



January 2014

A Novel Assembly Algorithm That Optimizes For RNA-Seq Data

Yi Yang

[How does access to this work benefit you? Let us know!](#)

Follow this and additional works at: <https://commons.und.edu/theses>

Recommended Citation

Yang, Yi, "A Novel Assembly Algorithm That Optimizes For RNA-Seq Data" (2014). *Theses and Dissertations*. 1610.

<https://commons.und.edu/theses/1610>

This Thesis is brought to you for free and open access by the Theses, Dissertations, and Senior Projects at UND Scholarly Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of UND Scholarly Commons. For more information, please contact und.common@library.und.edu.

A NOVEL ASSEMBLY ALGORITHM THAT OPTIMIZES FOR RNA-SEQ DATA

by

Yi Yang

Bachelor of Computer Science, Emporia State University, 2009

A Thesis

Submitted to Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

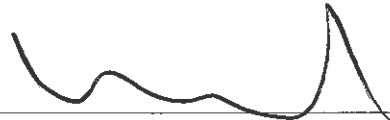
for degree of

Master of Science


Grand Forks, North Dakota

May 2014

This thesis, submitted by Yi Yang in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done, and is hereby approved.



Dr Ronald Marsh



Dr. Emanuel Grant



Dr. Ke Zhang

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the Graduate School at the University of North Dakota and is hereby approved.



Dr. Wayne Swisher
Dean of Graduate School



Date

PERMISSION

Title	A NOVEL ASSEMBLY ALGORITHM THAT OPTIMIZES FOR RNA-SEQ DATA
Department	Computer Science
Degree	Master of Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the Chairperson of the department or the dean of the Graduate School. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use that may be made of any material in my thesis.

Yi Yang

April 22, 2014

Table of Contents

LIST OF FIGURES.....	vii
LIST OF TABLES	ix
ACKNOWLEDGEMENTS	x
ABSTRACT	xii
CHAPTER	
I. INTRODUCTION	1
1.1 Introduction	1
1.2 Background	2
1.2.1 DNA Sequencing	2
1.2.2 Next-Generation Sequencing	3
1.2.3 RNA-Seq and Application	4
1.3 Thesis Organization	5
II. ASSEMBLY	6
2.1 Assembly Methods	6
2.2 De Novo Assembly Algorithms	6

2.2.1	Overlap Graph-Based Assembly Algorithm.....	7
2.2.2	de Bruijn Graph-Based Algorithm.....	8
2.2.3	Comparing de Bruijn Graph with Overlap Graph.....	10
2.3	Different Methods Based on de Bruijn Graph-Based Algorithm.....	10
2.4	Motivation	11
2.5	Objective	12
III.	METHODS	15
3.1	Method Overview	15
3.2	Clustering	16
3.2.1	Hash Table and Hash Map	17
3.3	Clustering Process	24
3.4	Group Merging	28
3.5	Alignment.....	30
IV.	RESULTS.....	32
4.1	Testing Data	32
4.1.1	ERCC Data.....	32
4.1.2	Human Chromosome 22 Simulation Data	36
4.1.3	Human Chromosome 22 Real Data	39
V.	DISCUSSION.....	44

VI. CONCLUSIONS.....	47
REFERENCES	49

LIST OF FIGURES

Figure	Page
1. Example of overlap graph method.	7
2. Diagram of using de Bruijn graph for DNA sequence alignment.....	9
3. Example of Chimeric Edge in a de Bruijn graph.....	11
4. Comparison of Chimeric Edges with different k-mer size.....	12
5. Cluster-based algorithm can be used to avoid Chimeric Edge.....	13
6. The de Bruijn algorithm Chimeric Edge.....	13
7. CBA process and Oases process.....	16
8. Relationship between the hash table and the hash map.....	18
9. Structure smodel.....	19
10. Structure mg.....	19
11. Construction of the hash map: no collision.....	20
12. Construction of the hash map: collision.....	21
13. MurmurHash function.....	22
14. The way to choose a k-mer's hash value.....	23
15. Structure glink and klink.....	25
16. Statistic result of number of sequences based on different cut-off values.....	26
17. The method to cluster SL short reads.....	27
18. The method to link two groups together.....	28

19. Structure halfc.....	29
20. Structure gp_rcd.....	29
21. Example of merging two groups.....	30
22. Alignment process.....	31
23. OpenMP barrier.....	31
24. ERCC-BGI data test results.....	34
25. Human chromosome 22 simulation data test results.....	37
26. Human chromosome 22 real data test results.....	40

LIST OF TABLES

Table	Page
1. Running Time Comparison of all Assembly Methods	43
2. Comparison of MurmurHash Function with Others in Speed and Collision Field	45

ACKNOWLEDGEMENTS

A major research project is never the work of anyone alone. The knowledge and effort of different people have made it possible. I would like to thank all those who worked with me on my research.

I would like to thank God for favoring me with the wisdom and courage during this research and, indeed, throughout my life: “I can do everything through Him who gives me strength” (Philippians 4:13, *New International Version*).

I would especially like to thank Dr. Ke Zhang, my supervisor and advisor, for his help in bioinformatics’ training and economic support throughout my graduate program. Over the past four years, I have learned many abilities from him, including research skills, writing skills, and presentation skills. He always encouraged me to finish the research even when I felt desperation. He also cares about my life since I am alone in the United States.

I would also like to thank all the members in the Department of Computer Science for teaching me technology and giving me the opportunity to improve my programming skills. I want to thank Dr. Emanuel Grant for taking care of my life. Every time when I talk with him about my troubles, he is like my father and helps me with his best advice. I want to thank our department chairman, Dr. Ronald Marsh—also the chair of my graduate committee—for his encouragement, insightful comments, and hard questions. I thank Nancy Rice, secretary of our department, for helping a great deal with my paperwork.

I thank the members working with me in Dr. Ke Zhang's lab: Kaitlin Clarke and Brent Weichel. We work together and help each other. Also, we are good friends outside of the lab.

Last but not least, I would like to thank my family: my parents, Yu-an Yang and Hui-fen Shen, for giving me life and supporting me spiritually in my life. I thank my girlfriend, Yi Liu, for her love.

ABSTRACT

The advent of next-generation sequencing (NGS) technology has shown unprecedented promise for accurately identifying and quantifying genomic variants for living organisms. For species whose genome sequences are unknown, the first step of RNA sequencing data analysis is to assemble all short reads. The de Bruijn graph-based algorithms, such as Oases, are usually used for short reads assembly to resolve the issue of computational complexity. However, de Bruijn graph-based assemblers normally generate high error rates when assembling RNA-Seq data. We have developed a novel assembly algorithm that can be used jointly with any other assembly methods for RNA-Seq short reads. The proposed method, clustering-based assembly (CBA), aims not only to maintain computational and memory efficiency but also improve the assembly accuracy in our simulation study. We tested CBA using ERCC RNA-Seq data, simulated data from Chromosome 22, and real human RNA-Seq data. The results showed that our algorithm was more accurate in comparison with other de novo methods in terms of short reads mapping rate, recover rate, and contigs mapping rate.

CHAPTER I – INTRODUCTION

1.1 Introduction

A major challenge in the new era of genomics research is to develop efficient bioinformatics tools to cope with rapidly growing biological data. In bioinformatics, DNA sequence assembly refers to aligning and merging fragments of shorter DNA sequences to reconstruct the original sequence. This is necessary, as DNA sequencing technology cannot read whole genomes at one time but instead reads small pieces of between 20 and 1,000 bases, depending on the technology used. Sequence assembly of next-generation sequencing data is such a computational intensive step that sometimes requires months of computation time using a mid-size server computer.

In this thesis, we present the development of a new assembly method called Clustering Based Assembly (CBA). This program is different from conventional assembly programs in that it clusters the short reads first based on genomic positions. It divides input fragments (short reads) into pieces of the same length; we call those “k length pieces” or “k-mer.” The program then uses a hash table to indicate all k-mers. CBA clusters short reads to many groups using this hash table. All short reads in the same group will share a number of k-mers with each other. Finally, the program uses the current prominent assembly method to align the short reads to the original sequence for each group. We tested

CBA and other assembly methods with real data and simulation data. CBA improves computational efficiency and assembly accuracy for both.

1.2 Background

1.2.1 DNA Sequencing

In general, DNA sequencing is used for determining the order of the nucleotide bases—adenine (A), guanine (G), cytosine (C), and thymine (T)—in a molecule of DNA. DNA sequencing is one of the most important techniques for molecular biological studies. DNA sequencing technique has been evolving rapidly, providing a powerful approach to understanding the structures of DNA and RNA and their associated biological functions (Turnpenny & Ellard, 2007).

Sequencing technologies have been significantly improved since the first genome was read in 1996. These technologies remain at the core of genomics and have many practical applications. Sequencing technologies are used to determine the genome sequence of a new species or of an individual within a population. A critical stage in de novo genome sequencing is the assembly of shotgun sequences, where DNA fragments are randomly extracted and sequenced.

Recently, the rapid and inexpensive next-generation sequencing NGS methods offer high-throughput gene expression profiling. Today, it is possible to sequence a human's genome in around eight days for approximately \$10,500 (2014 NGS Field Guide – Table 2 – Run time, reads, yields, and costs, 2014) (Shendure & Ji, 2008) (Wu, Zhu, Fu, Niu, & Li, 2011) (“2014 NGS Field Guide, Table 2,” 2014; Shendure & Ji, 2008; Wu, Zhu, Fu, Niu, & Li, 2011).

1.2.2 Next-Generation Sequencing

Next-generation sequencing (NGS) is a new method for DNA sequencing. This technology improves the DNA sequencing process and makes it run faster. The technology uses shotgun sequencing with cyclic-array methods, linking a common adaptor to DNA fragmentation. NGS conducts massive parallel sequencing using an array that includes millions of spatially immobilized PCR colonies. Each colony consists of many copies of a single shotgun library fragment. All array features run in parallel. Finally, the shotgun algorithm uses imaging-based detection and assembles similar fragments; all features run in parallel. Repeating those steps, NGS will build up a contiguous sequencing read for each colony (Costa, Angelini, Feis, & Ciccodicola, 2010; Hoppman-Chaney et al., 2010; Shendure & Ji, 2008) (Shendure & Ji, 2008). In other words, DNA fragmentation is first combined with an adaptor as an array, and then the array transfers those data to colonies and, finally, NGS uses an imaging-based method to assemble those colonies into groups. NGS allows for simultaneously sequencing thousands to billions of sequencing reactions in parallel (Costa et al., 2010, p. X).(Costa, Angelini, Feis, & Ciccodicola, 2010) Because NGS can parallel run arrays and imaging steps, NGS is both fast and cheap.

NGS has been widely used in whole-genome de novo sequencing, ChIP sequencing, RNA-Seq, and so on. Whole-genome sequencing identifies the complete DNA sequence of an organism's genome (Roach et al., 2010).(Roach, et al., 2010) ChIP sequencing is a method used to analyze the relationship or interaction between DNA and protein (Park, 2009). RNA-Seq refers to using NGS to study the transcriptome at the nucleotide level (Faghihi & Wahlestedt, 2009; Marguerat & Bähler, 2010; Zhong Wang, 2009). (Faghihi & Wahlestedt,

2009)

1.2.3 RNA-Seq and Application

In multicellular organisms, almost all cells include the same genes. However, not every gene can express itself in every cell. To find out when and where genes are turned on or off in various types of cells, we will study the transcriptome. By comparing the transcriptomes of different type of cells, we will deeply understand the constitution of the cell and know which gene may respond to a disease in the cell. A transcriptome represents that small percentage of the genetic code that can be transcribed into RNA molecules. Because each gene may produce more than one variant of mRNA, a transcriptome may be very complex (Adams, 2008; Ozsolak & Milos, 2011; Sadava, Hillis, Heller, & Berenbaum, 2012)(Sadava, Hillis, Heller, & Berenbaum, 2012)(Ozsolak & Milos, 2011) (Adams, 2008). For our program, we are using RNA-Seq. As we previously showed, RNA-Seq is a sequencing technology to study the transcriptome at the nucleotide level. It is used to discover the gene expression level. By mapping the RNA-Seq reads onto the exons¹ of the known genome, we will find out the total number of mapped reads. In doing so, we can get the gene expression level, which is represented as Fragments per Kilobase of transcript per Million mapped reads (FPKM) (Manteniotis S, 2013; Wang, Gerstein, & Snyder, 2009). (Wang, Gerstein, & Snyder, 2009) (Manteniotis S, 2013)

¹ The corresponding sequence in RNA transcripts.

1.3 Thesis Organization

The rest of this thesis is organized as follows:

- Chapter II – Assembly. This chapter describes different assembly methods and points out the disadvantages of de Bruijn algorithm. At the end of the chapter, we show our motivation and the basic idea behind the Clustering Based Assembly (CBA) method.
- Chapter III – Method. This chapter describes how we developed the CBA methods and tools, including the data structure, algorithm, and memory control.
- Chapter IV – Results. In this chapter, we use ERCC real data, Chromosome 22 real data, and Chromosome 22 simulation data to test different assembly methods. This chapter also discusses CBA results and compares them with other methods.
- Chapter V – Discussion. This chapter evaluates the CBA method and discusses the advantages and disadvantages of CBA assembly methods. Finally, the chapter lists possible future work.
- Conclusion. This chapter is a summary statement. It states CBA's good performance in terms of accuracy and speed. This chapter further explains why CBA would be useful in real RNA-Seq application.

CHAPTER II – ASSEMBLY

2.1 Assembly Methods

There are two common approaches for DNA assembly: reference-based assembly and de novo assembly. Reference-based assembly is also known as the “genome-guide assembly” method. This method first aligns short reads to the reference genome and then assembles overlapped alignments into transcripts. The de novo assembly method does not rely on the reference genome; it is used to reconstruct the nucleotide sequence. When the reference genome exists, researchers normally use the reference-based assembly method. In the reference-based assembly process, the transcriptome is analyzed by mapping on the reference genome. In the absence of a reference genome, the de novo assembly will be considered. De novo transcriptome assembly is the method of creating a transcriptome without the aid of a reference genome.

2.2 De Novo Assembly Algorithms

The de novo assembly method has two different algorithms: overlap graph-based assembly and de Bruijn graph-based assembly. The major difference between the two algorithms is how they construct the topology. The overlap graph-based assembly algorithm uses short reads to build the topology, but the de Bruijn graph-based assembly algorithm instead uses k-mers.

2.2.1 Overlap Graph-Based Assembly Algorithm

The overlap graph-based assembly algorithm is the traditional assembly approach. This algorithm has three phases: overlap, layout, and consensus. The overlap phase is the process that finds the overlaps between reads. Those overlaps capture all possible relationships between the fragment reads. The layout phase then orders fragment reads by those overlaps. Finally, the consensus phase aligns fragment reads to contigs. Overlap graph-based algorithm compares each pair of short reads. If one short read's tail matches another short read's head, the program will align them together.



Figure 1. Example of overlap graph method.

In Figure 1, the strings "ATCCCGAATGCA" and "AATGCAAACGTT" will align into the string "ATCCCGAATGCAAACGTT." The strings "ACCTGATTAGCC" and "CTGTGATTACAT" will be thought as coming from different genes because they share the

substring “TGATTA” which is neither the head of one string nor the tail of the other string. Mira is the type of software that uses overlap graph-based algorithm (Chevreux B, 2004). The major disadvantage of overlap graph-based algorithm is its slow computational speed.

2.2.2 de Bruijn Graph-Based Algorithm

The first idea for the de Bruijn graph-based algorithm came from Nicolas Govert de Bruijn (1946).(de Bruijn, 1946) He designed his signature de Bruijn sequences. In 2001, Pevzner, Tang, and Waterman developed the de Bruijn graph-based algorithm (Pevzner, Tang, & Waterman, 2001). The de Bruijn graph-based method compares each k-mer² instead of short reads. If two k-mers overlap, k-1 length nucleotides then align the k-mers. For example, k-mers “ATGGTC” and “TGGTCT” can be aligned to “ATGGTCT,” but k-mers “ATGGTC” and “GGTCAA” will not be aligned because they share only k-2 length nucleotides.

The de Bruijn graph is defined as:

Set V = All length-k subfragments (k-mer)

E = Directed edges between consecutive subfragments

D_k = de Bruijn graph, nodes overlap by k-1 words

Then exist

$D_k = (V,E)$

² A nucleotide sequence whose length is k.

The de Bruijn graph-based algorithm can be used to assemble RNA sequences (see

Figure 2).

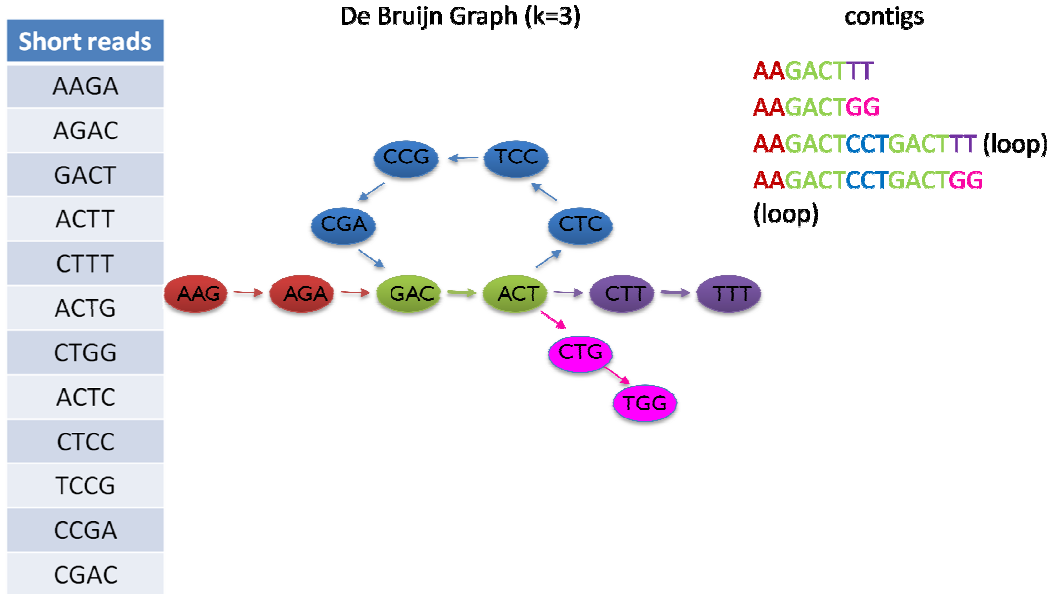


Figure 2. Diagram of using de Bruijn graph for DNA sequence alignment.

Figure 2 presents a simple de Bruijn graph for RNA-Seq assembly. All short reads have four nucleotides. $K = 3$ means all subfragments have three nucleotides. In the figure, there are one loop and two tips. The program will generate four kinds of different contigs: (a) red \rightarrow green \rightarrow purple, (b) red \rightarrow green \rightarrow pink, (c) red \rightarrow green \rightarrow blue \rightarrow green \rightarrow purple (loop), and (d) red \rightarrow green \rightarrow blue \rightarrow green \rightarrow pink (loop). Because this de Bruijn graph exists as a loop, the length of two looped contigs can be unlimited. This represents a deficiency of de Bruijn graph when dealing with repetitive sequences; all de Bruijn graph-based algorithms should have a loop detection process to avoid logical errors.

2.2.3 Comparing de Bruijn Graph with Overlap Graph

The overlap graph uses short read-to-short read comparison. The algorithm must compare short reads one by one, and also it needs to delete the short reads that overlap to the aligned contigs. Therefore, the overlap graph-based method is time consuming. The de Bruijn graph-based algorithm uses k-mer to k-mer comparison. It does not need to consider the overlap issue (as does the overlap-graph based method) but instead generates a large topology. Therefore, the de Bruijn graph-based algorithm is faster than the overlap graph-based algorithm. However, the de Bruijn graph-based algorithm is not an accurate assembly method; we will discuss this in a future section.

2.3 Different Methods Based on de Bruijn Graph-Based Algorithm

The common assemblers based on the de Bruijn graph include Assembly by Short Sequences (ABYSS), Velvet, Oases, and Trinity. ABYSS is a parallelized sequence or short reads assembler. It has two steps to assembly short reads: First, it splits all short reads into k-mers. Second, it uses a de Bruijn graph to align k-mers and generate contigs (Birol et al., 2009; Simpson et al., 2009). (Birol, et al., 2009) (Simpson, Kim, Jackman, Schein, Jones, & Birol, 2009) Trinity was developed in 2010 by Grabherr and colleagues, and it has three steps: The first they call “Inchworm” because it splits short reads to k-mers and aligns k-mers to contigs. The second, called “Chrysalis,” clusters contigs to pools when they share at least one (k-1)-mers. The third, “Butterfly,” splices pools and generates transcripts (Grabherr MG, 2011). Another assembler is Velvet, which runs in two steps: Velveth and Velvetg. Velveth helps construct the dataset for Velvetg. It takes in a number of sequence files and produces a hash table, and then it outputs two files into an

output directory. The two output files, Sequences and Roadmaps, are necessary to Velvet. Velvet is the core of the Velvet software, where the de Bruijn graph is built and processed (Zerbino & Birney, 2008). The last assembler, Oases, is an updated version of Velvet (Schulz, Zerbino, Vingron, & Birney, 2012).

2.4 Motivation

The Chimeric Edge is the major problem in the de Bruijn graph-based algorithm.

The de Bruijn graph algorithm will not detect those Chimeric Edges.

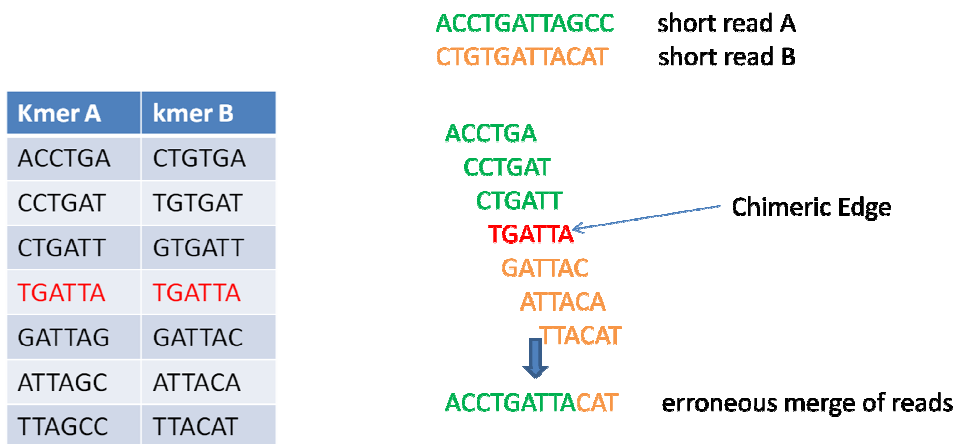


Figure 3. Example of Chimeric Edge in a de Bruijn graph.

Figure 3 shows two short reads, “ACCTGATTAGCC” and “CTGTGATTACAT.” They share k-mer “TGATTA,” also called the Chimeric Edge. When the program runs a de Bruijn graph algorithm, erroneous merge of reads will happen. Short reads are aligned as “ACCTGATTACAT.”

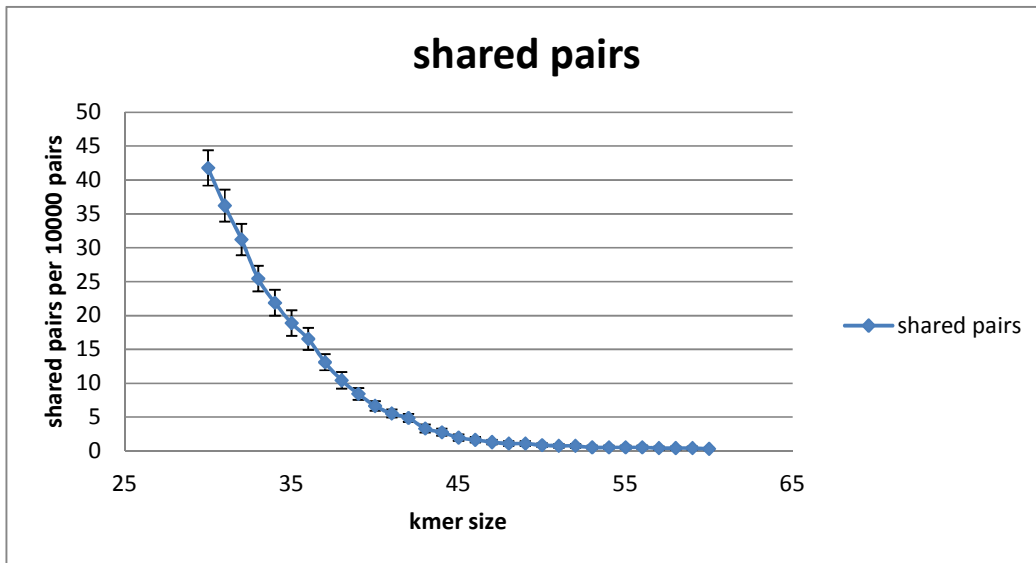


Figure 4. Comparison of Chimeric Edges with different k-mer size.

In Figure 4, we randomly collect 10,000 short reads from human genomes, which have 23 pairs of chromosomes. Since Oases uses 31 as the k-mer size, we also use 31 as the k-mer size to test. The result shows there exists 38 pairs of short reads from different chromosomes that share the same k-mers. That shows the de Bruijn graph-based algorithm has a high error rate.

Since de Bruijn based assemblers, such as ABySS and Velvet, usually have high error rates, we proposed optimizing the sequence assembly to make the assembly more accurate than in the de Bruijn graph-based algorithm.

2.5 Objective

This research presents a method to improve the current de novo assembly method for RNA-Seq. The method of improving the assembly result is by providing a “clustering method” before the sequence assembly.

The Clustering Based Assembly method also uses k-mer to judge the relationship of two short reads, and it also counts the number of k-mers overlapped between two short reads. If two short reads share more than 20 k-mers, it means they may come from the same gene.

If one group has five or more short reads sharing 14 to 20 k-mers with another group's short reads, the clustering process will merge the two groups together (Figure 5).

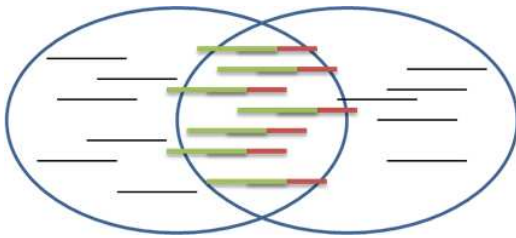


Figure 5. Cluster-based algorithm can be used to avoid Chimeric Edge.

The clustering algorithm is different from the de Bruijn graph algorithm in that the de Bruijn graph aligns two short reads when they share only one k-mer (Figure 6).

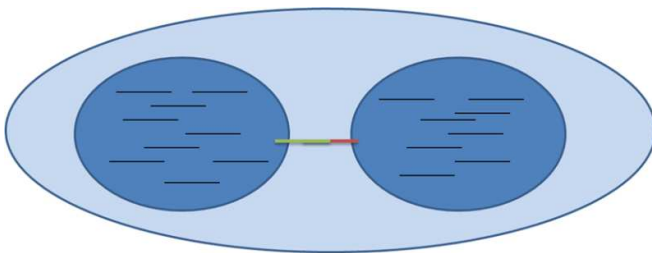


Figure 6. The de Bruijn algorithm Chimeric Edge.

The clustering process gives a more accurate solution for the Chimeric Edge problem, and it will divide different non-relationship genomes into different groups. That is why the clustering process reduces error rate of contigs mapping.

This research was completed by implementing the following three major steps:

1. The program designed a hash table for the grouping method.
2. The program clustered short reads into groups.
3. The program ran Oases parallel for each group and then merged all Oases' results into the final result.

The clustering algorithm plus the sequence assembly method provides a more accurate assembly result when compared to directly using the sequence assembly methods. This model can be collated with all current de novo assembly methods.

CHAPTER III – METHODS

In this chapter, we present the basic ideas of the clustering-based assembly (CBA) method. The program was implemented in the C++ language. CBA has two major steps: clustering and alignment. In this thesis, we focus on how to cluster short reads. We use the current prominent software Oases to align each clustering.

3.1 Method Overview

The CBA algorithm first splits short reads to k-mers and then uses a MurmurHash function (Appleby, 2011) to transfer k-mer into hash value. The program creates a hash table, which size is 2^{32} to store k-mer's hash values. The program creates a hash map, which is mapped to the hash table and stores short reads' index that share the same k-mer. Secondly, based on the hash table and hash map, CBA clusters all short reads into different groups. Then CBA runs Oases in parallel for each group. Finally, CBA uses a single thread to merge all result files generated by Oases into a new contigs file. The CBA process is shown on the right side of Figure 7.

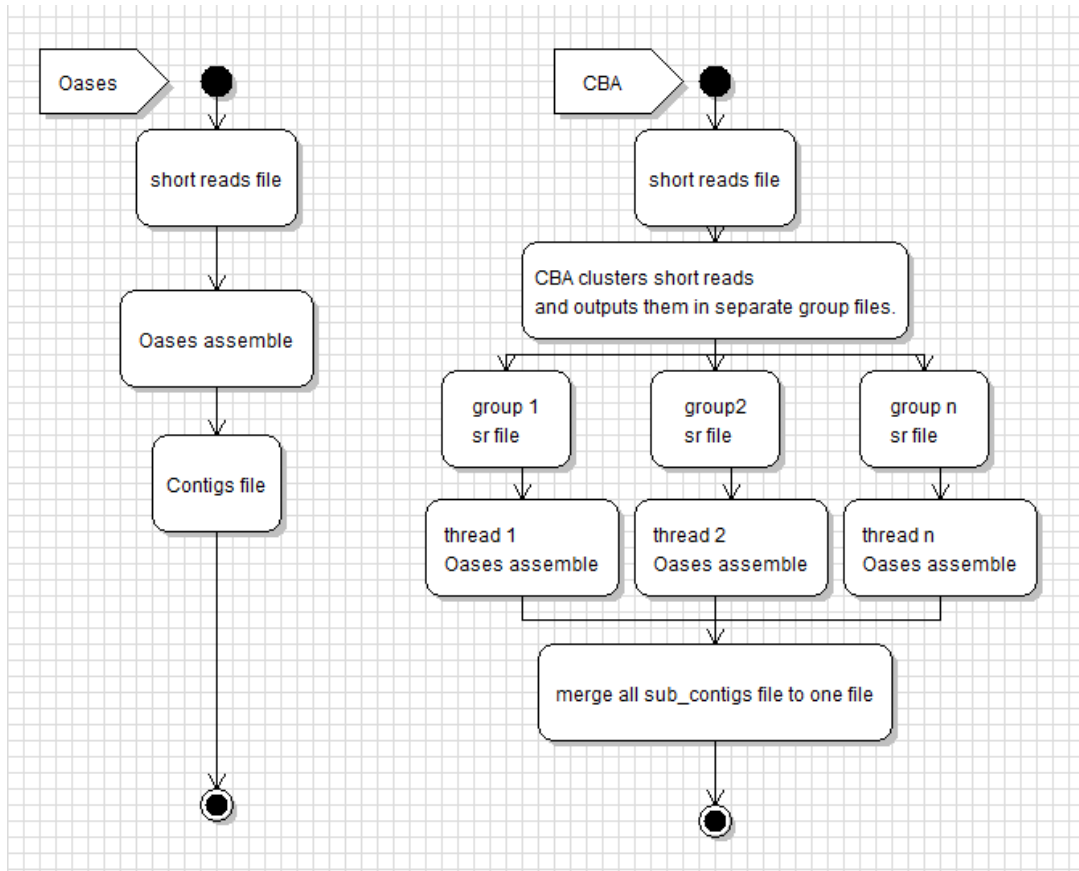


Figure 7. CBA process and Oases process.

3.2 Clustering

Clustering is the core of our program. The idea of clustering is simple: It merges overlapped short reads in one group. However, there are many steps to get to the finished clustering process. The major steps include: hash table, hash map, clustering, and merging groups. The hash table is an array. It stores the indexes of the hash map into cells. Those cells' indexes represent the k-mers' hash value, and the k-mers' hash value are calculated by hash function. The hash map is also an array, with each cell storing short reads that map to a specific k-mer. Because the hash table is a discrete structure, it may have many empty cells. If we directly store all short reads

information in the hash table, those empty cells may cost our system too much memory. That is why the hash map is needed: The hash map is a continuous structure; all data will be stored one by one with no skipping. The clustering process clusters short reads into groups. Finally, the clustering process will merge together unstable groups that have weak linkages.

3.2.1 Hash Table and Hash Map

The hash map stores the information of the short reads that share a specific k-mer. Whenever the program finds a new k-mer, the hash map will assign a new cell for storing all short reads mapping to this k-mer. The hash table stores only integer numbers, which represent the hash map's index. The hash table is used to locate different k-mers. In other words, when the program finds a new k-mer, that k-mer will be transferred into a hash value by the hash function, and the hash map will also create a new cell for the new k-mer. The index of the new cell in the hash map will be stored in a cell of the hash table; that location or index is the hash value. We explain the relationship of the hash table and hash map in Figure 8.

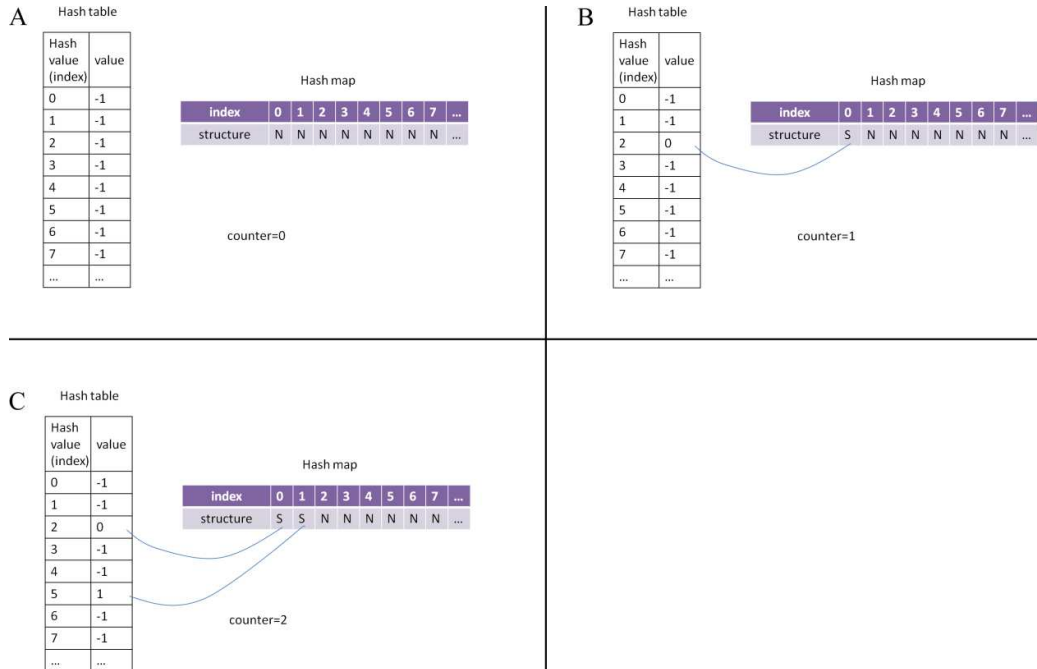


Figure 8. Relationship between the hash table and the hash map.

In Figure 8, we show how the hash table links to the hash map. In Step A, at the beginning, both the hash table and hash map are empty. The program initializes all cell values of hash table to -1 and initializes all cell values of hash map to NULL, setting the variable “counter” to 0. In Step B, the program transfers a k-mer to hash value as “2.” Based on this value, we can locate the hash table cell position, and we then insert the value of “counter” into hash table cell “2,” which is “0.” The program also inserts a structure adding the k-mer into hash map cell, and the cell’s index is the value of “counter.” Finally the program makes the “counter” increase by 1. Right now, “counter” is 1. Finally, our hash table cell “2” has a value “0,” and the hash map cell “0” stores a structure. Step C repeats Step B. After the program runs Step C, the hash table owns two values in cell “2” and cell “5,” the hash map inserts a new structure to cell “1,” and the variable “counter” becomes 2. For memory saving, we transfer a,t,g,c to 00,11,10,01 binary numbers, than transfer binary

numbers to strings (8 binary becomes 1 byte 1 character). The program stores the string in the hash map. For example, short read “ATTC” will transfer to binary bits “00111101,” and then the binary bits are converted to the character “=”.” Finally, the short read string “ATTC” is stored as character “=” into the hash map, saving 75% of memory. We call this process a “zipping string.”

We have two structures for the hash map: smodel and mg:

```
struct smodel {
    mg* srs; //a link list of structure mg, and it stores all short reads mapping to the new k-mer.
    char* value; //stores zipping string of the new k-mer.
    smodel* next; };
```

Figure 9. Structure smodel.

```
struct mg {
    long long int sr_num; //short read index
    mg* next; };
```

Figure 10. Structure mg.

Whenever a new short read comes in, the program will split it into many k-mers. For each k-mer, the program first checks whether the k-mer already exists in the hash table. We can have three situations:

In the first case, as Figure 8 shows, if the k-mer is not stored in the hash map, the program will create a new cell in the hash map. The new cell is initialed as “s structure smodel,” and the program puts the short reads index in the srs variable. As the same time, the program will store the new cell’s index to the hash table.

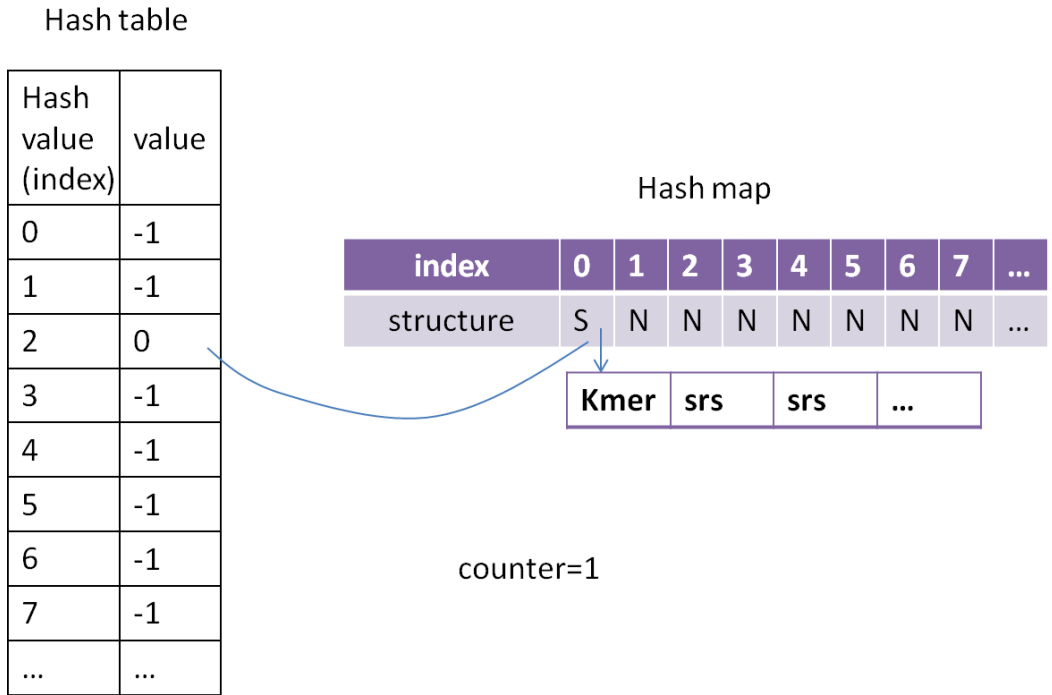


Figure 11. Construction of the hash map: no collision.

The second case is shown in Figure 11, when the k-mer is already stored in the hash map. The program will find the index of the hash map by searching the hash table, and directly goes to the cell of the hash map through the index. The cell already has an smodel structure. The program creates a new mg structure for storing the new short read's index. Finally, the program links the new mg structure to the srs variable in the cell.

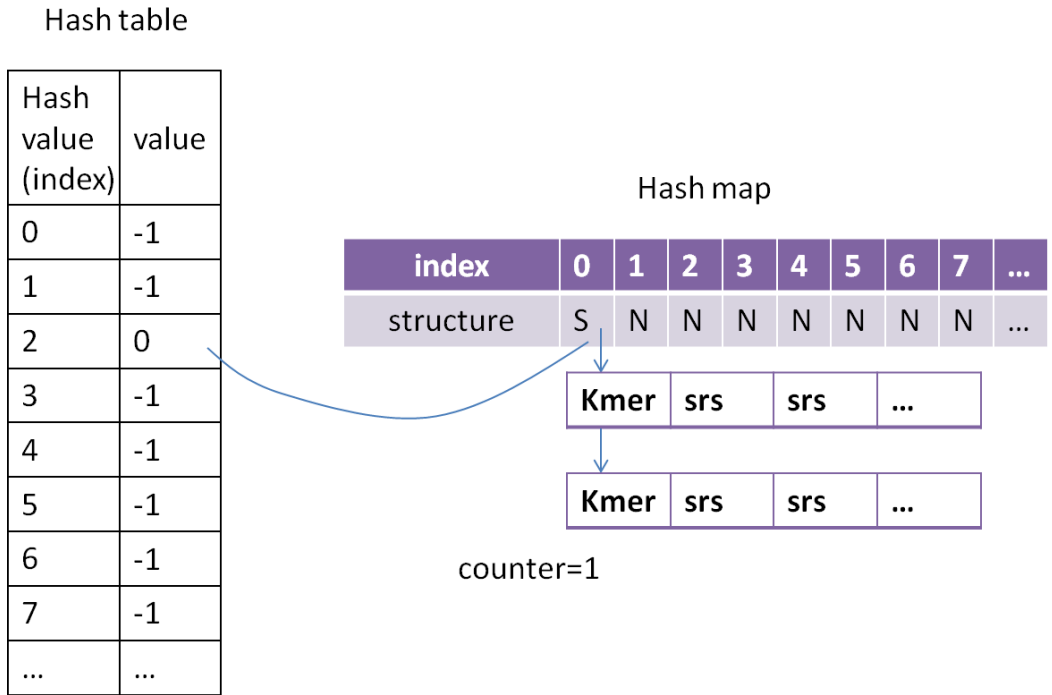


Figure 12. Construction of the hash map: collision.

The third case is shown in Figure 12, if the k-mer is not stored in the hash map but the k-mer’s hash value collides with other k-mer’s hash value. We first find the cell in the hash map as we did in the second case, and then we create a new smodel structure to store the new k-mer and short reads that belong to the k-mer. Finally, we link the new smodel to the cell in the hash map.

3.2.1.1 Memory Usage

The hash table is 2^{32} “long long int” array, it will take 32GB. The hash map takes up a lot of memory because the map stores the smodel structures, and the smodel structure is a two-dimension link list. “Long long int” style costs 8 bytes, and all pointers cost 4 bytes. For one smodel structure, we need at least 24 bytes. If the k-mer size is 45, we may have 4^{45} different k-mers. At most, we may need $4^{45} * 24$ bytes space. That amount of

space is impossible to be held by any machine. In actuality, in typical data size,³ we have approximate 14,000,000 k-mers, and a hash map takes only 200GB.

3.2.2.2 Time Efficiency

The hash table, whose size is 2^{32} , is generated by the MurmurHash function.

```
unsigned int MurmurHashNeutral2 ( const void * key, int len, unsigned int seed ){
    const unsigned int m = 0x5bd1e995;
    const int r = 24;
    unsigned int h = seed ^ len;
    const unsigned char * data = (const unsigned char *)key;
    while(len >= 4){
        unsigned int k;
        k = data[0];
        k |= data[1] << 8;
        k |= data[2] << 16;
        k |= data[3] << 24;
        k *= m;
        k ^= k >> r;
        k *= m;
        h *= m;
        h ^= k;
        data += 4;
        len -= 4;}
    switch(len){
    case 3: h ^= data[2] << 16;
    case 2: h ^= data[1] << 8;
    case 1: h ^= data[0];
        h *= m;};
    h ^= h >> 13;
    h *= m;
    h ^= h >> 15;
    return h;}
```

Figure 13. MurmurHash function.

Because one k-mer has two directions, we choose the higher hash value as the k-mer's hash value.

³ 10,000,000 short reads data.

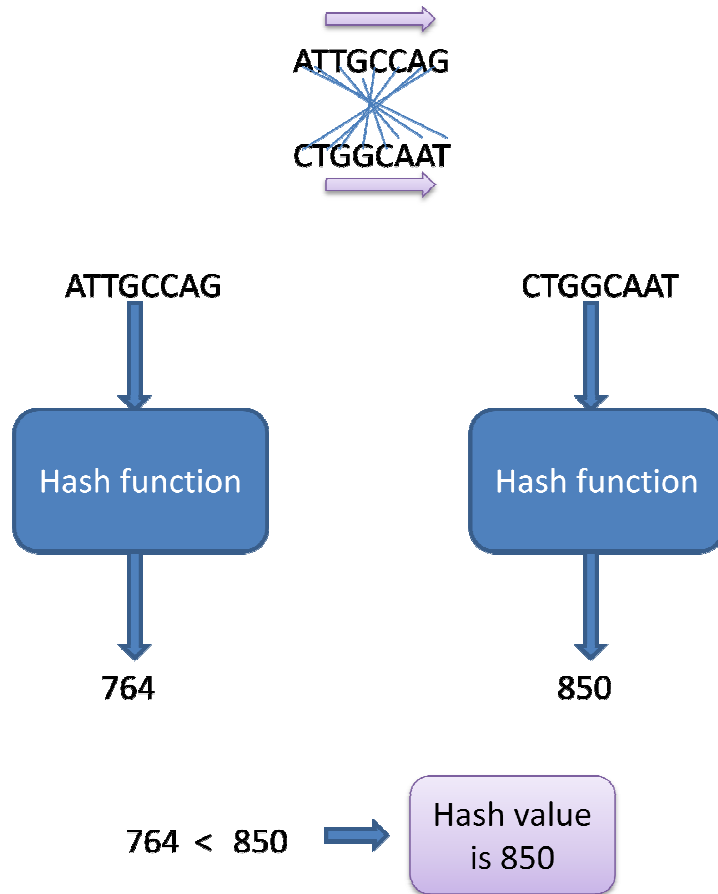


Figure 14. The way to choose a k-mer's hash value.

Figure 14 shows how to choose a k-mer's value. Because most DNA molecules are double-stranded helices, when DNA is transcribed into RNA, the RNA may have two directions. As Figure 14 shows, strings "ATTGCCAG" and "CTGGCAAT" represent the same k-mer from different directions. Because the two strings indicate the same k-mer, we only need one hash value for that k-mer. We run both strings through the hash function, and we may get two hash values. In this case, the string "ATTGCCAG" transfers into value 764, and the string "CTGGCAAT" transfers into value 850. We will choose the greater one as the k-mer's hash value. That means we will calculate hash values twice for each k-mer.

Normally, the search time complexity of the hash table is close to $O(1)$, but it is

really based on how many different input strings via hash function transfer into the same hash value. If two k-mers' hash value have a collision, we have to make a link list to store the two different k-mers. When the program searches a k-mer with a collision hash value in the hash table, it will go to the cell of hash map. Then the program finds the cell that stores the link list. It has to compare the current k-mer with the stored k-mers in the link list, one by one, to find the k-mer's position. In this case, our hash table and hash map's time complexity would be greater than $O(1)$.

3.3 Clustering Process

The clustering process is based on the hash map. We have two tables: The table of "k-mer->short reads" is a link list array and stores all short reads' indexes mapping to each k-mer. The index of the array represent k-mer's index, and each cell has a structure called "glink." The glink has two variables: Variable "seq" stores a short read's index mapping to the k-mer, the other pointer variable points to the next glink structure, which stores other short reads' indexes that map to the same k-mer. The "short read->k-mers" is also a linked list array and stores all k-mers' indexes mapped to each short read. The index of the array represents k-mers' indexes, and each cell has a structure called "klink." There are two variables: The k-mer variable stores the k-mer's index that exists in the short read, and the other pointer variable points to the next "klink" structure, which stores other k-mers' indexes in the same short read.

```
struct glink{
    long long int seq;
    glink *next;};
struct klink{
    long long int k-mer;
    klink *next;};
```

Figure 15. Structure glink and klink.

Based on “short read ->k-mers” array, the program can find all k-mers from a short read. For each k-mer, the program checks the “k-mer->short reads” array to find all other short reads sharing the same k-mer with the current short read. We use three cut-off values to control how the short reads are grouped. The first value is called “ori-cut,” which is the number of k-mers shared between two short reads. If two short reads share a number of k-mers less than the ori-cut value, the two short reads will not be thought to have any relationship by the program. The second value is called “max-cut,” which means if two short reads share a number of k-mers that is over the max-cut value, then the program will put the two short reads in one group. The last value is called “ave-cut”; it is used for merging small groups. If two groups share a number of weak linkage short reads pairs,⁴ and this number is greater than ave-cut, then the program will merge the two small groups together.

⁴ Paired short reads in different groups that share the number of k-mers less than max-cut and greater than ori-cut.

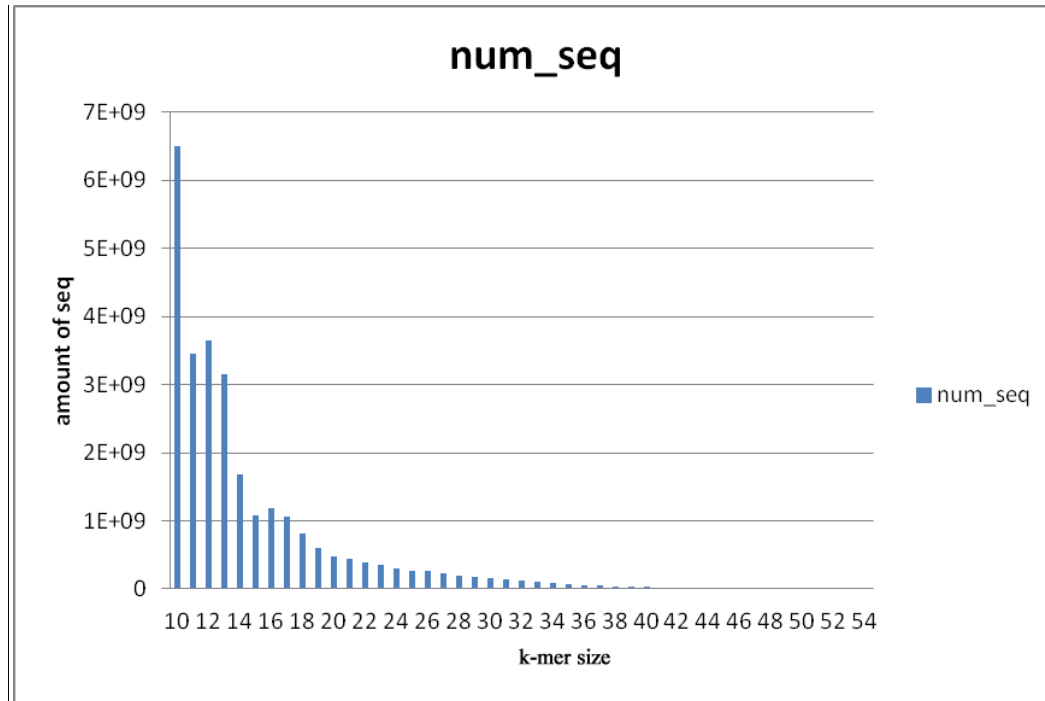


Figure 16. Statistic result of number of sequences based on different cut-off values.

We decide the ori-cut value based on statistic results. In Figure 16, x-axis means how many k-mers short reads are shared, and y-axis means the number of short reads that share the same number of k-mers, and k-mer length is 31. This graph shows that 14 is a magic number because the number of short reads sharing from 12 and 14 k-mers drops quickly. We can use 14 as a cut-off number and discard short reads that shared fewer than 14 k-mers. That also decreases the number of short reads.

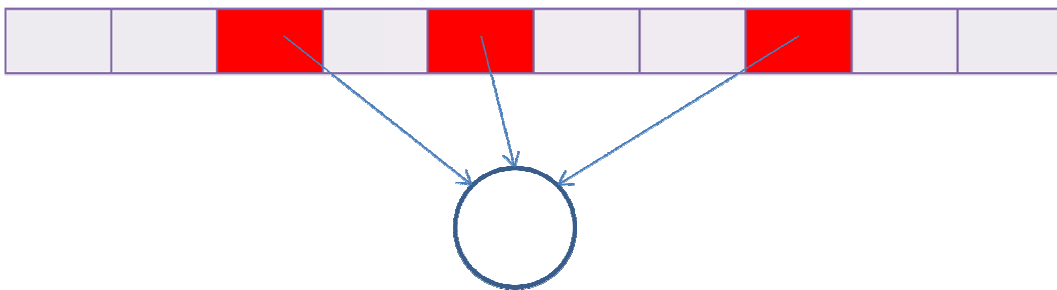
In the grouping process, the program creates a pointer array, called “sr_array.” The length of the array is the number of short reads, and each cell represents a short read.

If the short read that shares a number of k-mers is over the max-cut value, we call those short reads Strong Linked (SL) short reads with each other.

We have two situations in the grouping process:



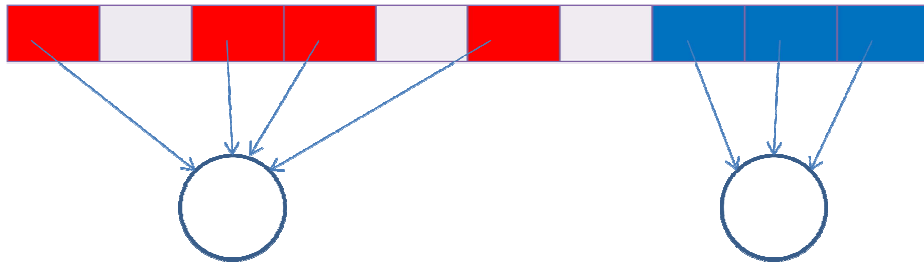
Step1: program detects short reads shared number of k-mers greater than max-cut.



Step2: program links short reads pointer to a new node and considers them a group.

Figure 17. The method to cluster SL short reads.

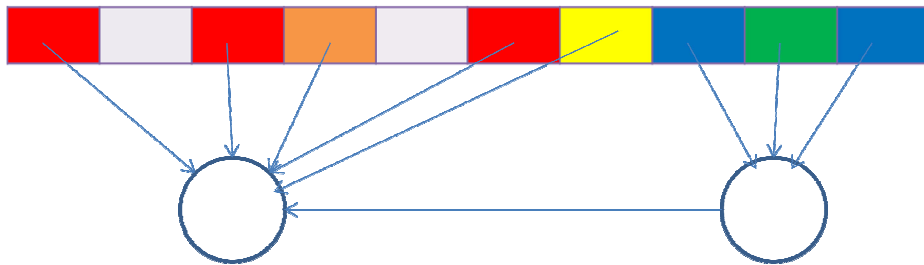
If the program finds some SL short reads linked with each other, then the program links them together to a new node and generates a small tree. In Figure 17, all red cells are SL short reads linked with each other, and then the program will create a new node and link all red cells to the node. In this case, the program considers them one group.



Step1: existing two groups



Step2: program detects another group, one of short reads from the group also belong to the red group, another short read from the group belong to the blue group.



Step3: program links the yellow group short reads to the red group and links the blue group to the red group.

Figure 18. The method to link two groups together.

If there exists different groups, the program will check these groups. One short read from the yellow group is a SL short read linked with the red group. Another short read from the yellow group is a SL short read linked with the blue group. In this case, the program will merge the three groups together (Figure 18).

The program goes through the whole `sr_array` and runs the two cases above. After the grouping process, all short reads are grouped. Some groups are really small—only 1 to 5 short reads. We will regard those small groups as useless and discard them.

3.4 Group Merging

We called paired short reads in different groups that share a number of k-mers less than max-cut and greater than ori-cut as Weak Linkage (WL) pair. The short reads in a WL pair must differ from other WL pairs' short reads. The program stores WL pair in a structure array. We create the structure called "halfc" that includes two variables; the two variables store the WL pair:

```
struct halfc {  
    long long int fstn; //the first short reads number of WL pair  
    long long int sndn ; //the second short reads number of WL pair };
```

Figure 19. Structure halfc.

After the grouping process, we have a structure called "gp_rcd." It offers a mapping between short reads and groups:

```
struct gp_rcd{  
    long long int sr; //shot read index  
    long long int gp; //group index, the short read belong to.};
```

Figure 20. Structure gp_rcd.

Considering some transcripts have a low number of short reads, this means short reads in this transcript may align to many groups with a weak connection. We set a value, "ave-cut," which represents how many WL pairs are shared between two groups. When the number of WL pairs in two groups are greater than ave-cut, we will merge the two groups.

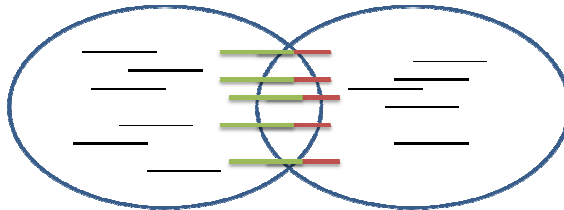
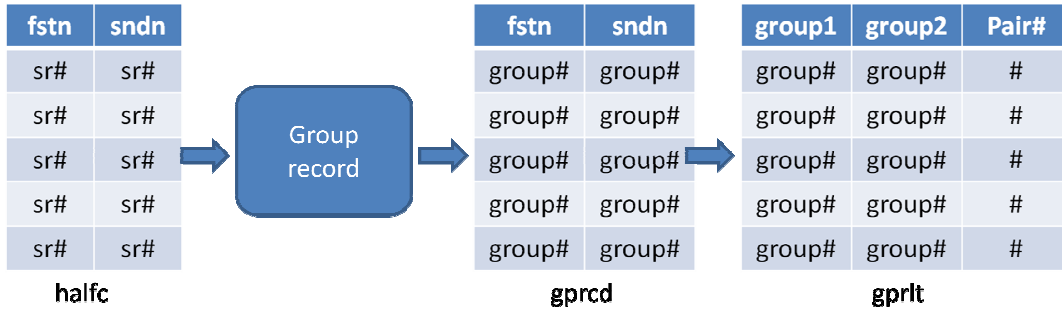


Figure 21. Example of merging two groups.

In Figure 21, all WL pairs are stored in structure halfc. The gp_rcd gives a group number for every short reads in WL pairs. Then the program will calculate the number of WL pairs between the two groups. If the value is greater than ave-cut, we will merge the two group or will keep both groups.

3.5 Alignment

After we get every group, we use OpenMP to run the Oases method for each group in parallel. In Figure 22, for each thread, we create a buffer folder. Each thread handles one group each time and stores the Oases assembly method result file (transcript.fa) in its buffer folder. When the results are ready, we use a single thread to merge all results files into the final contigs file (contigs.fa).

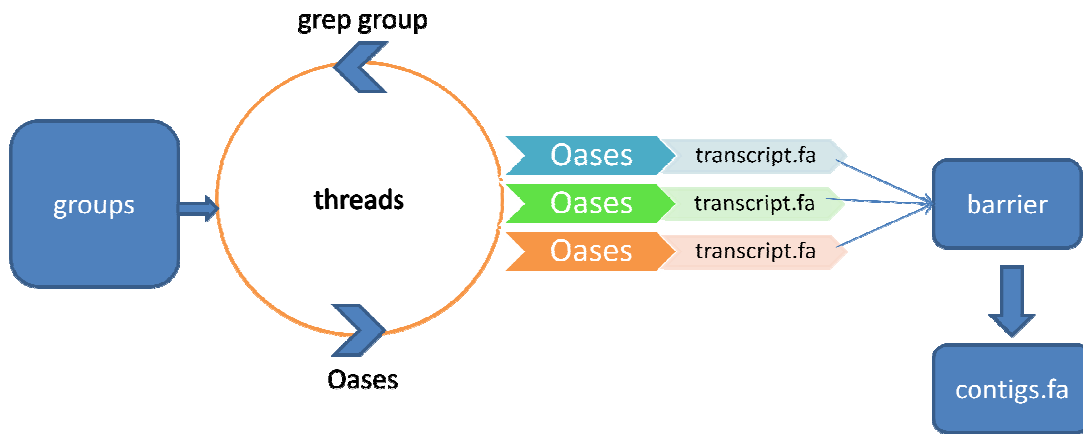


Figure 22. Alignment process.

```

#pragma omp critical{
  sprintf(commandbb, "cat %s/%d/Oases/transcripts.fa >> %s/CBA/contigs.fa", strPath, omp_get_thread_num(), strPath);
  system(commandbb);}
  
```

Figure 23. OpenMP barrier.

CHAPTER IV – RESULTS

This chapter presents the results of testing our program using various types of data and then compares them with other assembly algorithms. In the results, the most important comparisons are short reads mapping rate, contigs mapping rate, and recovery rate. The short reads mapping rate shows how many short reads are used for assembly. Contigs mapping rate shows how many contigs are valid, which can be mapped to the template. Recovery rate shows how many nucleotides in the templates are covered by contigs. Those three statistics indicate how well the assembly methods work.

4.1 Testing Data

We use ERCC-BGI data, human Chromosome 22 real data, and human Chromosome 22 simulation data to test the CBA-based-on-Oases program. We compared the CBA-based-on-Oases program with Oases, Trinity, ABySS, and Mira. All experiments were run with Velvet version 1.2.03, Oases 1.2.03, ABySS 1.3.3, Mira 3.0.4.1, and Trinity 2012-10-05.

4.1.1 ERCC Data

To control the quality of RNA quantification, a common set of external RNA controls was developed by the External RNA Controls Consortium (ERCC), an ad-hoc group of academic, private, and public organizations hosted by the National Institute of Standards and Technology (NIST). Approximately 90 companies, universities, and federal laboratories in the ERCC are developing materials and tools that can be used to benchmark

(Baum, 2006). Therefore, ERCC data can be used to calibrate bioinformatics methods in analyses of RNA-Seq data.

Our ERCC data come from Beijing Genomics Institute (BGI). We used ERCC data to test CBA and other assembly methods. Our ERCC data has 92 transcripts. We got 220,000 short reads with 50 bp in length from BGI (Introduction to BGI-Hong Kong, 2013). Not all 92 transcripts contained mapped short reads. To reduce noise and variability, we removed all transcripts with fewer than 1k short reads mapped and also removed their corresponding short reads from the raw data. Finally, we had 216k short reads and 16 transcripts.

We use Bowtie 2 (Ben Langmead, 2011) to calculate mapping rates: short reads map to contigs, and recovery rate: nucleotides cover to transcripts. We use ABySS-fac (Simpson et al., 2009) (Simpson, Kim, Jackman, Schein, Jones, & Birol, 2009) to calculate N: how many contigs, and N50: average length of contigs. We randomly collected short reads from raw data for each number 3 times and tested them.

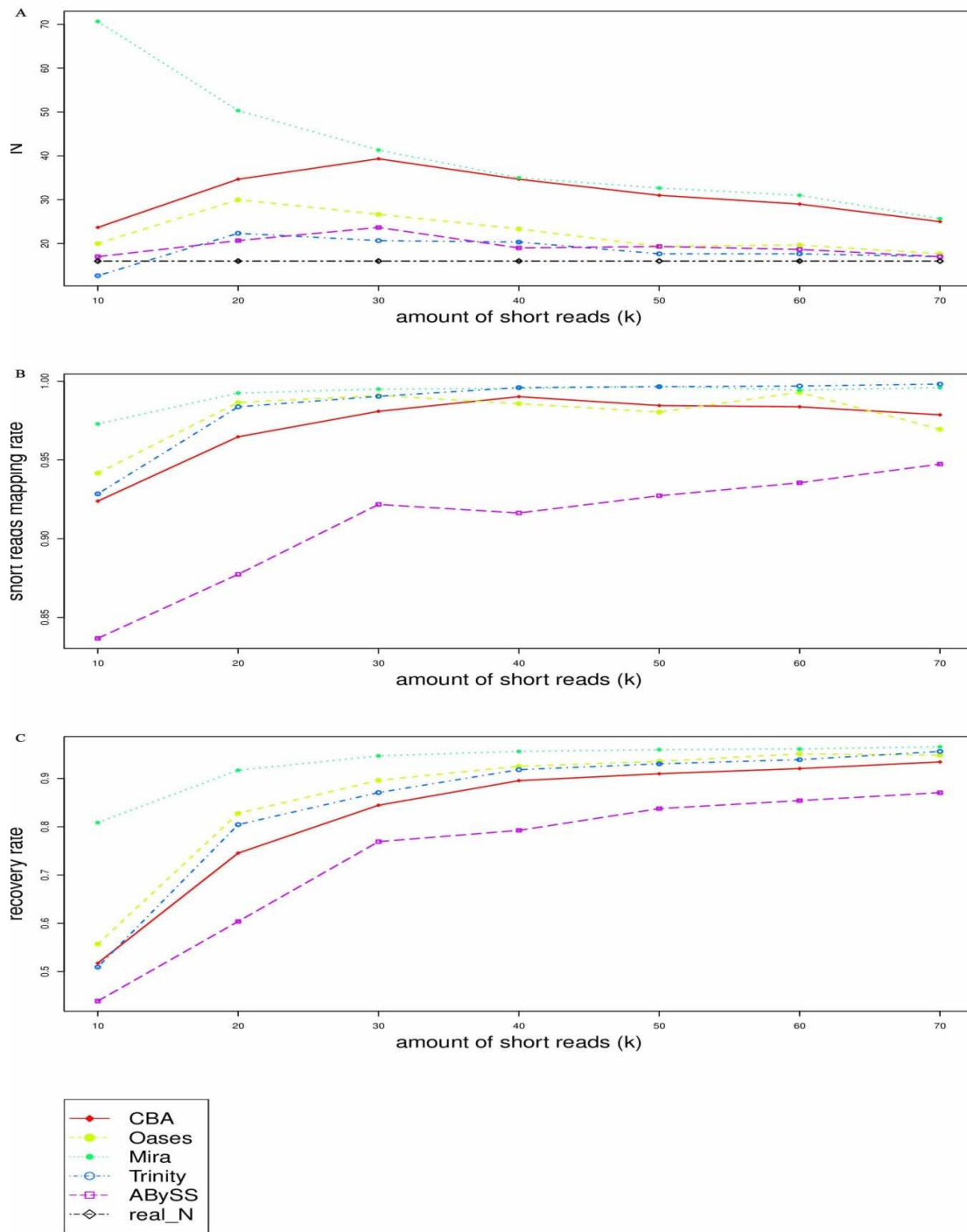


Figure 24. ERCC-BGI data test results.

In Figure 24, the x-axis indicates the number of short reads collected from the ERCC-BGI short reads file. The y-axis shows our statistical measures.

Figure 24A shows N value, which indicates the number of contigs for each assembly method. We expect N will be closer to the number of transcripts, which is 16. Mira has highest value at 70 contigs, when the number of short reads was low. As the short reads number increased, Mira and CBA's N values dropped to about 25—but still higher than other de Bruijn graph-based programs. The N values of Oases, ABySS, and Trinity were very close to the expected value.

Figure 24B shows the short reads mapping rate, which indicates how many short reads are overlapped to contigs. ABySS mapped only 83% short reads to contigs at the beginning. As short reads numbers increased, ABySS's mapping rate was increasing but was still lower than the other methods. Mira and Trinity were stable and almost mapped all short reads to contigs. CBA and Oases were also doing a good job here, with a 98% mapping rate.

Figure 24C shows the recovery rate, which indicates how many nucleotides in the template are covered by contigs. When the number of short reads was low, all methods had recovery rates lower than 60% except Mira, which was 80%. But as short reads numbers increased, all method recovery rates were higher than 80%—especially Mira, Trinity, Oases and CBA, with recovery rates higher than 90%, a very high value.

CBA did not outperform other assembly methods for the ERCC tests because the ERCC templates have few overlapped nucleotides, and de Bruijn graph-based assembly methods would not have many Chimeric Edges. Therefore, CBA showed little difference

against other assembly methods in short reads mapping rate, contigs mapping rate, and recovery rate.

4.1.2 Human Chromosome 22 Simulation Data

Simulated data is generated from a genomic region in human chromosome 22 that has 5,000,000 nucleotides and 337 transcripts. The simulation program randomly generates 100 to 1,000 short reads for each transcript, and short reads randomly generate a 0.2% error rate. We also randomly generated short reads from transcripts for each number 3 times and tested them.

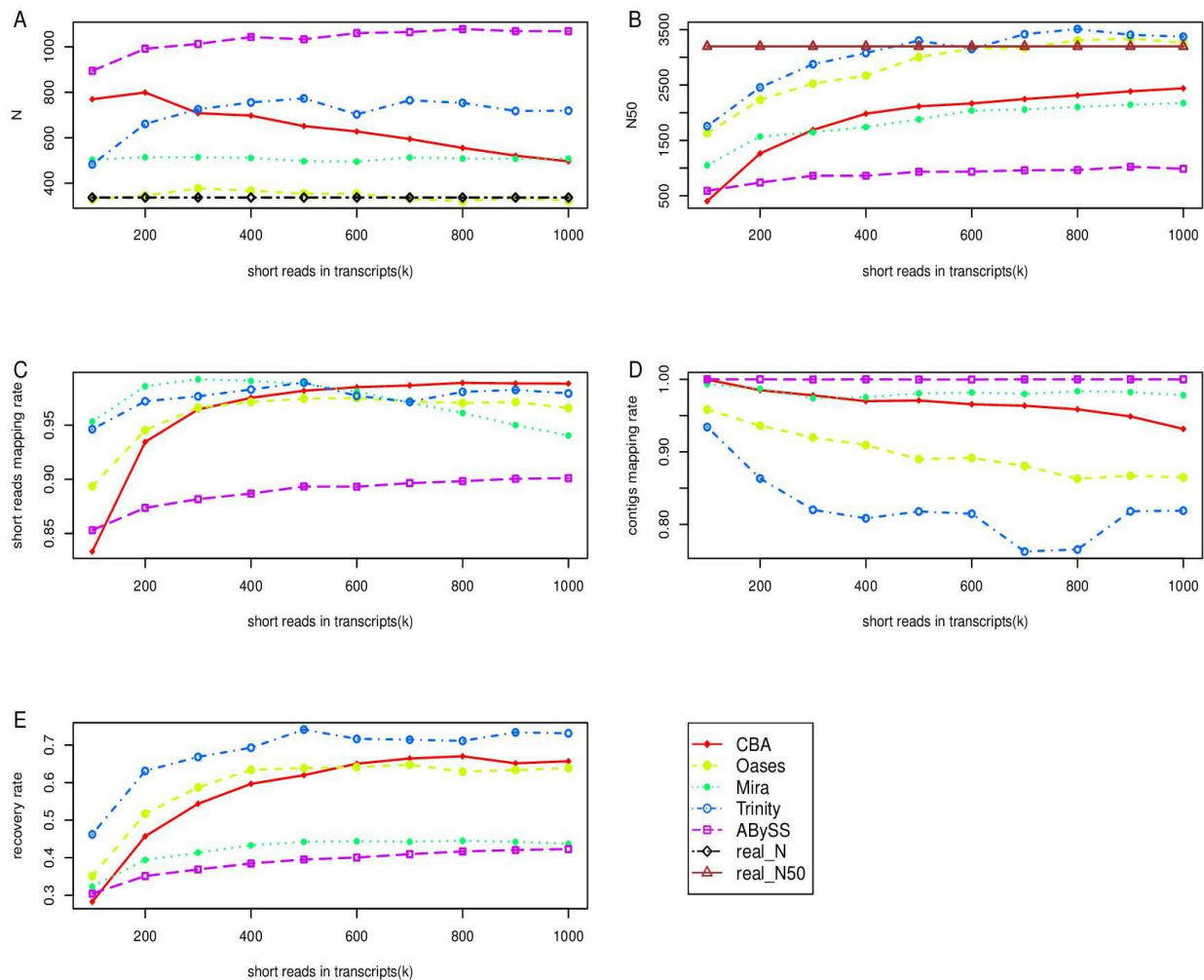


Figure 25. Human chromosome 22 simulation data test results.

In Figure 25, the x-axis indicates the number of short reads generated from human chromosome 22 transcripts file. The y-axis shows our statistical measures.

Figure 25A shows the number of contigs. Real_N is our expected value, which equals 337. Oases showed a very good and stable curve, close to value 337. CBA and ABySS had very high values at the beginning. The CBA curve was dropping to 500 and the ABySS curve increased to over 1,000 when the number of short reads in each transcript

began increasing. Trinity and Mira both had an N value of 500 when the short reads number was low. Th Trinity curve increased to 700 and Mira kept stable at 500, when the short reads number was increasing.

Figure 25B shows N50, which means the average length of contigs. This is a very clear graphic. Trinity and Oases had a higher N50 value, both generating longer contigs, with an average length of 3400 bp. CBA and Mira generated medium-length contigs, average length of 2400 bp. ABySS generated very short contigs, average length only 100 bp.

Figure 25C shows the short reads mapping rate. CBA had the worst short reads mapping rate, 83% when the short reads number was low. As short reads numbers increased, CBA had the best short reads mapping rate, 99%. Trinity and Oases' short reads mapping rate were close to 95%. ABySS's short reads mapping rate was lower than 90%.

Figure 25D shows the contigs mapping rate. Mira and ABySS's contigs mapping rates were stable and close to 100%. Next highest was CBA, which had about 95% contigs mapping rate. Oases and Trinity had a little bit lower contigs mapping rate, about 80%.

Figure 25E shows the recovery rate. When short reads numbers were low, all methods' recovery rates were lower than 50%. As short reads numbers increased, Trinity's recovery rate increased fast, finally at 72%. Oases' and CBA's recovery rate increased to 65%. Mira's and ABySS's recovery rate stayed lower than 50%.

Results show CBA had the best short reads mapping rate, better contigs mapping rate, and better recovery rate. Oases had better N and N50 value. Trinity had the best recovery rate but had the worst contigs mapping rate. ABySS and Mira were not doing a

good job in this test. Even Trinity had a higher recovery rate, but its contigs mapping rate was too low. In Figure 25A and B, we can see that Trinity generated more contigs than others except for ABySS, and its contigs average length was high, even higher than template average length. That indicated lots of contigs Trinity generated but cannot map to the template, but it still had some long contigs mapped to the template, which is why Trinity had a high recovery rate but low contigs mapping rate. ABySS had the highest contigs mapping rate, but its short reads mapping rate and recovery rate were both low. In summary, CBA had the best result in Chromosome 22 simulation test.

4.1.3 Human Chromosome 22 Real Data

Chromosome 22 was the first human chromosome to be fully sequenced, representing between 1.5–2% of the total DNA in cells. We randomly collect 200k to 2m short reads from original chromosome 22 file, which has 24,388,258 short reads. We used those short reads to test CBA, Oases, Mira, and Trinity. The results showed N, N50, short reads mapping rate, contigs mapping rate, and recovery rate. We randomly collected short reads from raw data for each number 3 times and tested them.

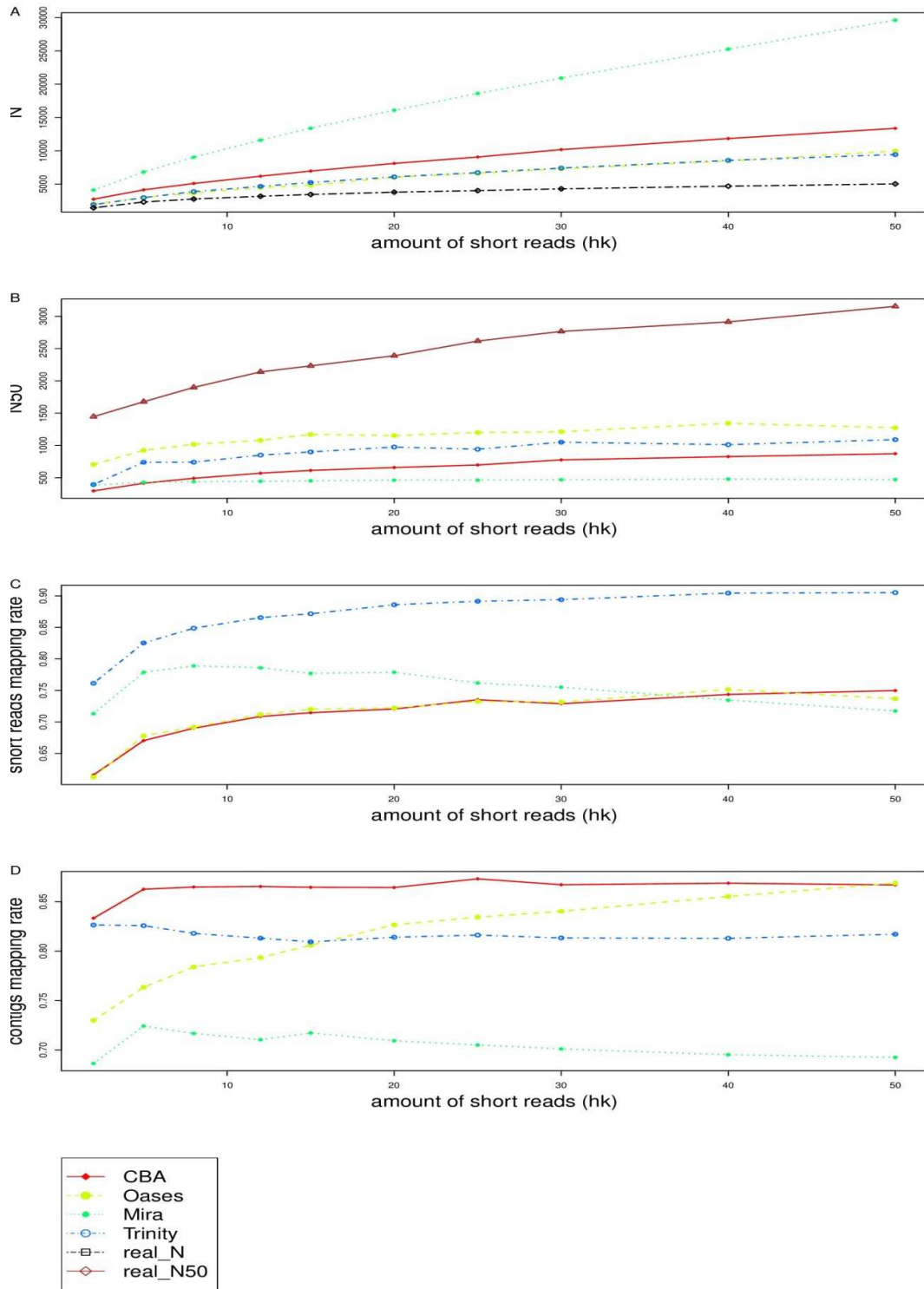


Figure 26. Human chromosome 22 real data test results.

In Figure 26, the x-axis indicates the number of short reads, collected from human chromosome 22 real short reads file. The y-axis shows our statistical measures.

Figure 26A shows the number of contigs. Real_N shows transcripts number, which is our expected value. When the short reads number increased to 5,000,000 per transcript, Mira had 30,000 contigs, higher than the expected value 6 times. This value was not acceptable. The next highest N value was CBA, which was 3 times higher than the expected value.

Figure 26B shows N50. Oases has the highest value, which means Oases generated the longest contigs, average length was 1200 bp. CBA and Trinity also generated long contigs when the short reads number was increasing. Mira had stable low N50 value, contigs, and an average length was 420 bp. This value was not acceptable.

Figure 26C shows a short reads mapping rate. At the beginning, Mira and Trinity had the highest short reads mapping rate. The graphs show their short reads mapping rate at greater than 70%. As the short reads numbers increased, Mira's short reads mapping rate was going down, but Trinity's short reads mapping rate was going up. Finally, Trinity had the highest short reads mapping rate, 90%. Mira and other methods only had 70–75% short reads mapping rate.

Figure 26D shows the contigs mapping rate. When the short reads number was low, Trinity and CBA had 83% contigs mapping rate, and Mira's and Oases' contigs mapping rate were lower than 75%. As the short reads numbers increased, Trinity's contigs mapping rate was going down, and CBA's and Oases' contigs mapping rate were going up. CBA and Oases had 85% contigs mapping rates when the short reads number was 5,000,000. Mira

kept a low contigs mapping rate, lower than 70%, the worst one.

Results show that CBA had the best contigs mapping rate. Oases had better N and N50 value, which means Oases generated longer contigs. Trinity had the best short reads mapping rate. Mira showed the worst results in this test. Trinity had the best short reads mapping rate, but its contigs mapping rate was low. That shows a classic de Bruijn graph algorithm error. High short reads mapping rate indicate de Bruijn graph assembles short reads when they share one k-mer, so almost all short reads can map to contigs. But some of those contigs may generate by Chimeric Edges, which cannot map to the template. That is why Trinity had a low contigs mapping rate. CBA had the second-highest short reads mapping rate and kept the highest contigs mapping rate. In conclusion, CBA was the best algorithm in the Chromosome 22 real test.

Mira is the only one that uses the overlap graph-based algorithm. In the ERCC test, Mira did a good job when compared with other de Bruijn graph-based algorithms. But in the human Chromosome 22 test, Mira did not have good results, which means Mira cannot solve complex gene structures. Further, the overlap graph-based algorithm usually runs slowly. Table 1 shows the time cost for each assembly method when the short reads number was 5,000,000 in the human Chromosome 22 real test. Mira's running time was twice that of CBA's and 10 times longer than other de Bruijn graph-based assembly method.

Table 1

Running Time Comparison of All Assembly Methods

Short Reads Number /Time Consume (Minutes)	CBA	Oases	Trinity	ABYSS	Mira
5,000,000	615	40	150	15	1203

CHAPTER V – DISCUSSION

We have developed the Clustering Based Assembly (CBA) method, a novel approach that uses clustering and de Bruijn graph-based algorithm (Oases) for de novo assembly of non-model species, such as plants, yeasts, and animals such as the snapping turtle.

The ori-cut value, max-cut value, and ave-cut value are used in the program to control how to create groups. For example, in the human chromosome 22 real test, based on Figure 16, the x-axis means how many k-mers short reads are shared, and the y-axis means the number of short reads that share the same number of k-mers. We tested chromosome 22 real data for different k-mer size. There is a big gap in the frequencies of short reads between 12 and 14 (changing more than 2 times), implying most of the k-mer-sharing k-mers below 14 are random noise. We used 25 as the max-cut value because it gave the lowest error rate after testing 20 to 30 in all ERCC and human chromosome studies. Finally, we set ave-cut value as a control value for merging groups. We tested number 2 and number 5 as ave-cut values in all ERCC and human chromosome study, and the simulation study shows number 2 gave the highest mapping rate. If two or more WL pairs connect two groups, we will then consider whether the two groups should be merged. We require each group to contain 5 or more short reads.

CBA is an accurate assemble algorithm, but grouping is time consuming. Using openMP can make the programs run in parallel, which saves running time. Another way to

save time is using a hash table as the searching method. As we showed in chapter 3, hash table's time complexity is close to $O(1)$. It is still important to choose the right hash function. We choose MurmurHash function in our program. MurmurHash function is a kind of hash function used by Google's search engine. Compared with other hash functions, MurmurHash has a faster computing speed and lower collision hash value.

Table 2

Comparison of MurmurHash Function with Others in Speed and Collision Field

Hash function	Speed	Collided item
Crc32	1.725	0
Crc-O	0.185	0
Md5	2.86	112
Doobs	0.251	110
Murmur	0.033	0
Stdhash	0.917	992336
Jhash	0.239	126

In Table 2 (Sina, 2012), the author compared MurmurHash function with other accepted hash functions. Speed analysis indicates how much time is used when running 10,000 times with 1,000,000 different words. Collided item indicates how many collision values hash functions generated. MurmurHash has the highest speed and zero collided items. As we know, a hash table is memory consuming, so our program maps a discrete hash table to a hash map to save memory. Further, CBA uses a discrete memory store structure that means memory will not be easy to release. A Boost pool algorithm gives a solution to create a big block memory pool used for assigning single discrete memory

block. When we need to release memory, we remove only the memory pool, which helps save time and memory. Even with all the optimizations shown, CBA still used more than 200GB in memory. It cannot be used in a personal computer.

CBA cannot work for two transcripts that overlap a long fragment of nucleotides. If the length of the overlapped fragment is longer than the length of the short read, it means short reads from the two transcripts share almost all k-mers in the critical area. The cluster process of CBA will think all short reads from the two transcripts have strong links. Finally, CBA will cluster them into one group. To our knowledge, no algorithm can split those transcripts that share long fragment nucleotides. But CBA is better than any other program in the condition of overlapping transcripts.

CBA has another limitation: Since CBA is memory constrained; it cannot run on a personal computer. We used 40 threads parallel run CBA, so CBA would run better in multiple core servers.

Currently, we use Oases as the CBA alignment method because Oases performs well overall by adapting to varying conditions and is superior overall compared to other alignment methods (Schulz et al., 2012). (Schulz, Zerbino, Vingron, & Birney, 2012)

In the future, we will develop a new alignment algorithm to work with the grouping algorithms. We will try to discover a way to assemble short reads more accurately compared to the de Bruijn graph-based algorithm.

CHAPTER VI – CONCLUSIONS

The fast development of next-generation sequencing presents a major challenge to bioinformatics analysis. One of the underdeveloped areas in bioinformatics is de novo assembly of RNA-Seq data. Although many assemblers have been developed for next-generation sequencing data, few can provide desired accuracy as well as maintain satisfactory computing speed. We set out to develop a novel assembler, taking two steps to assemble RNA-Seq short reads: clustering and alignment. By combining a clustering step with de Bruijn graph-based algorithm, we targeted minimizing the error rate of sequence alignments.

Our results show that CBA is the best assembler in overall performance when comparing with other de Bruijn graph and overlap graph-based algorithms in various types of data. For ERCC data, all assemblers gave acceptable performance. This is because ERCC templates have few overlapped nucleotides, and even de Bruijn graph-based assembly methods would not generate many incorrect contigs. The human genome, however, has long transcripts, and there exists many overlapped fields between each transcript. As the results for human chromosome 22 simulated and real data illustrate, the de Bruijn graph-based assembly methods had high error rates, as shown by the low short reads mapping rates, contig mapping rates, and recovery rates. For the human data, CBA consistently had high short reads mapping rates, contig mapping rates, and recovery rates. It was expected that an overlap graph-based assembler, such as Mira, would have a high

accuracy rate for human data. However, our tests failed to show that. Actually, Mira performed rather poorly on human data. Furthermore, when comparing the computational time for all these methods, Mira took at least twice the running time than did other methods. Therefore, an overlap graph-based algorithm is not suitable for assembling NGS sequences.

Based on the test results, we proved our CBA method is valuable in RNA-Seq applications. CBA not only provides a high accuracy when assembling RNA-Seq data but also gives acceptable computational speed. We expect the CBA method will be widely used for RNA-Seq studies. We look forward to developing upgraded versions of CBA by optimizing the alignment step.

REFERENCES

- 2014 NGS field guide – Table 2 – Run time, reads, yields, and costs. (2014). Retrieved April 30, 2014, from <http://www.molecularecologist.com/next-gen-table-2-2014/>
- Adams, J. U. (2008). Transcriptome: Connecting the genome to gene function. *Nature Education*, 1(1), 195.
- Appleby, A. (2011). *MurmurHash*. Retrieved October 3, 2013, from <https://sites.google.com/site/murmurhash/>
- Baum, M. (2006). ERCC to begin test rounds for final RNA reference set. *NIST*. National Institute of Standards and Technology. Retrieved October 22, 2013, from http://www.nist.gov/mml/bbd/cell_systems/ercc_091406.cfm
- Ben Langmead, S. L. (2011). Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 1: 357–359.
- Biol, L., Jackman, S. D., Nielsen, C. B., Qian, J. Q., Varhol, R., Stazyk, G., et al. (2009). De novo transcriptome assembly with ABySS. *Bioinformatics*, 5(21): 2872–2877.
- Chevreux B, P. T. (2004). Using the miraEST assembler for reliable and automated mRNA transcript assembly and SNP detection in sequenced ESTs. *Genome Research*, 14(6), 1147–1159.
- Costa, V., Angelini, C., Feis, I., & Ciccodicola, A. (2010). Uncovering the complexity of transcriptomes with RNA-Seq. *Journal of Biomedical Biotechnology*. doi: 10.1155/2010/853916

- de Bruijn, N. G. (1946). A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 758–764.
- Faghihi, M. A., & Wahlestedt, C. (2009). Regulatory roles of natural antisense transcripts. *Nature Reviews: Molecular Cell Biology*, 10(9), 637–643.
- Grabherr MG, H. B.-T. (2011). Full-length transcriptome assembly from RNA-Seq data without a reference genome. *Natural Biotechnology*, 29, 644–652.
- Hoppman-Chaney, N., Peterson, L. M., Klee, E. W., Middha, S., Courteau, L. K., & Ferber, M. J. (2010). Evaluation of oligonucleotide sequence capture arrays and comparison of next-generation sequencing platforms for use in molecular diagnostics. *Clinical Chemistry*, 56(8), 1297–1306.
- Introduction to BGI-Hong Kong*. (2013). Retrieved April 30, 2014, from http://www.genomics.cn/en/navigation/show_navigation?nid=4179
- Manteniotis S, L. R. (2013). Comprehensive RNA-Seq expression analysis of sensory ganglia with a focus on ion channels and GPCRs in Trigeminal ganglia. *PLoS One*, 8(11), e79523.
- Marguerat, S., & Bähler, J. (2010). RNA-Seq: From technology to biology. *Cellular and Molecular Life Sciences*, 67(4), 569–579.
- Ozsolak, F., & Milos, P. M. (2011). RNA sequencing: Advances, challenges and opportunities. *Natural Reviews. Genetic*, 12(2), 87–98.
- Park, P. J. (2009). ChIP-Seq: Advantages and challenges of a maturing technology. *Natural Reviews. Genetics*, 10(10), 669–680.

- Pevzner, P. A., Tang, H., & Waterman, M. S. (2001). An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences U S A*, 98(17), 9748–9753.
- Roach, J. C., Glusman, G., Smit, A. F., Huff, C. D., Hubley, R., Shannon, P. T., et al. (2010). Analysis of genetic inheritance in a family quartet by whole genome sequencing. *Science*, 328(5978), 636–639.
- Sadava, D., Hillis, D. M., Heller, C. H., & Berenbaum, M. (2012). *Life: The science of biology*. New York: W. H. Freeman.
- Schulz, M. H., Zerbino, D. R., Vingron, M., & Birney, E. (2012). Oases: Robust de novo RNA-Seq assembly across the dynamic range of expression levels. *Bioinformatics*, 28(8), 1086–1092.
- Shendure, J., & Ji, H. (2008). Next-generation DNA sequencing. *Nature Biotechnology*, 26, 1135–1145.
- Simpson, J. T., Kim, W., Jackman, S. D., Schein, J. E., Jones, S. J., & Birol, I. (2009). ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6), 1117–1123.
- Sina. (2012, December 17). MurmerHash [Web log comment]. Retrieved October 3, 2013, from http://blog.sina.com.cn/s/blog_ab30208a010154ec.html
- Turnpenny, P. D., & Ellard, S. (2007). *Emery's Elements of medical genetics*. Cambridge, UK: Cambridge University Press.
- Wang, Z., Gerstein, M., & Snyder, M. (2009). RNA-Seq: A revolutionary tool for transcriptomics. *Natural Reviews. Genetics*, 10(1), 57–63.

Wu, S., Zhu, Z., Fu, L., Niu, B., & Li, W. (2011). WebMGA: A customizable web server for fast metagenomic sequence analysis. *BMC Genomics*, *12*:444.

Zerbino, D. R., & Birney, E. (2008). Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, *18*(5), 821–829.

Zhong Wang, M. G. (2009). RNA-Seq: A revolutionary tool for transcriptomics. *Natural Reviews. Genetics*, *10*(1), 57–63.