution to human society and inspired me that how important DNA is. This final year project can construct a complete genome of a marine species which can help scientists understand more about its nature and assists the development of biological technology. For example, some species do not get cancer and if scientists could find out the reason hidden in DNA, cancer may be able to be cured in the future. As a result, I choose this topic to be my final year project.

1.3 Contribution

This project can make one contribution to the field of bioinformatics.

- This project aims to compare the performance of different genome assemblers available for genome assembly. The scientists can choose suitable assembler according to the source of dataset and the specification of the machine.

1.4 Objective

The objective of this project is to compare the assembly ability of different genome assemblers using a dataset of Paired-end sequencing (2x100 base) of E. coli library. Time for assembling, total number of contigs and the N50 contig length will be used to determine the ability of the genome assemblers.

1.5 Report Outline

This report is divided into seven chapters. Chapter one provides the background and motivation of the project. Chapter two provides the literature review on the biological algorithm. Chapter three provides the methodology that will be used when achieving the goal of the project. Chapter four provides the introduction of three chosen genome assemblers. Chapter five provides the result and comparison of the genome assemblers. Chapter six provides the conclusion of the project. Chapter seven provides the reference list.

Chapter 2. Literature Review

In this chapter, literature review will be presented to provide a clear understanding of the development of biological algorithms which are used by scientists for analysing and constructing the genome. These common algorithms will be used in the project to tackle the difficulties using a more efficient and effective way.

2.1 AN INTRODUCTION TO BIOINFORMATICS ALGORITHMS

This book provides an overview of the bioinformatics algorithms used by the current scientists. It provides concepts of computer science and biology as an introduction and make a clear explanation of how they are related to the bioinformatics algorithms.

In Chapter 2, it introduces what is algorithm. In Biology and Computer Science, the expression of algorithm is totally different as biologist tends to express in word, but computer scientists express it in symbols. The difference between expression may make confusion between biologists and computer scientists when they are cooperating with others, so it is necessary to have a common standard. Then, the book introduces the basic concepts of algorithms such as recursive function and Big O expression, which is about the analysis of program's running time. These concepts are important because these can help create an efficient and effective algorithm, which is highly related to the bioinformatics algorithms introduced in the following chapters.

In Chapter 3, the author introduces the basic concept of Molecular Biology which is about DNA. These can help scientists create a correct and functioning algorithm because there are many variations while constructing genome. Without the basic knowledge, scientists may neglect the incorrect implementation of the algorithms and generate an unexpected result, which may significantly disturb the analysis of DNA.

Having the knowledge of Computer Science and Biology, the remaining chapters introduce many algorithms that handle different stages when constructing the genome. For example, in Chapter 8.10 and 8.11, it introduces how to do protein sequencing making use of greedy

algorithm and biological concepts. As a result, scientists can reconstruct the protein sequence and identify its structure.

This book has introduced many useful algorithms with enough concepts and data structure, which can help tackle the problems faced in the future work of the project.

Chapter 3. Methodology

In order to analysis the DNA sequence more efficiently, scientists use different tools to assist their works. This chapter will introduce some commonly used tools and algorithms which are commonly used by scientists when doing DNA sequence. These basic techniques will be frequently used throughout the project and should be familiarized with.

3.1 Operating system

In this section, the operating system and its functions will be introduced to give an overview of the current technology. The system has several advantages when handling the data from DNA sequencing and it is commonly used among biologists. Moreover, there are several standards for creating files of DNA sequence so that the information can be shared and accessed all over the world.

3.1.1 Linux

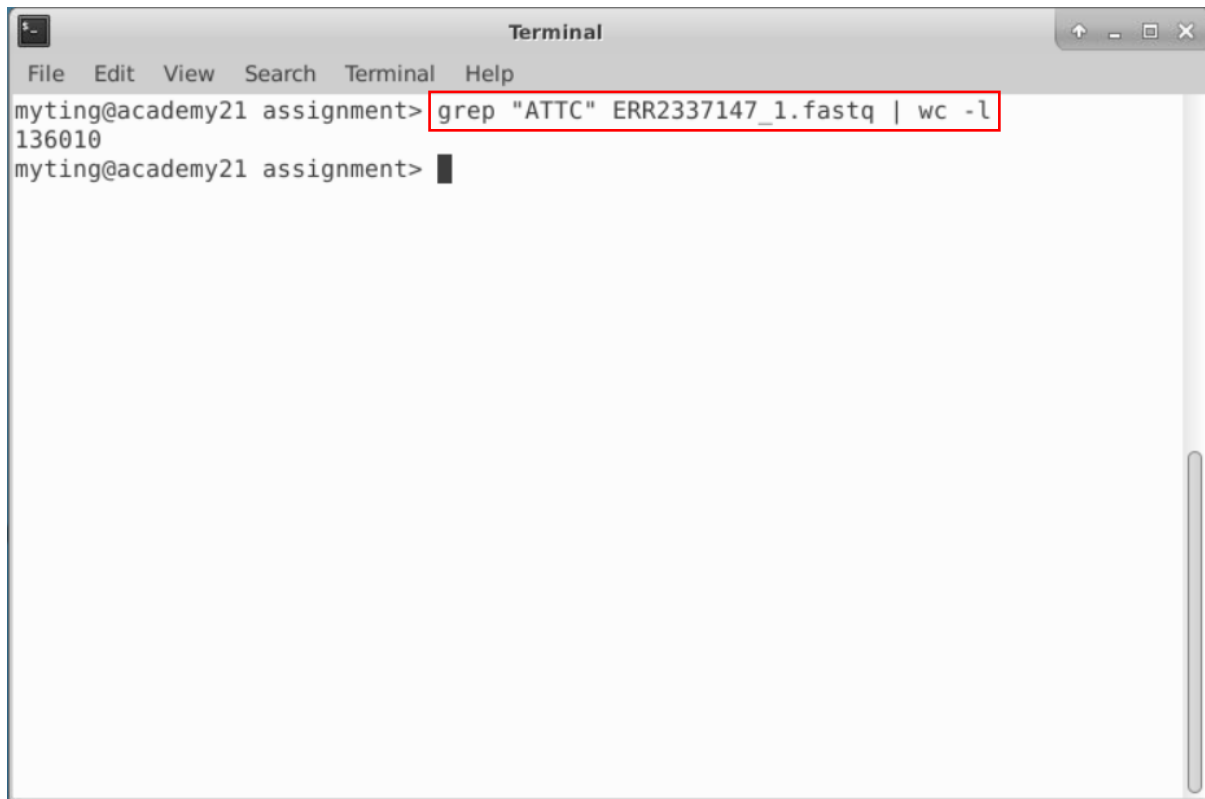
Linux is commonly used by bioinformaticians to handle and analyse the DNA sequence as it provides lots of convenient tools such as command line and file managing system which make the process simple because the functions are built in the operating system by default. Biologists do not need to construct a new program but use the functions provided.



```
myting@academy21 assignment> head ERR2337147 1.fastq | grep --color "ATTC"
GTATTAGCTCATTGATTATCTAGTCATAATTC AAGCAACTACTACAATATAACAAAATCCTTTTATAACGCAAGTTCAT
TTTATGCTACTGCTCAATTTTTTACTTTTATCGATTAAAGATAGAAATACACGATGCGAGCAATCAAATTTCTTAACAT
CACCATGAGTTTGGTCCGAAGCATGAGTGTTTACAATGTTTGAATACCTTATAAAGTTCTTATACATACTTTATAAATTA
TTTCCAAGCT
TATAACTTTGTGTGTAATTTCTAATTATCCACAATTCGAAAACTGTAAATGTGCATAAGTGGATAACTTTTCCTTCTAT
AGAGTATCTGTTAGTGAGTGTATCAAAACAGCTTGGGAAATAATTTATAAAGTATGTATAAGAACTTTATAAGGTATTC A
AACATTGTAAACACTCATGCTTCGGACCAAACCTCATGGTGATGTTATGAAATTTGATTGCTCGCATCGTGATTTTCTATC
TTTAATCGAT
TCATTGATTATCTAGTCATAATTC AAGCAACTACTACAATATAACAAAATCCTTTTATAACGCAAGTTCATTTTATGCT
ACTGCTCAATTTTTTACTTTTATCGATTAAAGATAGAAATACACGATGCGAGCAATCAAATTTTATAACATCACCATGA
GTTTGGTCCGAAGCATGAGTGTTTACAATGTTTGAATACCTTATAAAGTTCTTATACATACTTTATAAATTATTTCCAA
GCTGTTTTGAT
myting@academy21 assignment>
```

Fig. 3.1: “grep” function provided in the linux system. The matching patterns “ATTC” in DNA sequence are highlighted in red color.

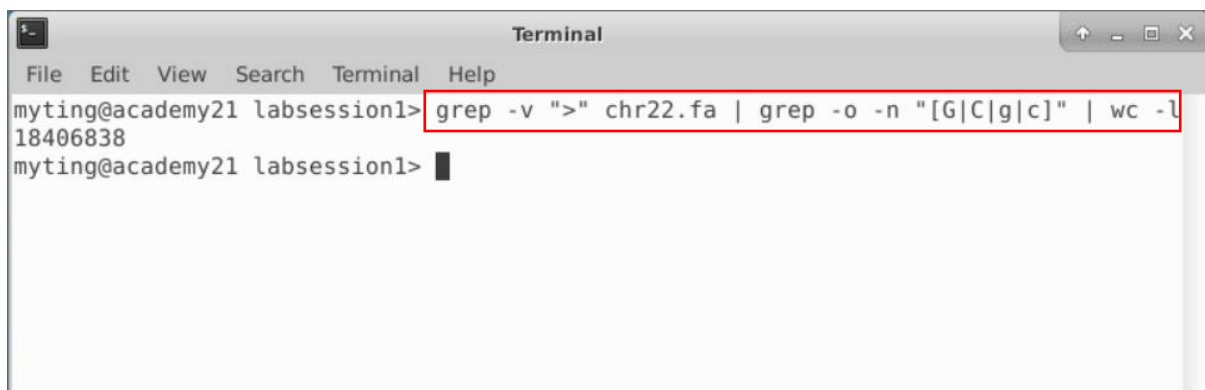
For example, the “grep” function can search a certain pattern in a file with one command line (Fig. 3.1). It can also perform different behaviours by simply adding extra symbols to the line such as “-o”, “-n” which shows the exact wording and line position of the input pattern. The features of this function make Linux powerful when the position of a genome pattern need to be found within millions of lines.



```
Terminal
File Edit View Search Terminal Help
myting@academy21 assignment> grep "ATTC" ERR2337147_1.fastq | wc -l
136010
myting@academy21 assignment>
```

Fig. 3.2: Combining the use of “grep” and “wc -l” to show the total number of lines matching the pattern “ATTC”. “-l” means counting the number of lines.

Another example is using the word count function (wc). In Figure 3.2, it shows a combination of using “grep” and “wc -l” to show the number of lines matched. With the ability of combining different commands, Linux makes things much easier when scientists want to analyse the data using one line of commands. See Fig. 3.3 which shows how to calculate the total number of base “G” and “C” in a chromosome.



```
Terminal
File Edit View Search Terminal Help
myting@academy21 labsession1> grep -v ">" chr22.fa | grep -o -n "[G|C|g|c]" | wc -l
18406838
myting@academy21 labsession1>
```

Fig. 3.3: Calculate the total number of GC bases appear in a chromosome.

3.1.2 File Format

In Bioinformatic, there are different file formats for different usage. For example, “.fastq” is used for storing the base qualities of the DNA sequence (Fig. 3.4). The DNA sequence is divided into many small sequences and below the line is the corresponding base qualities which is the possibility that the reading of the bases is incorrect due to the malfunctioning of the machine. This file can be used as a reference to determine whether the data is reliable or not.

[illegible]

Fig. 3.4: A “.fastq” file showing the base qualities of the DNA sequence. The sequence is above the “+” symbol and below that is the corresponding base qualities represented by ASCII code.

There are many other file formats such as “.BAM” and “.SAM” which store different information about the DNA sequence. Searching for the correct file format is the first step of analysing DNA sequence. Moreover, when a new DNA sequence is created, it should follow the standard of the file format to let other scientists search and access the data easily.

3.2 String Matching Algorithm

As mentioned in section 3.1.1, DNA sequencing often uses string matching method to analyse the data. However, it is not fast enough for searching the whole DNA sequence because the sequence is too long and the running time of the algorithm can be counted in hours. Therefore, there is another string matching algorithm created which is called Burrows-Wheeler Transform (BWT).

The procedure of BWT of a given sequence (ACTCG) is as follows:

1. Add a dollar sign '\$' to the end of sequence (ACTCG\$).
2. Generate all the combinations by putting the first letter to the back.

1	ACTCG\$
2	CTCG\$A
3	TCG\$AC
4	CG\$ACT
5	G\$ACTC
6	\$ACTCG

3. Sort the combinations with alphabetical order and remember the original order of the sequence.

1	\$ACTCG	6
2	ACTCG\$	1
3	CG\$ACT	4
4	CTCG\$A	2
5	G\$ACTC	5
6	TCG\$AC	3

4. Take the last position of each sequence and get a BWT sequence (G\$TACC) and a SA sequence (614253)

With the above result, the range of a string "T" is defined as [6,6] and SA of T is 3.

Therefore, the position of "T" in the original sequence is 3 which is correct.

Let the range of P be [u, v]. Then the range of adding a character x before P (xP) is [u', v'], where

$$u' = \#(\text{chars} < x \text{ in BWT}) + \#(x \text{ in BWT}[1, u-1]) + 1$$

$$v' = \#(\text{chars} < x \text{ in BWT}) + \#(x \text{ in BWT}[1, v])$$

Using the above formula, adding a character “C” before “T”

$$u' = \text{number of characters} < C \text{ in (G\$TACC)} + \text{number of characters } C \text{ in } [1, (6-1)] + 1$$

$$= 2 + 1 + 1 = 4$$

$$v' = \text{number of characters} < C \text{ in (G\$TACC)} + \text{number of characters } C \text{ in } [1, 6]$$

$$= 2 + 2 = 4$$

“CT” = [4, 4] which has SA 2 so that the position of “T” in the original sequence is 2.

This algorithm requires many spaces and running time to prepare all the result of [u', v'].

However, it can run at a constant time after preparation despite how long the matching length is. Spending more time on constructing a data base can significantly shorten the running time for the procedure of analysing.

3.3 Assembly Algorithms

In this section, two main assembly algorithms will be introduced – Overlap – Layout – Consensus (OLC) Assembly and De Bruijn Graph Assembly. These algorithms have been used by the scientists for many years to construct the genome, which is also the commonest method for genome assembly today.

3.3.1 Overlap – Layout – Consensus (OLC) Assembly

The first step of the OLC Assembly is to construct an overlap graph. When readings are generated from the sequencing machine, they will be marked as a node in the overlap graph. If it has overlapped with other node, a directed edge will be drawn between two nodes.

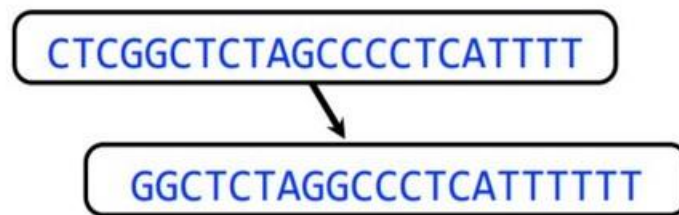


Fig 3.5: An overlap graph with readings and directed edge.

The number of base pairs overlapped will be marked on each directed edge and an overlap graph can be constructed. Here is a simple example to explain the algorithm.

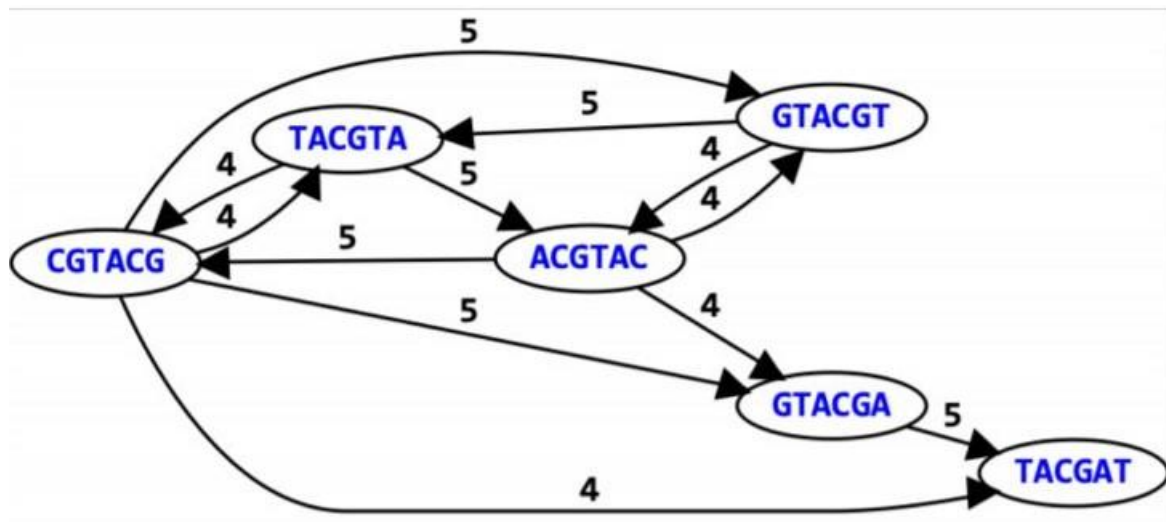


Fig. 3.6: A simple overlap graph constructed from short readings.

With the help of the overlap graph, the original genome can be assembled using the greedy shortest common superstring. One of the edge with the longest overlapped length will be chosen and the two nodes will be merged into a new string. The in-going edges of the node with overlapped prefix and the out-going edges of the node with overlapped suffix will be deleted. The steps will be repeated until the superstring is found.

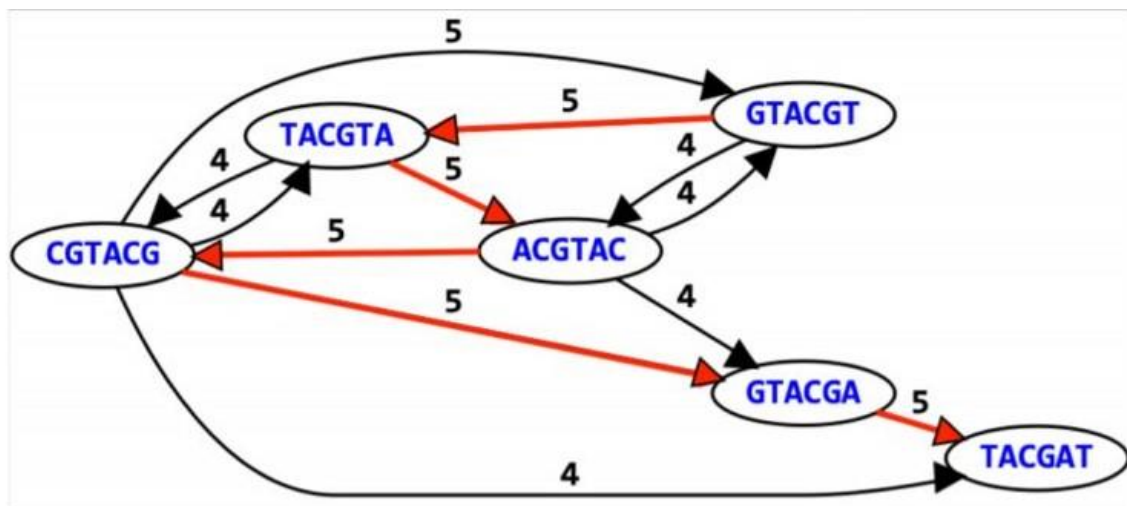


Fig. 3.7: A solution of the greedy shortest common superstring. The result will be GTACGTACGAT.

However, there are two problems that the shortest greedy common superstring cannot be used in real life. The first problem is the greedy solution may not be the optimal solution. Here is an example showing that the greedy solution is not the optimal solution.

```

AAA AAB ABB BBB BBA
AAA AAB ABB BBB BBA
AAAB ABB BBB BBA
AAAB BBBA ABB
AAABB BBBA
AAABBBA

```

Fig. 3.8: A figure showing the steps of the greedy algorithm. The red strings are the nodes chosen to be merged in each step. The length of the result is 7 base pairs.

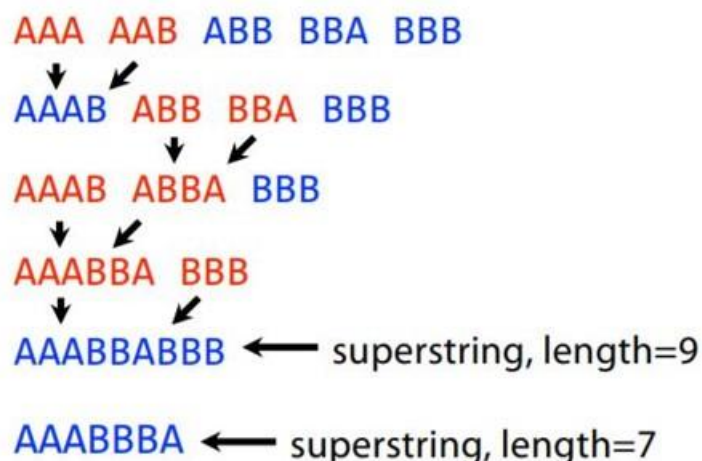


Fig. 3.9: A figure showing the other way of solution of the greedy algorithm using the same readings. The length of the result is 9 base pairs which has 2 more base pairs than the optimal one.

The second problem is that the greedy algorithm cannot solve the problem of repeated patterns. It is because the graph cannot tell how many times the pattern should be repeated. For example, if there is a sequence “AAAAAA” and the read length of the machine is 3

base pairs, it is impossible to reconstruct the original sequence. However, the overlap graph is still useful to construct part of the genome called contig.

In this OLC Assembly, repeated patterns are not possible to be constructed so that it will not be assembled. The overlap graph will be simplified into a more concise and precise graph by the layout phase in order to show the contig clearly. In this phase, some useless edges will be deleted as it can be deduced by other edges. For example, using the example in fig 3.6, “CGTACG”, “GTACGA”, “TACGAT” are three consecutive nodes with overlapped length 5. Therefore, it can be deduced that “CGTACG” and “TACGAT” have overlapped length 4. As a result, that edge can be deleted. Repeat the above steps until a clear graph is left.

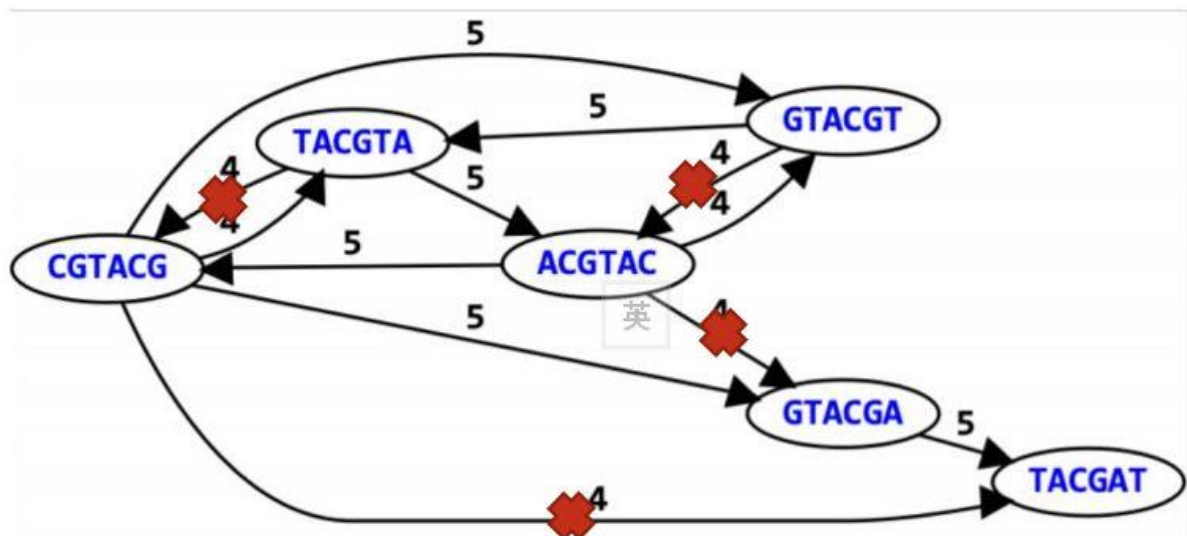


Fig. 3.10: This figure shows which edges can be deleted in the layout phase.

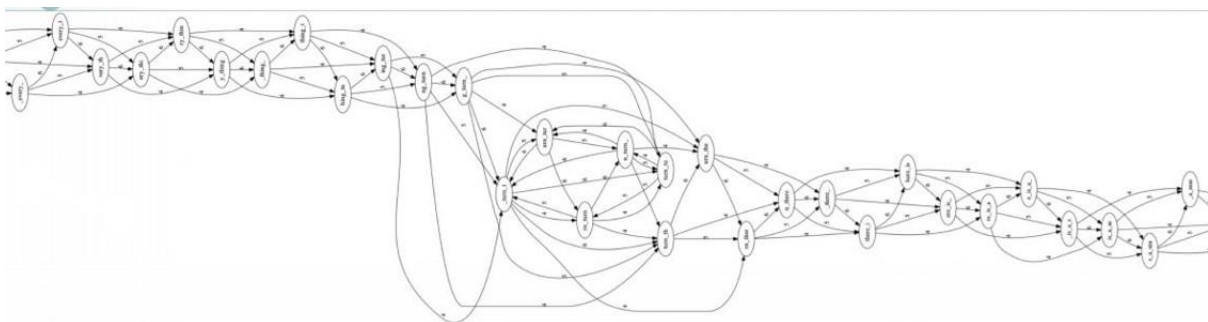


Fig. 3.11: This figure is a complicated overlap graph before the layout phase.

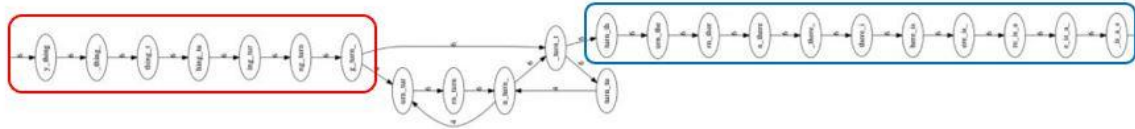


Fig. 3.12: This figure shows the result of the figure 3.11 after the layout phase. The regions circled by the red and blue circle are the readings that can be merged into a sequence of string called contigs, which will be used to construct the original genome. The remaining part of the graph will be left as it formed a loop which is unresolvable.

3.3.2 De Bruijn Graph (DBG) Assembly

DBG assembly is another way for genome assembly. It uses a different graph from the overlapped graph for constructing the genome. The name of the graph is De Bruijn graph. Firstly, the readings with k -length will be divided into two $k-1$ length nodes, one is the leftmost $k-1$ region and the other is the rightmost $k-1$ region. Then, a directed edge will be drawn between these nodes and start constructing the De Bruijn graph. Then, the same procedure will be repeated among all the readings and check whether there exist the same nodes in the graph. If yes, draw the edge from the old node. Otherwise, create a new node and draw the edge from it.

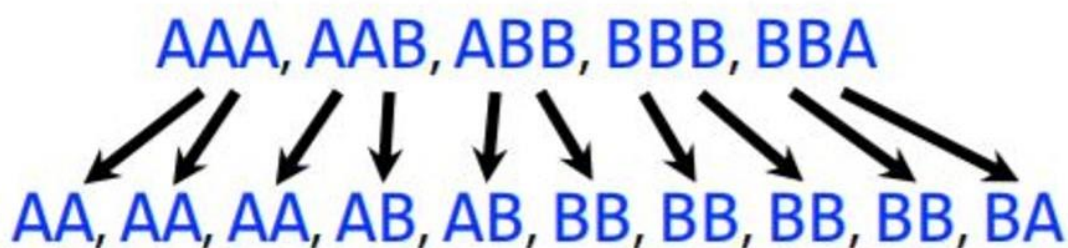


Fig. 3.13: Divide the reading with k length into $k-1$ nodes.

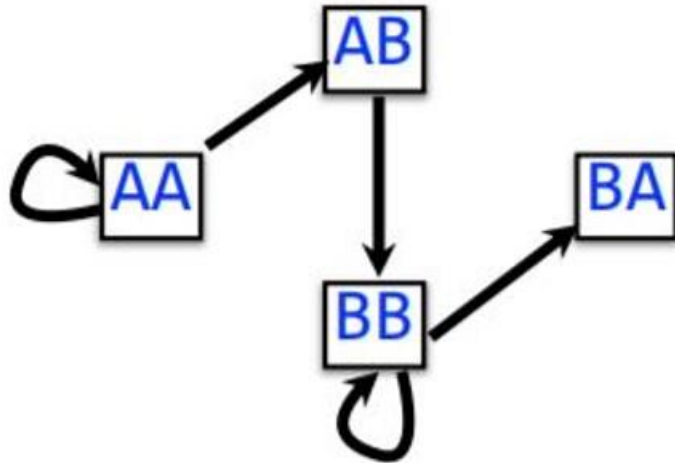


Fig.3.14: Draw the De Bruijn graph according to the result of fig. 3.13.

To construct the original sequence, it simply goes through all the edges once and construct the string by appending the starting string with destination node. For example, in fig 3.14, the result of the sequence is AAABBBBA starting from the node AA.

However, there are also two problems with this constructing method. The first problem is that in real life, the regions of the original genome will not be read only once by the sequencing machine and so there will be repeated readings which make the graph not reliable.

To tackle this problem, the directed edges will be changed into weighted edge, which counts the number of out-going and in-going edges and give a value to the edge.

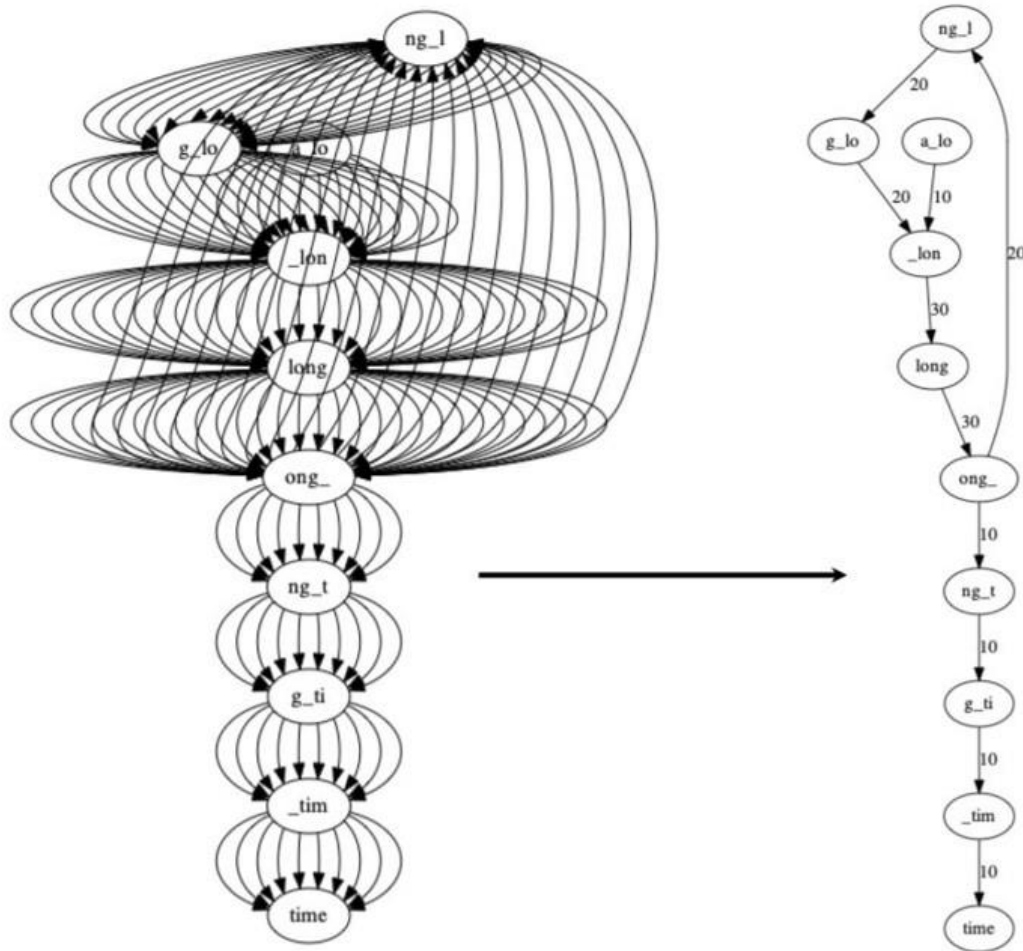


Fig. 3.15: Change the directed edges into weighted edges by counting the number of edges.

The weighted edge can be used to determine the error occurred during the DNA sequencing.

For an error-free sequencing, the number of edges should be normally distributed as shown in fig. 3.16.

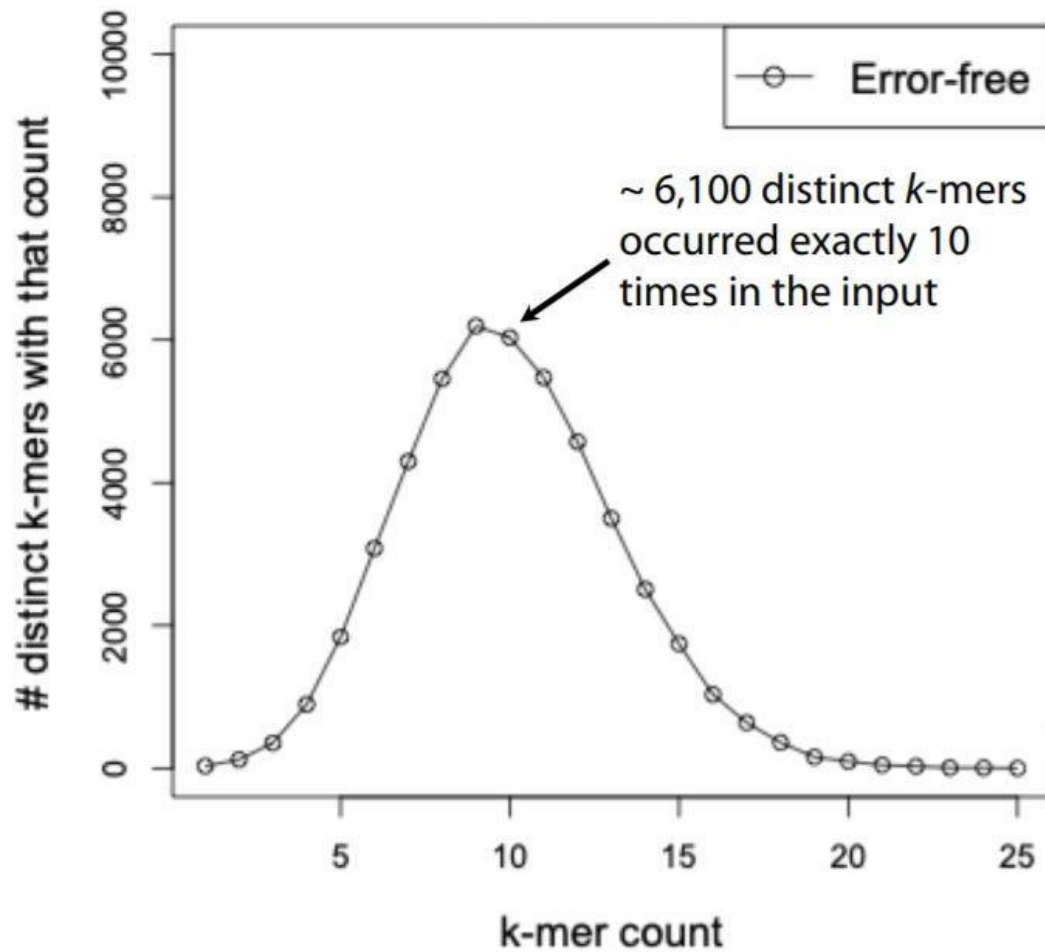


Fig. 3.16: A graph showing the number of nodes occur in the De Bruijn graph with error-free sequencing.

However, when an error occurs, there will be nodes separated from the main graph as errors are usually unique and they will create nodes with a single directed edge connected to the main graph. As a result, it is easy to figure out the error nodes.

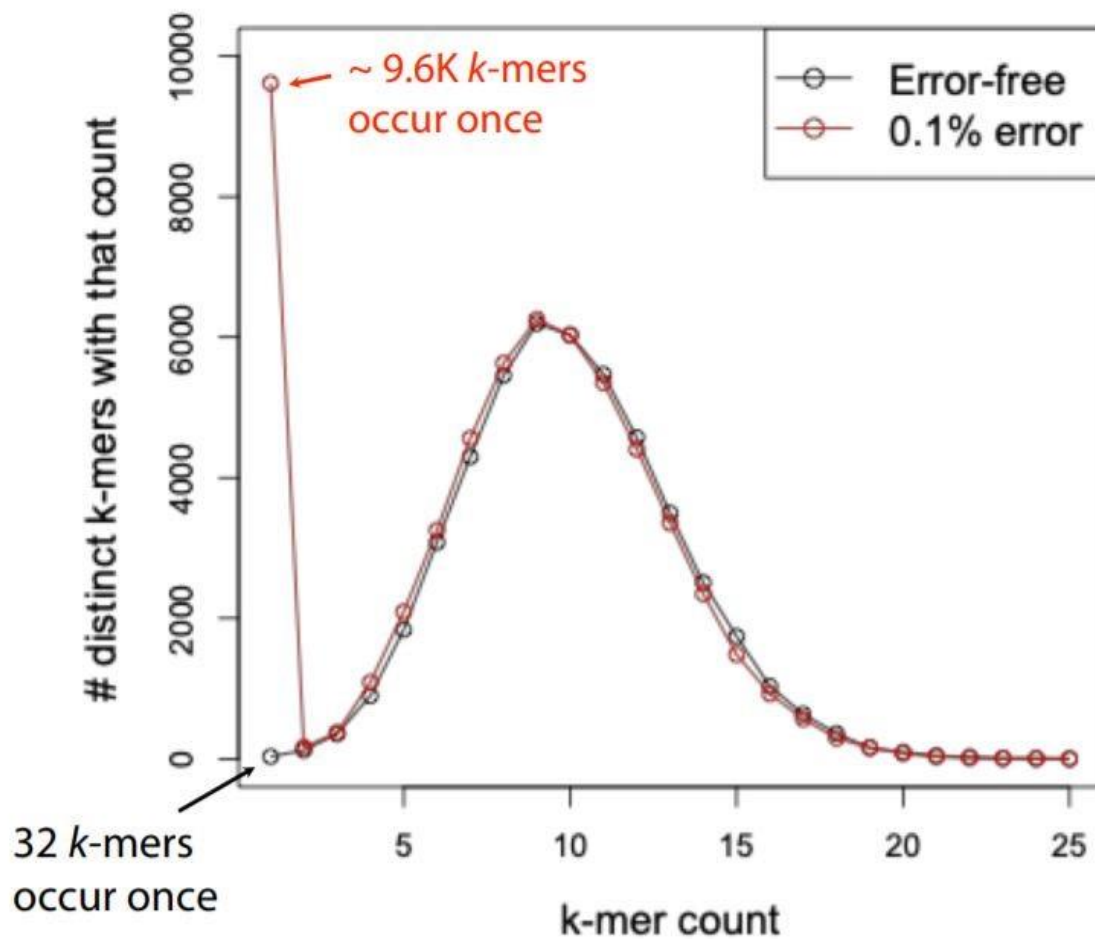


Fig. 3.17: A graph comparing the number of nodes appear between an error-free sequencing and one with 0.1% error. There are more distinct nodes detected in 0.1% error sequencing.

To tackle the problem, the error nodes will be compared with its neighbour nodes. If the number of its neighbour nodes are normal, the error nodes will be shifted to that correct nodes in order to reduce the number of errors and increase the reliability of the graph.

The second problem is that this graph cannot resolve the repeated pattern in the original genome, as same as the OLC Assembly. As the genome will be constructed through the Eulerian path, there may be two or more possible solutions that satisfy the requirement.

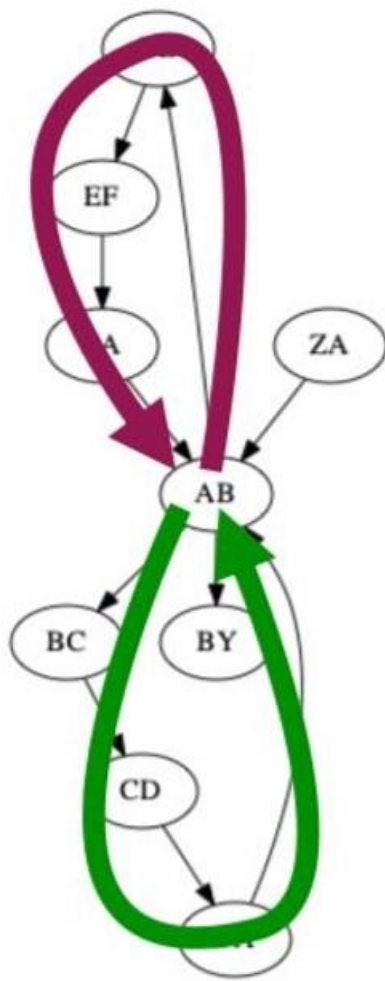


Fig. 3.18: There are two possible solutions of this De Bruijn graph. It does not matter whether the purple path goes first or the green path goes first.

Both the assembly algorithms cannot resolve the repeated patterns due to the problems discussed. The only solution is to increase the read length of the readings in order to cover the repeated regions so the number of unresolvable regions can be reduced.

3.4 Conclusion

This chapter has introduced many useful tools and algorithms to assist the analysis of DNA sequence. These techniques and algorithms are efficient and widely used by other scientists while doing DNA sequencing. Finding a suitable tool is always the first step of efficient analysis.

Chapter 4. Genome Assembler and E.coli Dataset

In this chapter, three different genome assemblers will be introduced, which are used to assembly a dataset of Paired-end sequencing (2x100 base) of E. coli library. They are ABySS, Ray and Edena.

4.1 Genome Assembler

Three different genome assemblers are chosen to compare the performance of assembling pair-end sequencing dataset.

Assembler	Programming Language	Algorithm	Input reads
ABySS[6][7]	C++	De Bruijn graph (DBG)	Paired-end and single-end
Ray[8]	C++	Hybrid	Paired-end and single-end
Edena[9]	C++	Overlap/layout/consensus (OLC)	Paired-end and single-end

From the above table, the three chosen de novo assemblers are ABySS, Ray and Edena. They are chosen because they use different algorithm for genome assembly, ABySS using De Bruijn graph (DBG), Edena using OLC and Ray using hybrid algorithm. They will be used to assemble paired-end dataset in order to show their performance during the assembly using the following evaluation.

1. Efficiency Evaluation

All the assemblers are installed in bal machine which is provided by the Bioinformatics Department. The machine has 48 cores, 1.5TB Ram and 128 GB hard disk space for genome assembly and the assemblers will use twenty four threads doing the assembly. The time used for the whole process will be recorded and compared.

2. Accuracy evaluation

After the completion of assembly, assemblers will provide the statistics about the contigs generated, including the total number of contigs, total length and the N50 of the contigs. This information can be used to justify the performance of the assemblers. More detailed information will be discussed in Chapter 5, comparing the results of different assemblers.

4.2 E.coli dataset

The E.coli dataset used in the comparison of assemblers used is *Escherichia coli* str. K-12 substr. MG1655, which can be freely downloaded from the official website of European Bioinformatics Institute[10]. The dataset is submitted by Illumina Cambridge Ltd. using illumine genome analyser IIx, which fragment genomic DNA randomly using nebulisation and a ~600bp fraction using gel electrophoresis[10].

The dataset is Paired-end sequencing (2x100 base) of *E. coli* library and contains two files, each refers to one end of the paired-end DNA molecule. The total number of reads is 45440200 and the average read length is 503bp with a standard deviation 34.

The assemblers will use the same dataset for assembly and generate the assembled contigs for comparison, using the total number of contigs and the N50 of them.

Chapter 5. Schedule and Milestone

This chapter will discuss about the result computed after the completion of assembly and the comparison of performance between different assemblers.

5.1 Result

5.1.1 Efficiency Evaluation

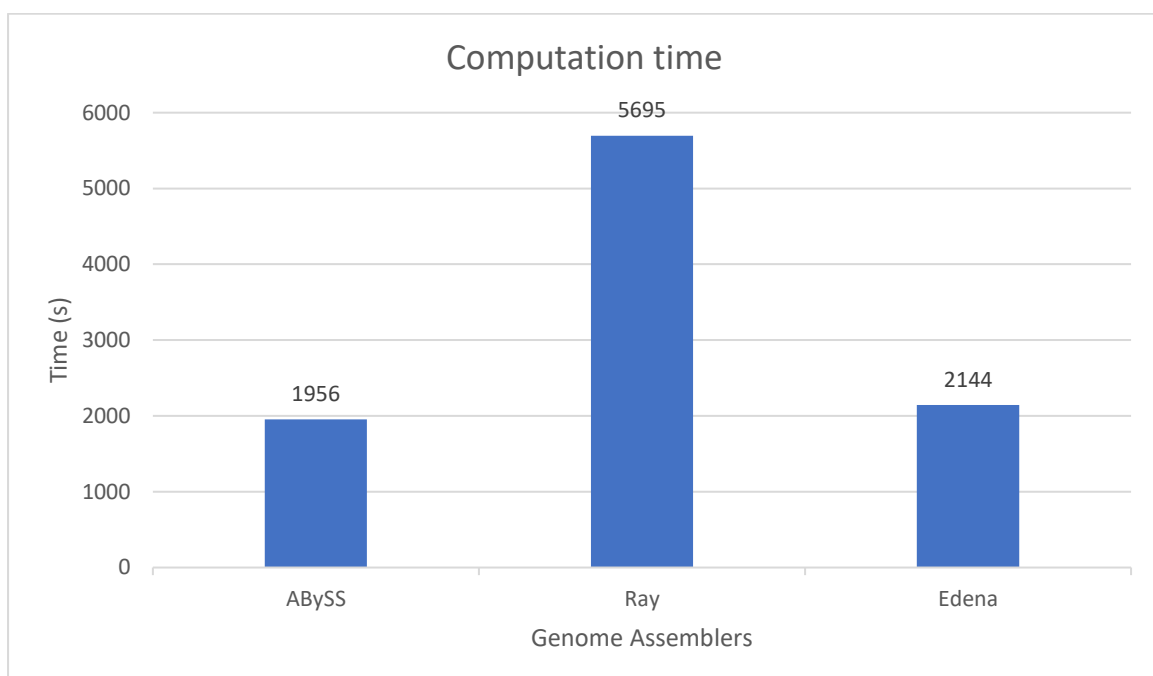


Fig. 5.1: The computation time for each assemblers to finish assembling the input dataset of pair-end E.coli library. ABySS takes 1956 seconds, Ray takes 5696 seconds and Edena takes 2144 seconds.

5.1.2 Accuracy Evaluation

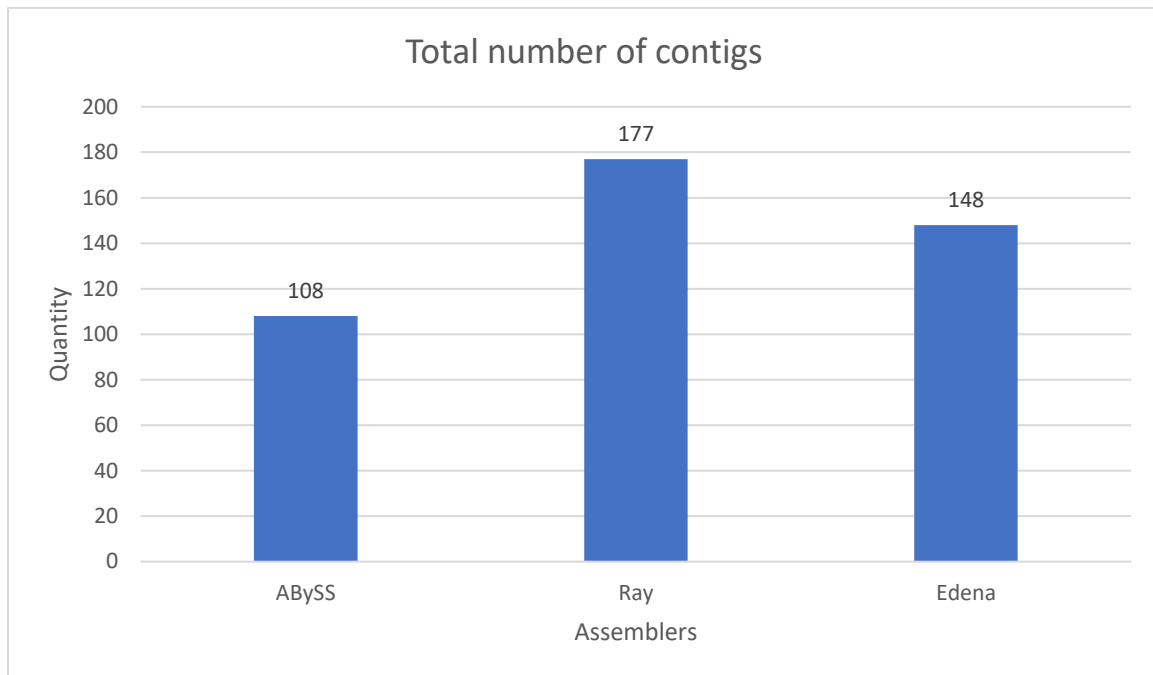


Fig. 5.2: The total number of contigs generated after the assembly process. ABySS has 108 contigs, Ray has 177 contigs and Edena has 148 contigs.

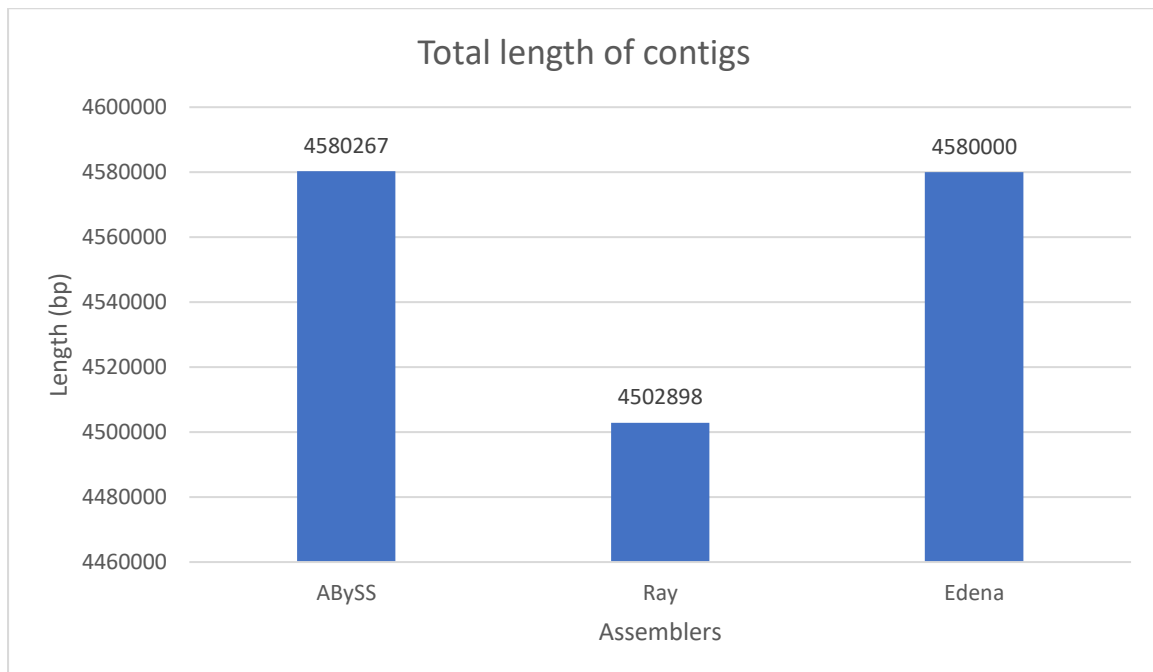


Fig. 5.3: The total length of contigs generated after the assembly process. ABySS produces contigs with total length ~4.58Mbp, Ray produces contigs with total length ~4.50Mbp and Edena produces contigs with total length ~4.58Mbp.

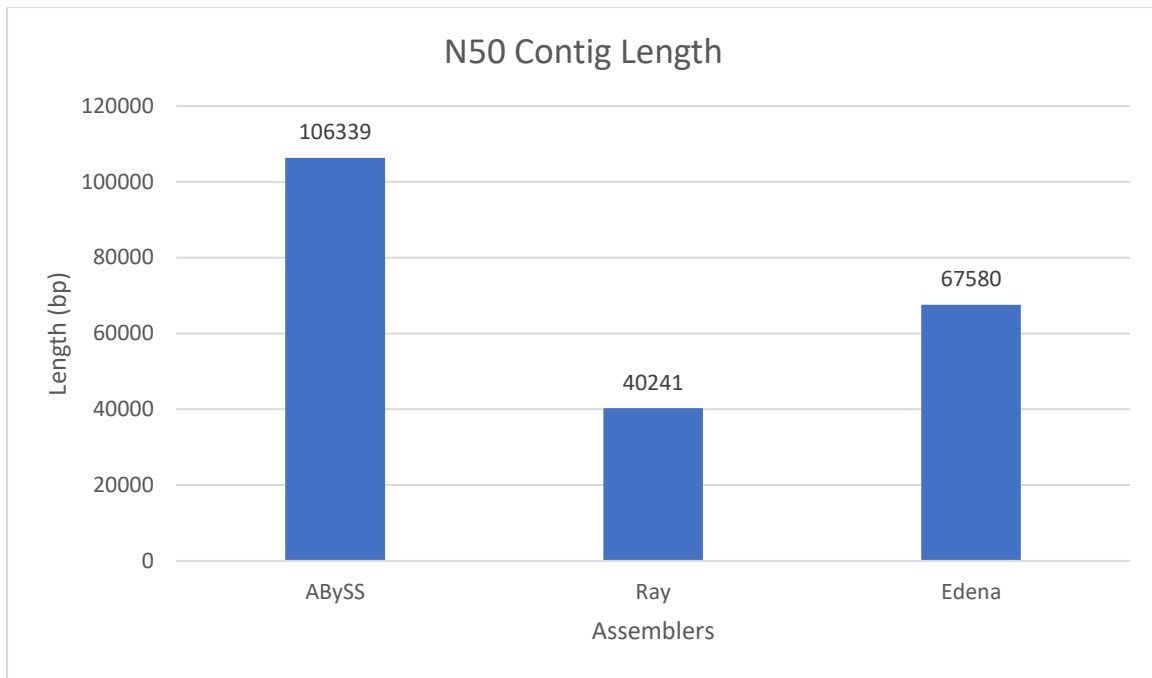


Fig. 5.4: The N50 length of contigs generated after the assembly process. ABySS produces contigs with N50 ~106Kbp, Ray produces contigs with N50 ~40.2Kbp and Edena produces contigs with N50 ~67.6Kbp.

5.2 Comparison

5.2.1 Efficiency

From fig. 5.1, it shows that ABySS and Edena consume similar time to assemble the given dataset of genome, with about 2000 seconds for assembly. However, Ray consumes much more time than the other two assemblies, with a total time 5695 seconds for the assembly, which is almost twice the time of the other two assemblers. The data shows that Ray is the least efficient among the three assemblers and ABySS and Edena have similar performance.

5.2.2 Accuracy

From fig. 5.2, it shows that Ray generates the most contigs compared to the other two assemblers, with a total number of 177. The second high is Edena, which has 148 and ABySS has the least, with a total number of 108.

From fig. 5.3, it shows that three assemblers generate similar total length of contigs, with ~4.58 Mbp, although Ray generates ~4.50Mbp total length of contigs, which is slightly less than the other two assemblers.

From fig.5.4, it shows that ABySS has the highest N50 contig length with ~106 Kbp, while Ray has the lowest N50 contig length with ~40 Kbp. Edena has the second high N50 contig length with ~67.6 Kbp.

N50 is a measurement that the contigs are sorted from longest to shortest. Then, add the lengths of contigs starting from the longest and stop when the summed length of contigs is larger than 50% of the total length, and the value of N50 is the length of contig where the calculation stops. N50 is an important measurement to justify the performance of assemblers in terms of contiguity. The higher of the value N50 means the longer contig length can be generated by assembler and the assembly process can be more accurate. However, to make the comparison of N50 be meaningful, the assembly size of the genome of different assemblers should be the same. This is not a problem in this project as the same set of dataset is used during the assembly process.

Combining the three results, Ray produces the highest number of contigs but the lowest N50 length. ABySS has the lowest number of contigs but the highest N50 length. Edena has the average of both measurements.

Ray produces high number of contigs, which mean it is more conservatively than the other two assemblers while merging smaller contigs to larger contigs. However, it has the lowest N50 length which shows that the performance of Ray is not ideal while comparing to ABySS and Edena. The reason may be the hybrid algorithm of assembling the genome consumes too much time for generating intermediate contigs. In general, ABySS performs the best among the three assemblers. Although it produces the least number of contigs, it has the highest N50 length which is twice of that of Ray. It means that the ability of ABySS assembling short sequences into large contigs is much better than the other two assemblers. Edena performs

moderate among the three assemblers with an average number of contigs and average length of N50.

5.3 Conclusion

Combining the evaluation of efficiency and accuracy, ABySS perform the best among the three assemblers. It consumes the lowest time for assembly and generates the highest N50 value compared to the other two assemblers. Ray performs the worst among the three assemblers. It consumes lots of time for assembly, which is almost twice as that of ABySS, and generate contigs with low N50 length. Edena performs moderate when compared to the other two assemblers. It is as efficient as ABySS, but the result of N50 is not as good as ABySS's. Therefore, it performs worse than ABySS

Chapter 6. Conclusion

In conclusion, while assembling the paired-end genome library of E.coli, ABySS performs the best among the three assemblers. It consumes the least of time to generate the highest N50 length, which means it is highly efficient when compared to other two assemblers. Ray performs the worst among the three assemblers. It consumes the most of time to generate the lowest N50 length, which means it is not efficient as the other two assemblers. ABySS can be the best choice when assembling the paired-end genome library.

Chapter 7. Reference

- [1] *Science Focus*. Retrieved from: <https://www.sciencefocus.com/the-human-body/how-long-is-your-dna/>
- [2] *Your Genome*. Retrieved from: <https://www.yourgenome.org/stories/the-discovery-of-dna>, 2018, February 26.
- [3] *National Human Genome Research Institute*. Retrieved from: <https://www.genome.gov/12011238/an-overview-of-the-human-genome-project/>, (2016, May 11).
- [4] *Ian Murnaghan BSc*. Retrieved from: <http://www.exploredna.co.uk/the-importance-dna.html>, (2018, April 28)
- [5] Neil C. Jones and Pavel A. Pevzner, *An Introduction to Bioinformatics Algorithms*, Cambridge, Massachusetts, 2004
- [6] *ABYSS 2.0*, Shaun D Jackman, Benjamin P Vandervalk, Hamid Mohamadi, Justin Chu, Sarah Yeo, S Austin Hammond, Golnaz Jahesh, Hamza Khan, Lauren Coombe, René L Warren, and Inanc Birol (2017). **ABYSS 2.0: Resource-efficient assembly of large genomes using a Bloom filter**. *Genome research*, 27(5), 768-777. doi:10.1101/gr.214346.116
- [7] *ABYSS*, Simpson, Jared T., Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven JM Jones, and Inanc Birol (2009). *ABYSS: a parallel assembler for short read sequence data*. *Genome research*, 19(6), 1117-1123. doi:10.1101/gr.089532.108
- [8] *Ray Meta: scalable de novo metagenome assembly and profiling*. Sébastien Boisvert, Frédéric Raymond, Élénie Godzaridis, François Laviolette and Jacques Corbeil. *Genome Biology (BioMed Central Ltd)*. 13:R122, Published: 22 December 2012
- [9] *De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer*. Hernandez D, François P, Farinelli L, Osterås M, Schrenzel J. *Genome Research*. 18:802-809, 2008.
- [10] *European Bioinformatics Institute*. Retrieved from: <https://www.ebi.ac.uk/ena/data/view/ERX008638>
- [11] *Initial sequencing and analysis of the human genome*, Lander et al, 2001