Electronic Thesis and Dissertation Repository

2-11-2015 12:00 AM

Metagenome Assembly

Wenjing Wan, *The University of Western Ontario*

Supervisor: Lucian Ilie, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science
© Wenjing Wan 2015

METAGENOME ASSEMBLY

(Thesis format: Monograph)

by

Wenjing <u>Wan</u>

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Masters of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

The advent of the next generation sequencing technology (NGS) makes it possible to study metagenomics data which is directly extracted and cloned from assemblage of micro-organisms. Metagenomics data are diverse in species and abundance. Because most genome assemblers are designed for single genome assembly, they could not perform well on metagenomics data. To deal with the mixed and not uniformly distributed metagenomics reads, we developed a novel metagenomic assembler named MetaSAGE, on the platform of the existing SAGE assembler. MetaSAGE finds contigs from the overlap graph based on the minimum cost flow theory and uses mate-pair information to extract scaffolds from the overlap graph. When facing chimeric nodes, the MetaSAGE splits them separately according to the coverage of edges. MetaSAGE exhibits good performance compared to existing metagenomic assemblers.

Acknowlegements

I would like to express my sincere gratitude to everyone who was involved in the completion of this thesis. First of all, I would like to express my deep appreciation to my supervisor Dr. Lucian Ilie. He guided me into this research area, taught me the method and attitude to do scientific research. His diligence and preciseness show me a good example to my future work.

I would like to express my thankfulness to Michael Molnar who helped me a lot in understanding and testing this program.

I also would like to thank to Nilesh Khiste. I learned a lot from the discussion with you. I would like to say thank you for the help from all my lab-mates and colleagues, and the support from my friends and family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The mechanism of life is complex and evokes curiosity. After scientists discovered the structure of the nucleus, the public understood that it is genes that control the birth, fading, disease, death and all other processes of life. Modern biology theories tell us that genes are some functional parts on chromosomes, which are made up of sequences of DNA (Deoxyribonucleic acid). When the double helix structure of DNA sequences was discovered, the studies of the DNA sequences began to boom. A DNA sequence is made up of pairs of nucleotides binding with the hydrogen bounds. It is very important to know the order of those nucleotides because it has been proved that their order controls the construction and function of organisms. Due to their huge size, biologists need some tools to help them understand the structure of DNA sequences. The study of developing mathematical methods and software tools for understanding biology data is called bioinformatics.

DNA sequences must be encoded before we study them. Scientists use four characters A, T, G, C to represent the nucleotides on a DNA sequence. In order to obtain the structure of a DNA sequence, biologists will first break a long DNA sequence into manageable pieces. Then they clone these fragments and sequence them individually. So the fragments are represented as a set of short strings, which are called reads. The process of obtaining reads from DNA sequences is called DNA sequencing. After producing those fragments, scientists will try to reconstruct the DNA sequence with the reads they encoded. The process of reconstructing DNA sequences from reads is called genome

assembly. Genome assembly became a very important topic as it can give biologists the first-hand references of unknown genomes. In the last several years, many genome assemblers have been published, such as Velvet [40], ALLPATHS [5], ABySS [37], SOAPdenovo [17], SGA [36] and so on. Most of them are applications of one or more of the following strategies: greedy, overlap graph and de Bruijn graph. These assemblers build a graph on the set of reads, connect overlaped edges and then extract contigs (definition are given in Chapter 2.3.2) from the graph.

Recently, new sequencing technologies were developed by scientists, such as Roche/454 [18], Illumina/Solexa [8, 38] and SOLiD sequencing [35], which are generally referred to as *next generation sequencing* (NGS). Compared to the old Sanger sequencing method, the NGS methods generate short reads with hundreds of base pairs (bp) or even less than one hundred bp. However they can sequence very fast and generate high coverage. A number of new applications of sequencing technologies have become available because of these new seqencing methods. One of these applications is metagenomics.

Metagenomics is defined as "the genomic analysis of micro-organism by direct extraction and cloning of DNA from an assemblage of micro-organisms", and its importance stems from the fact that 99% or more of all microbes are deemed to be unculturable [13]. Goals of metagenomic studies include assessing the coding potential of environmental organisms, quantifying the relative abundances of specific species, and estimating the amount of unknown sequences. Such studies are made possible by the use of next-generation sequencing technologies. Metagenomic assemblers are similar to classic genomic assemblers, since both of them look for the optimal assembly of the reads and try to produce long contigs or scaffolds from short reads. However, metagenomic assemblers are faced with more difficulties because of the uncertainty of the abundance and composition of metagenomic data.

In this thesis, we proposed a new metagenomic assembler, MetaSAGE, which is based on the recent assembler SAGE [14]. In Chapter 2 we introduce the background and some notions in DNA sequencing and genome assembly, as well as a brief overview of the next generation sequencing (NGS) technologies. Subsequently, we introduce two paradigms used in genome assembly: overlap graph and de Bruijn graph. At the end of Chapter 2,

a review of several popular metagenomic assemblers is provided.

In Chapter 3 we introduce our novel metagenomic assembler, MetaSAGE. An overview of this program is provided at the beginning of this chapter, followed by the algorithm and technical details of each step of this program. We discuss also our algorithm for correcting errors in reads, MetaRACER, which is a modification of the existing RACER program [15], adapted to the features of metagenomics data.

In Chapter 4, a comparison between MetaSAGE and three top metagenomic assemblers is made. We introduce the criteria used in the comparison and present the detailed results of this comparison. The datasets we used were artificially generated by MetaSim [32]. We generated input data in 4 taxonomic levels to make the comparison comprehensive.

In Chapter 5, a conclusion about this program is made, including the analysis of our work and its prospect for the future.

Chapter 2

Background

In 1953, James Watson and Francis Crick [39] suggested the first correct *double-helix* model of DNA structure. Since then scientists have been trying to understand the information stored in DNA. DNA is a double-helix sequence composed of many small units called nucleotides. Each nucleotide has a nitrogen-containing nucleobase, either *guanine*, *adenine*, *thymine*, or *cytosine*. In short we denote them as G, A, T, C. These nucleotides in the DNA molecule are also known as *bases*. The order of four bases appearing in a DNA molecule provides the instructions for making proteins. This order spells the genetic information and controls all biological functions of a living organism. Within cells, DNA is organized into long structures called *chromosomes*. On a given chromosome, there are specific sequences of nucleotides at given positions that code for some proteins. We call them *genes*. In another word, genes are some functional parts of a DNA sequence. They determine the construction of protein, known as the basic functional component of living things. Figure 2.1 [2] indicates the relationship between chromosome, DNA, and gene. It is very important to understand DNA sequence because it helps scientists to understand how a living organism is constructed and what is its function. For example, scientists can use DNA sequence of an organism to identify and predict health risks. Hence, knowing the DNA sequence of an individual could help discover diseases long before they might be identified otherwise.

In a DNA double helix, each type of a nucleotide will typically bond to another type of a nucleotide forming a *base pair*. Adenine (A) always bonds to thymine (T) with two
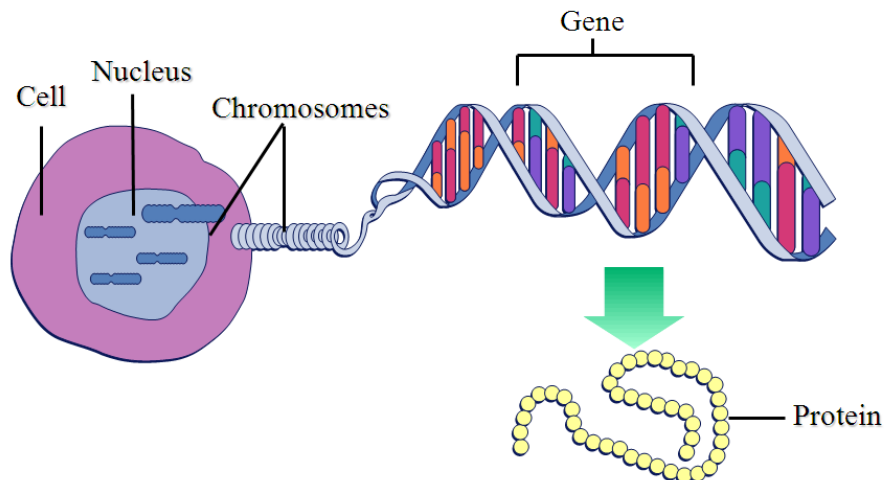
Figure 2.1: Chromosome, DNA, gene and protein.

hydrogen bonds, and cytosine (C) bonds to only to guanine (G) with three hydrogen bonds. Figure 2.2 from [4] shows a G-C base pair with three hydrogen bonds and an A-T base pair with two hydrogen bonds. Non-covalent hydrogen bonds between the pairs are shown as dashed lines. The back bone of a DNA is made from alternating phosphate and sugar residues. The sugar in DNA is 2-deoxyribose, which is a five-carbon sugar. The sugars are joined together with bonds between the third and fifth carbon. This means that the DNA sequence has two different ends. We denote the two ends as 5'(*five prime*) and 3' (*three prime*). Directions on DNA usually start from 5'-end to 3'-end. Figure 2.3 [1] describes how nucleotides are located on the DNA sequence.

2.1 Sequencing

In 1977 the first full genome was sequenced. This remarkable achievement was attained by Frederick Sanger [7] and his team, who sequenced the genome of *bacteriophage phiX*174, which is about 5 kb in size. This technology is known as *Sanger sequencing*, or *first generation sequencing* technique. To sequence a genome, first DNA is broken into manageable pieces. Second, the fragments are multiplied through a process called cloning, and then individual fragments are sequenced. In the end, a library of DNA subsequences is generated. However the Sanger sequencing technique has a few disadvantages. The
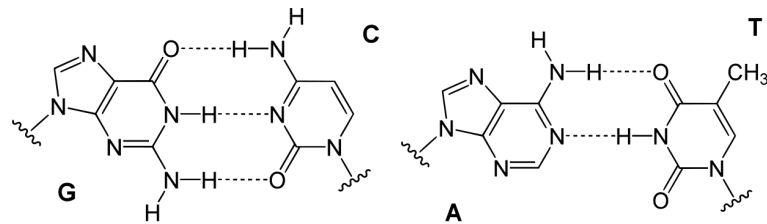
Figure 2.2: Hydrogen bonds on DNA.
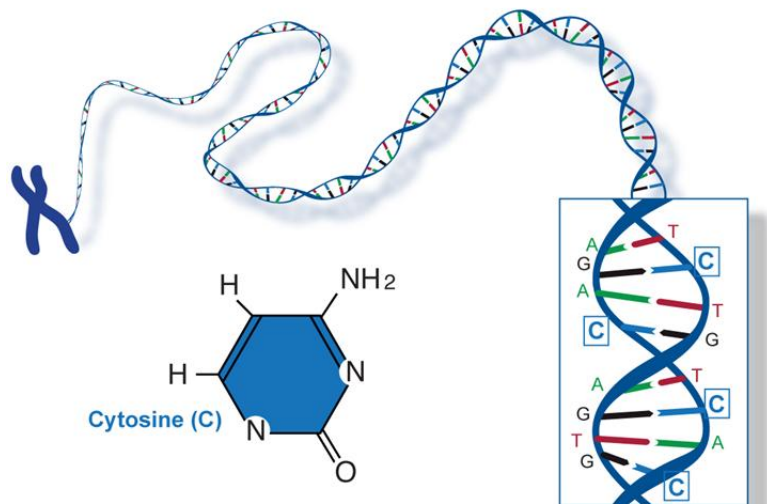
Figure 2.3: Nucleotides on DNA sequence.

major one is that it is a costly and time consuming process. At about $1 per kpb (kilo base pairs), it would cost about $30,000,000 to sequence a complete human genome with 10x coverage (for an explanation of the notion coverage, see Section 2.3.2). The coverage of the Sanger sequencing is usually low, meaning that it is impossible to sequence the parts of a genome which are not sampled. Moreover, it is not possible to clone some parts of a chromosome with the Sanger method because its cloning method is biologically biased.

2.2 Next generation sequencing

Even though the Sanger sequencing method was a really significant breakthrough in technology and it has been widely used in biochemistry, the incentive for developing en-

tirely new strategies for DNA sequencing has emerged on at least four levels [34]. First, in the wake of the Human Genome Project, it was hard make a significant reductions in the cost of conventional DNA sequencing methods. Second, with the success of the whole genome assemblies, potential utility of short-read sequencing has been tremendously strengthened. Third, high-throughput DNA sequencing methods make a wide range of biological phenomena accessible. And fourth, alternative strategies have been made in disparate fields, including microscopy, surface chemistry, nucleotide biochemistry, polymerase engineering, computation, data storage and others, making the new DNA sequencing increasingly practical to use.

Under the promotion of the incentives mentioned above, several new sequencing techniques were developed by scientists, such as Roche/454 [18], Illumina/Solexa [8, 38] and SOLiD sequencing [35]. Generally they are referred to as *next generation sequencing* (NGS). Compared to the old Sanger sequencing method, the NGS methods generate short reads with hundreds bp or even less than one hundred bp. However they can sequence very fast and generate high coverage. These features made many applications become possible with reads coming from NGS. In the past several years there has been an accelerating flurry of publications in which NGS is applied for a variety of goals. Many of these applications rely on the possibility to assemble the reads to reconstruct the original genome. This procedure is called *genome assembly*, which is one of the most difficult and widely investigated problems in bioinformatics. It is also the topic of this thesis.

2.3   De novo genome assembly

In this section, the problem of genome assembly is introduced, along with several basic concepts of DNA sequencing.

2.3.1   Problem description

Genome assembly refers to reconstructing the original DNA sequence from the reads. In biology, the nucleotides are represented as the four characters A, T, G, C. So the assembly of genome can be regarded as merging short string reads to form a long string

(the genome) over the alphabet $\Sigma = \{A, T, G, C\}$.

Formally, suppose we are given a set $R = \{r_1, r_2, \ldots, r_n\}$ of n reads, where the length of the reads is $|r_i| = l$. The goal of assembly is to construct a string $\mathscr{G}$ such that all reads in R are substrings of $\mathscr{G}$ (supposing the reads error free).

Example Consider the set $R = \{$ACG, CGA, CGC, CGT, GAC, GCG, GTA, TCG$\}$ of $n = 8$ reads of length $l = 3$. Concatenating all the reads in R produces the string of length 24 shown in Figure 2.4. Clearly, this is very far from the actual genome sequence. A much better solution would be the shortest superstring that has all reads as substrings, shown in Figure 2.5. As we shall see, we are not looking for the shortest superstring but of the most probable one.

```
ACGCGACGCCGTGACGCGGTATCG
012345678901234567890123
```

Figure 2.4: Concatenation of the reads.

```
TCGACGCGTA
0123456789
```

Figure 2.5: Shortest superstring of the reads.

2.3.2   Reads

The original genome sequence is referred to as the *reference genome*. Genome sequencing techniques cannot read the whole reference genome at one time. They generate fragments of the reference genome which are called *reads*. As mentioned before, NGS methods usually generate reads of hundreds bp in size. The size of reads that are generated by sequencing technology is called *read length*. Table 2.1 (from [25]) indicates the read lengths generated by several widely used NGS technologies.

Coverage

The technology generates many short reads, $R = \{r_1, \ldots, r_n\}$ which are all supposed to be subsequences of the genome sequence $\mathscr{G}$. Usually, the total length of reads is much larger than the reference genome length. Suppose the reference genome length $|\mathscr{G}|$ is L.

Technology	Read length (bp)	Output per run	Error rate	Paired-end
ABI/Solid	75	120 GB	Low(~2%)	Yes
Illumina/Solexa	100-150	1000 GB	Low(<2%)	Yes
Roche/454	400-600	700 MB	Medium(~4%)	No
Sanger	Up to ~2000	84 KB	Low(~2%)	Yes

Table 2.1: Sequencing technologies.

We define the *coverage* as:

$$coverage = \frac{\sum_i^n |r_i|}{|\mathscr{G}|} = \frac{nl}{L}. \tag{2.1}$$

The coverage shows the concentration when sequencing the reference genome. It represents the expected number of times that each position of the reference genome appears in the reads.

Complement

DNA sequence has two ends, the 5'-end and the 3'-end. The reads are always in the 5'-3' direction, but may come from either strand. Shown in Figure 2.6 are two reads, r_1 is the prefix of the top insert and r_2 is the suffix of the bottom strand. These are typical reads generated by the sequencing software of 20 bp long each. Since the DNA sequence is double stranded, every nucleotide on one string of DNA has a complementary nucleotide on the other string. The sequence made from the complementary nucleotides of a read is called the *complement* of that read. Because two complementary sequences in DNA are in reverse order, we call them *reverse complements*. The reverse complement of a sequence w is denoted as $\overline{w}$.

Insert and mate pairs

The sequencer brakes the reference genome into fragments to generate reads. The fragments are called *inserts*. The length of the insert is *insert size*. Insert is the unit for sequencing and it is double-stranded. The sequencer produces paired reads from both sides of one insert on its two complemented strands. The reads are always generated from 5'-end to 3'-end. The paired reads are called *mate pairs*. The mate pair is very

important for genome assembly because it keeps the information about the original lo-
cation of reads, which can help assemblers improve their accuracy. Figure 2.6 [12] shows
an example of mate pair obtained from the genome sequence.

Genome $\mathscr{G}$

Insert I_i

Insert size 61 bp

$5'-$ACTTACGATCGGCTAAGTTACACCGTTACATTGCCGATTACGCCGATACGACTAAAGCTAT$-3'$
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
$3'-$TGAATGCTAGCCGATTCAATGTGGCAATGTAACGGCTAATGCGGCTATGCTGATTTCGATA$-5'$
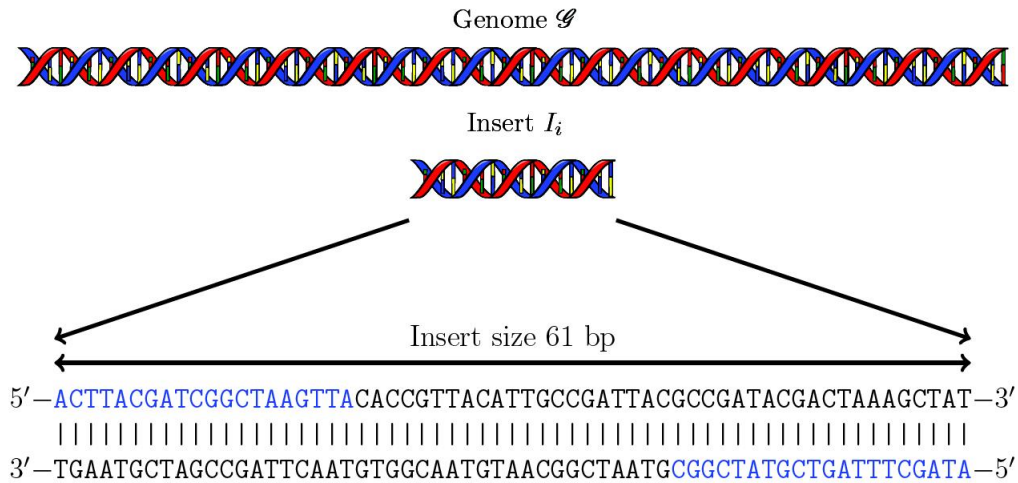
Figure 2.6: A mate pair is obtained from the genome sequence. The insert size is 61 bp
and the two reads (coloured in blue), each on one end of the insert, are of length of 20
bp.

Contigs and scaffolds

After assembling, a set of output sequences is obtained. We call those contiguous pieces
of DNA *contigs*. After assemblers obtain the contigs, they link those contigs that could
be ordered to get the *scaffolds*. A scaffold is a series of contigs that are in the right order
but not necessarily connected in one continuous stretch of sequence. The goal of the
genome assembly is to produce long and accurate contigs and scaffolds, covering as much
as possible of the reference genome.

2.4 Overlap graph

In genome assembly, short reads that overlap in suffix or prefix will be connected into
longer contigs. To achieve this, most algorithms will represent the reads as vertices in
the graph, connecting the vertices corresponding to overlapping reads by edges. There

are two kinds of graphs used in today's genome assembly algorithms: one is the overlap
graph and the other one is the de Bruijn graph. In this chapter, we first discuss the
overlap graph.

2.4.1   Type of edges

An overlap graph is a bidirected graph $G(V, E)$. Each node v in the overlap graph
represents a read and each edge $e = (u, v)$ represents the overlap between the reads
corresponding to two nodes u and v. Edges in the overlap graph have two arrowheads,
one at each point. Since the DNA sequence has two ends 5' and 3', the arrowhead
shows the orientation of the read, always from 5' to 3'. According to the combinations
of different arrowheads, there are 3 kinds of edges (overlaps); note the that the reverse-
reverse overlap is the same as the forward-forward overlap.

Forward-Forward Overlap

```
Read r₁  :   CACGTGCTGCCGATAATGTGGTTTCAGTAT
Read r₂  :      GTGCTGCCGATAATGTGGTTTCAGTATAAT
```
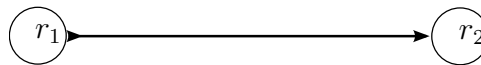

Reverse-Forward Overlap

```
Read r₁  :   CACGACGGCTATTACACCAAAGTCATATTA
Read r₂  :   GCTGCCGATAATGTGGTTTCAGTATAATGA


Read r̄₁  :   GTGCTGCCGATAATGTGGTTTCAGTATAAT
Read r₂  :      GCTGCCGATAATGTGGTTTCAGTATAATGA
```
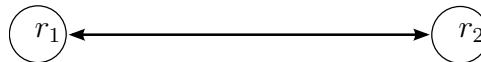
Forward-Reverse Overlap

Read r_1 : **GCTGCCGATAATGTGGTTTCAGTATAATGA**
Read r_2 : **GACGGCTATTACACCAAAGTCATATTACTC**

Read r_1 : **G****CTGCCGATAATGTGGTTTCAGTATAATGA**
Read $\overline{r_2}$:   **CTGCCGATAATGTGGTTTCAGTATAATGA****G**
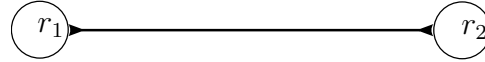
Figure 2.7 shows a set of reads. The read length is 30 bp. Figure 2.8 indicates the overlaps among these reads. If we represent those reads as vertices in the overlap graph and insert edges according to the overlaps among them we will obtain the overlap graph shown in Figure 2.9.

Read r_1 : **CACGTGCTGCCGATAATGTGGTTTCAGTAT**
Read r_2 : **TGCACGACGGCTATTACACCAAAGTCATAT**
Read r_3 : **CACGACGGCTATTACACCAAAGTCATATTA**
Read r_4 : **GCTGCCGATAATGTGGTTTCAGTATAATGA**
Read r_5 : **GACGGCTATTACACCAAAGTCATATTACTC**
Read r_6 : **GCCGATAATGTGGTTTCAGTATAATGAGGG**
Read r_7 : **ATAATGTGGTTTCAGTATAATGAGGGCAAT**
Read r_8 : **ACGACGGCTATTACACCAAAGTCATATGCA**
Read r_9 : **GCTGCCGATAATGTGGTTTCAGTATACGTA**
Read r_{10} : **ACGGCTATTACACCAAAGTCATATGCATAC**

Figure 2.7: A Set of 10 Reads

Formally the *overlap length* between two strings s_1 and s_2 is defined as the longest common substring that is a suffix of one string and a prefix of the other string. Consider two reads in Figure 2.8, r_1 and $\overline{r_2}$. The suffix of length $|r_1| - 1$ of r_1 is the same as the prefix of length of $|\overline{r_2}| - 1$ of $\overline{r_2}$ and is the longest overlap between that two strings. So the overlap length between r_1 and r_2 is 29. After finding all overlaps between all reads, the overlap graph in Figure 2.8 will be obtained.

When overlapping reads, *minimum overlap length* is defined as a threshold to control the overlap: only overlaps longer than the minimum length are considered. The structure

```
Read  r₁   :     CACGTGCTGCCGATAATGTGGTTTCAGTAT
Read  r̄₂   :      ACGTGCTGCCGATAATGTGGTTTCAGTATA
Read  r̄₃   :       GTGCTGCCGATAATGTGGTTTCAGTATAAT
Read  r₄   :        GCTGCCGATAATGTGGTTTCAGTATAATGA
Read  r̄₅   :         CTGCCGATAATGTGGTTTCAGTATAATGAG
Read  r₆   :          GCCGATAATGTGGTTTCAGTATAATGAGGG
Read  r₇   :            ATAATGTGGTTTCAGTATAATGAGGGCAAT
Read  r̄₈   :      TGCTGCCGATAATGTGGTTTCAGTATACGT
Read  r₉   :       GCTGCCGATAATGTGGTTTCAGTATACGTA
Read  r̄₁₀  :        TGCCGATAATGTGGTTTCAGTATACGTATG
```

Read Length l : 30bp
Overlap Length : 26bp

Figure 2.8: Overlapping reads
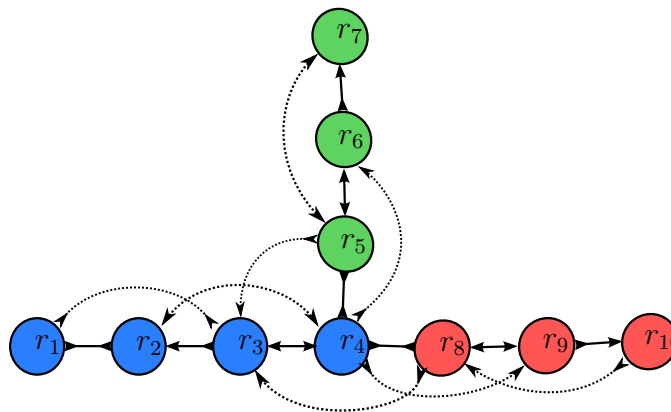
Figure 2.9: Overlap graph with 10 reads.

of the overlap graph is significantly influenced by the length of the overlapping. The smaller the overlap length, the higher the number of overlaps between reads and the overlap graph will produce more contigs because of the increase of density. However the overlap graph will be tangled and more mistakes will be made if we assemble reads with a very short overlap. On the other hand, if we choose a larger overlap length, the graph will become simpler, and the number of mistakes will decrease. However the sensitivity of the algorithm will change since the more overlaps we abandon the more information we lose. So choosing a good overlap length is very important for an assembly algorithm.

2.4.2   Transitive edge

In the overlap graph, if there are three reads r_1, r_2 and r_3 (Figure 2.10) connected by edges (r_1, r_2) , (r_2, r_3) and (r_1, r_3), a triangle can be found in the overlap graph which is made from the three edges among r_1, r_2 and r_3. We call this triangle *transitive triangle*. The edge (r_1, r_3) is called *transitive edge*. This transitive triangle (edge) contains redundant information. In the above case, edge (r_1, r_3) can be removed to make the graph simpler without causing any loss of information, see Figure 2.10. There are several algorithms to remove transitive edges [3, 24].

ACGTGCTG

CACGTGCTGC

CACGTGCT   CGTGCTGC
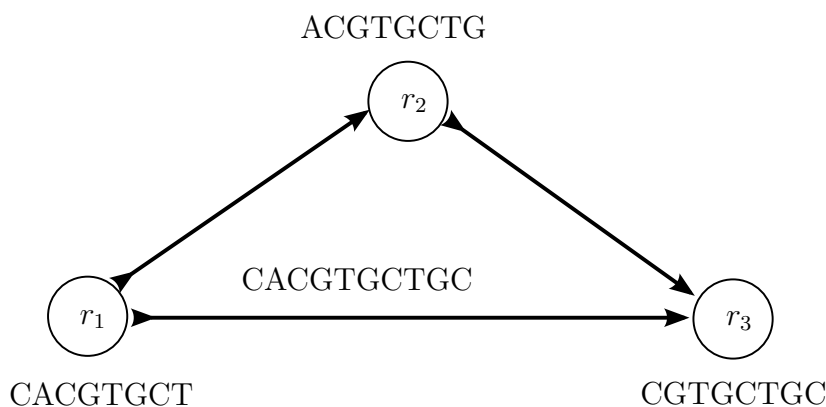
Figure 2.10:   Transitive edge.

2.5   De Bruijn graph

The de Bruijn graph [31, 30] is another popular data structure used for genome assembly. It is quite similar with the overlap graph in that vertices represent sequences of nucleotides while edges represent the overlap between sequences. However the vertices in the de Bruijn graph correspond usually to shorter strings than the edges in the overlap graph. A *k-mer* is any string of k letters. A *k-mer* de Bruijn graph has as vertices all k-mers in the reads. Another difference between the overlap graph and the de Bruijn graph is in the length of the overlap. In an overlap graph, vertices can have any length of overlap longer than the minimum overlap. However in de Bruijn graph, all k-mers are linked with a fixed $k - 1$ overlap. A example of the de Bruijn graph is shown in Figure 2.11.

This is a 3-mer de Bruijn graph of the reads ACCGTCAGAAT and ACCGTGAGAAT. All edges represent an overlap of 2bp.
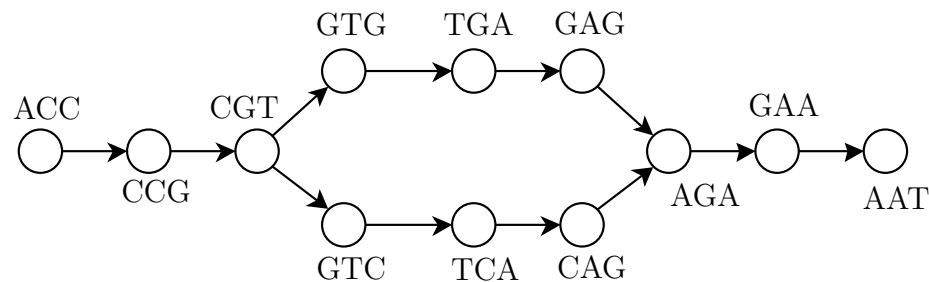
Figure 2.11: De Bruijn Graph

2.6 Metagenome assembly

A variety of software tools are available for analysing next-generation sequencing data [34]. Alignment and assembly are particularly interesting problems. Many assemblers were designed in the past several years, such as Velvet [40], ABySS [37], SOAPdenovo [17], SGA [36]. However, despite the many improvements in single genome assembly, assembly of a metagenomic sequence is still a nontrivial task.

The advent of next generation sequencing (NGS) has allowed an explosion in sequencing of individual genomes, and started a revolution in a new area: metagenomic sequencing and analysis. The increased throughput and decrease in costs of sequencing, coupled with additional technological advances have transformed the landscape of DNA research and related areas ([33]). With NGS, scientists could sequence a whole microbial community or a sample obtained directly from the environment. So the task in metagenomic assembly can be described as assembly of multiple species in a microbial community [26]. The goal for any metagenome sequencing project is the full characterization of a community, and scientists are trying to understand:

- the composition and structure of a community, including the relative abundance of various species;

- genetic contribution of each member of the community, including the functional capacity;

- intra-species or intra-population difference of genes.

The remaining part of this section describes several popular metagenomic assemblers.

2.6.1 MetaVelvet

MetaVelvet [26] is extended from Velvet [40], a well known single-genome assembler using the de Bruijn graph. MetaVelvet consists of four major steps: [a] Construction of a de Bruijn graph from the input reads. [b] Detection of multiple peaks on k-mer frequency distribution. [c] Decomposition of the constructed de Bruijn graph into individual subgraphs. [d] Assembly of contigs and scaffolds based on the decomposed subgraphs.

In Step [a], MetaVelvet constructs the main de Bruijn graph by Velvet from a given set of mixed sequence reads generated from multiple species. In Step [b], MetaVelvet calculates the histogram of k-mer frequencies and detects multiple peaks on the histogram. Those peaks indicates the different frequencies of different species. The expected frequencies of k-mer occurrences in a single-genome follow a Poisson distribution. Because the frequency of different genome species can be regard as independent, the expected k-mer frequencies in metagenome assembly follow a mixture of several separated Poisson distributions. After that, MetaVelvet draws histograms of k-mer frequencies from a mixture of Poisson distributions and detects those condensed regions as peaks on the chart. Furthermore, MetaVelvet clusters nodes into different groups based on their frequency, and maps each group to a peak in the chart. In Step [c], MetaVelvet merges those nodes belonging to the same peak as a subgraph and then removes those edges linked between different subgraphs. In step [d], MetaVelvet builds contigs and scaffolds based on the decomposed subgraphs using Velvet functions.

2.6.2 Meta-IDBA

Meta-IDBA [28] is extended from IDBA [27], also a single-genome assembler using the de Bruijn graph. The idea of Meta-IDBA is simple but practical that it iterates on a range

of k values from $k = k_{min}$ to $k = k_{max}$ and maintains accumulated de Bruijn graph h_k at each iteration. On each iteration, Meta-IDBA adds or removes some edges to make the graph more reliable.

Meta-IDBA defines two types of branches in de Bruijn graph to help readers understand its process: *sp-branches, cr-branches*. Sp-branches indicates those branches caused by the polymorphism of similar subspecies which consist of very similar sequences with a few variations and each variation introduces a branch in the de Bruijn graph. Another source of branches is due to the common or similar genomic regions, shared by different species which are called cr-branches.

There are 4 steps in Meta-IDBA. Initially it constructs a de Bruijn graph from sequencing reads. Each simple path in the de Bruijn graph might represent a contig of the genome. As there are some sequences appearing in multiple species, the de Bruijn graph of reads from different species are interconnected by cr-branches. In the second step, Meta-IDBA divides the de Bruijn graph into many small connected components by removing cr-branches. Meta-IDBA assumes that one genome sequence is more similar with one genome sequence from the same species or subspecies than the one from a different species. So Meta-IDBA will group those highly similar sequences, as they may be from the same species or close subspecies, to obtain separate components in the de Bruijn graph. These components are then merged into bigger components, which represent longer consensus contigs using paired-end information. In the last step of Meta-IDBA, each component is transformed to a multiple alignment of similar contigs of different subspecies.

Based on Meta-IDBA, the authors updated another version named IDBA-UD [29]. IDBA-UD extends and enhances the idea of variable thresholds of Velvet-SC [6] to filter out erroneous contigs. To cater to very extreme sequencing depths, instead of using a global average of the multiplicity of all k-mers as the threshold, they adopt variable 'relative' thresholds depending on the sequencing depths of their neighbouring contigs tend to be erroneous.

2.6.3   Omega

Omega [11] is a newly released metagenomic assembler, also based on SAGE [14]. Since our new metagenomic assembler MetaSAGE is also based on SAGE, Omega is in some sense our direct competitor. Omega is therefore based on the overlap graph and cost flow analysis, inherited from SAGE. Omega has four steps in logic. First, it constructs the overlap graph from a set of reads. To improve the efficiency of the construction, Omega builds the hash table for all reads in advance. Second, Omega makes some reductions on the overlap graph, including removing some dead-ends and bubbles, which are caused by the errors in the reads. Third, it estimates the copy count of each edges in the overlap graph by the minimum cost flow theory. In the end, it improves contigs and scaffolds from the overlap graph using pair-end information.

2.6.4   Genovo

Genovo [16] is another de novo sequence assembler that discovers likely sequence reconstructions under a generative probabilistic model. Genovo's approach is different from the algorithms we mentioned above. First, it introduces a probabilistic model of a read set. The model associates a probability to each possible list of sequences that could have given rise to this readset. The model simulates the process of constructing a number of sequences and sampling reads from the reference genome. So the assembly of sequences can be seen as a reasonable summary of the read set. The model estimates the size of genome automatically without setting any parameter in advance. Second, Genovo describes an algorithm that reconstructs a likely assembly from a read set. The algorithm accomplishes this by seeking the most probable assembly iteratively, moving between increasingly likely assemblies via a set of moves designed to increase the probability of the assembly. After the move, reads are rearranged into a more compact assemblies, and they still represent the whole read set. Crucially, the moves are not all greedy, thus allowing some undoing of potential erroneous moves. The process is iterated until no reasonable move is available. At this point, the assembly is regarded with the best probability. This is the assembly that best trades off the compactness and read set representation from

among the assemblies that the algorithm explored, thus being a likely candidate for the true set of sequences that generated the reads.

Unlike the other methods, Genovo does not throw away reads. So it is able to extract more information from the data, especially when works on low-abundance sequences. Another special point in Genovo is its joint denoising. Genovo does not make a decision about the error correction until the end of assembling process which is hopefully leading to a better assembly. However it will be at a higher computational cost.

Chapter 3

MetaSAGE

In this chapter, our new metagenomic assembler MetaSAGE is introduced. MetaSAGE is based on SAGE [14], a well designed single genome assembler using the overlap graph. As SAGE has good performance on single genome assembly, we preserve the general structure of SAGE and add several metagenomic-specific changes concerning error corrections and node-splitting in the overlap graph. Compared with other genome assemblers, MetaSAGE has three main improvements. First, MetaSAGE does not build the overlap graph directly from the entire collection of reads. Instead, it removes transitive edges while building the overlap graph, which significantly reduces the amount of memory used. Second, MetaSAGE uses minimum cost flow theory to estimate the copy count of each edge and assembles edges based on their copy count. Third, in order to adapt to the metagenomic assembly, MetaSAGE not only does graph trimming, splitting and simplification used in typical single genome assemblers, but also splits edges from different species according to their coverage. This process extends the contigs obtained from the overlap graph as well as reduces the number of misassemblies.

In this chapter an overview of MetaSAGE is given, followed by detailed description of all steps of MetaSAGE. In order to maintain readability, the steps that are inherited from SAGE are descibed as well, although in less details.

Genome assembly is always preceded by error correction of the reads. SAGE uses the RACER program [15] for this purpose. We have adapted RACER as well for metagenome assembly and the new program, MetaRACER, is described also in this section.

3.1   Overview of MetaSAGE

MetaSAGE has five steps. First, input reads are corrected by MetaRACER which is modified from the RACER program [15]. Second, a bidirected graph is built from the input dataset using a hash table. Third, the overlap graph is simplified and edges coming from different species are split. Then MetaSAGE makes the copy count estimation based on the minimum cost flow theory and extracts contigs from the overlap graph. In the last step, MetaSAGE extracts scaffolds from the overlap graph using mate-pair information. In the following sections, we describe these steps one by one. A flowchart of MetaSAGE is shown in Figure 3.1.
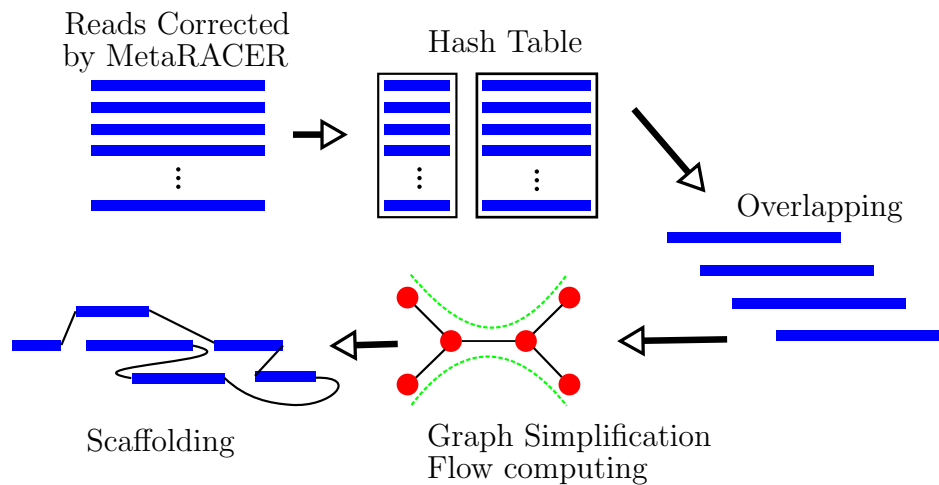
Figure 3.1: Flowchart of MetaSAGE.

3.2   Error correction

3.2.1   RACER and MetaRACER

The main idea of the error correction methods is as follows. NGS provides reads in high coverage which implies that each position of the genome is sequenced multiple times. Since the errors happen in a minority of case, error correction software will use the majority reads to correct the minority. RACER counts k-mers from the reads and stores them into a hash table. For each k-mer, RACER counts all possible nucleotides on both

sides. Then a threshold t is used to check whether one position is correct. The criterion is that if a nucleotide a following a k-mer is counted over t times, it will be regarded as correct, otherwise wrong.

Figure 3.2: An example of error correction.

RACER gives good performance on reads from a single genome. However, when it comes to the metagenomic data, the test results are far from satisfactory. Because the metagenomic data is diverse not only with respect to species but also with abundance, the correction criterion mentioned above no longer works because it will miss-correct very many reads that have low coverage. So modifications are needed to make RACER less aggressive in detecting errors. In order to preserve the information from genome in low coverage, we need two thresholds, t_c and t_e. A count larger than t_c indicates a correct position, while one below t_e indicates an error. We keep $t_c = t$ from RACER and set $t_e = 1$, that is, the strictest condition for correcting an error. The new error correction software is called MetaRACER. In order to give a detailed evaluation of MetaRACER, in the following subsection, we introduce the evaluation methods from the study of Molnar and Ilie [23] and perform several tests to compare the correction using RACER with that of MetaRACER.

3.2.2 Evaluation of error correction results

According to the study of Molnar and Ilie [23], a dataset has two main parameters that can be improved by the correction programs: 'depth of coverage' (the number of times each base is covered on the average) and 'breath of coverage' (the proportion of the genome that is covered). Each parameter can be evaluated using the proportion of

correct reads or correct k-mers. Therefore, four criteria are obtained to evaluate the performance of the error correction software: READDEPTHGAIN, KMERDEPTHGAIN, READBREATHGAIN and KMERBREATHGAIN. These four criteria indicate how much the depth or breadth as given by correct reads or k-mers, respectively, increases after the error correction. Before understanding these four criteria, we need to introduce several notions we used.

Suppose we have a read set $R = \{r_i | 1 \leq i \leq n\}$ generated from the reference genome $\mathscr{G}$. A read r_i is considered as "correct" if it is a substring of $\mathscr{G}$, and "erroneous" otherwise. Based on this definition, we can make a binary classification on the dataset $\mathscr{G}$. We can obtain four classes *true-positive(TP), true-negative(TN), false-positive(FP)* and *false-negative(FN)*, where TP is the number of reads that are erroneous before correction and correct after correction, TN is number of reads correct both before and after correction, FP counts reads correct before and erroneous after correction and FN counts reads erroneous both before and after correction. Then we can define

$$\text{READDEPTHGAIN} = \frac{TP - FP}{P} = \frac{TP - FP}{TP + FN} \ .$$

Concerning breadth, TP becomes the number of reads that are not covered before correction but covered afterwards; with TN, FP and FN correspondingly defined. We have then READBREATHGAIN$= \frac{TP-FP}{TP+FN}$. We refer the reader to [23] for more details.

KMERDEPTHGAIN and KMERBREATHGAIN are similar to the above cases where TP, TN, FP and FN represent the corresponding value for k-mers instead of reads; again, see [23].

We compare RACER and MetaRACER using the above mentioned criteria by testing the two programs on the four read sets used also later for testing metagenome assemblers; the datasets are artificial and are described in detail in Chapter 4. The results are listed in Table 3.1. We can see that metaRacer is less sensitive but also less aggressive than RACER. RACER has higher performance with respect to READDEPTHGAIN, READBREADTHGAIN, and KMERDEPTHGAIN for all datsets; the difference is particularly large for the "depth" measures. However, for KMERBREADTHGAIN, RACER destroys a

lot more k-mers than MetaRACER. Note that both programs have only negative values for KmerBreadthGain on all datasets, which is expected behaviour, already noted as such in the original study of Molnar and Ilie [23].

The higher the threshold, the higher the number of corrections applied in the read sets and so RACER miss-corrects many reads in low coverage, destroying the read diversity. However, it is expected that MetaRACER can keep more information of those reads in low coverage which is essential for metagenomic assembly.

The direct comparison between RACER and MetaRACER is insufficient to decide which one is better for metagenome assembly. It gives only some insight in their expected performance, that was proven correct by the superior behaviour of MetaSAGE when used on MetaRACER-corrected reads.

Data	RACER	MetaRACER	Data	RACER	MetaRACER
ReadDepthGain			ReadBreadthGain		
orderLevel	97.13	53.39	orderLevel	46.79	36.01
familyLevel	91.12	41.30	familyLevel	42.81	32.31
genusLevel	96.09	60.70	genusLevel	44.34	36.29
speciesLevel	96.64	53.40	speciesLevel	45.18	34.60
Data	RACER	MetaRACER	Data	RACER	MetaRACER
KmerDepthGain			KmerBreadthGain		
orderLevel	98.00	64.17	orderLevel	-9.39	-3.94
familyLevel	94.20	49.44	familyLevel	-3.01	-2.71
genusLevel	97.26	71.18	genusLevel	-10.15	-4.92
speciesLevel	97.81	65.07	speciesLevel	-34.11	-9.60

Table 3.1: Comparison between RACER and MetaRACER using the four measures.

3.3   Overlap graph construction and composition

The overlap graph is composed of overlapping read pairs. If there are n reads in the input data set, comparing all possible pairs of reads will take $O(n^2)$ time which is too time-consuming. What we do is build a hash table for fixed-length prefixes and suffixes

of each read r_i and its reverse complement $\bar{r}_i$. Then we search for these prefixes and suffixes as substrings of the reads in the hash table instead of making an $O(n^2)$ search. MetaSAGE sets a threshold named *minimum overlap* to control the minimum overlap length in the graph. If two reads r_1 and r_2 have an overlap larger than *minimum overlap*, a suffix or prefix of r_1 can be found that matches a suffix or prefix of r_2 at length larger than the *minimum overlap*. Therefore, in order to find overlap efficiently, MetaSAGE stores a prefix and suffix of length $h = min\{minimum\ overlap, 64\}$ for each read and its reverse complement. We choose 64 here because most of the reads generated by NGS are around 100 bp. MetaSAGE only stores prefixes or suffixes of length up to 64 bp and represents them with two 64-bit integers. In the following part, we introduces the encoding methods and the process of building the hash table.

3.3.1   Hash table

For each read r_i and its reverse complement $\bar{r}_i$, MetaSAGE stores their suffixes and prefixes of length h into the hash table. Supposing there are n reads, $2n$ suffixes and $2n$ prefixes will be stored into the hash table. For efficiency, MetaSAGE encodes each base with 2 bits in the array (A=00, C=01, G=10, T=11). So the $4n$ suffixes and prefixes will be stored into an array of $8n$ 64-bit integers and each suffix or prefix is represented as 2 64-bit integers. The size of hash table is set to a prime number $p > 8n$ to reduce the number of hashing collisions. The algorithm is shown in Algorithm 1.

Algorithm 1: Algorithm for building hash table

   input : Read set $R = \{r_1, r_2, \ldots, r_n\}$ and minimum overlap length $minOverlap$
   output: Hash table $hashTable$
1 Create an empty $hashTable$
2 $h \leftarrow min\{64, minOverlap\}$
3 **for** *each read $r \in R$* **do**
4   compute the prefix and suffix of r and $\bar{r}$
5   Store them in the $hashTable$
6 **end**
7 **return** $hashTable$

3.3.2   Inserting edges

After the hash table is built, we start to build the overlap graph. As mentioned before, each read is a vertex in the overlap graph and we insert an edge between two vertices if they have overlap longer than the *minimum overlap*. Suppose the read length is l. For each read r in R, MetaSAGE will scan it from the beginning to the end with a "window" in size of *minimum overlap*, see Figure 3.3. For every subsequence caught by the window, MetaSAGE will check whether it appears in the hash table. If there is a subsequence s (caught by the window) on the read r being found in the hash table, supposing it hits the prefix of read r_h, we will know that r and r_h share an overlap to be inserted into the overlap graph. Then MetaSAGE will continuously enlarge the size of the window, including the following bases on both reads if they overlap, to obtain the longest overlap between r and r_h. Until the window reaches the end of one sequence or no more bases can be added, the longest overlap s_l between r and r_h is obtained. Finally an edge (r, r_h) can be inserted into the overlap graph along with the longest overlap s_l between them. An example is shown in Figure 3.4. In this example, we suppose the hash size is 10 bp, then the window size is set as 10. At first, we move the scanning window on r from the left to the right and search it in the hash table. Then we obtain a hit which is in green color in the figure. We keep on enlarging the size of the window, and includes all matching bases, shown with blue color in that figure. In the end, we have the largest overlap as both of the blue and the green sequences.
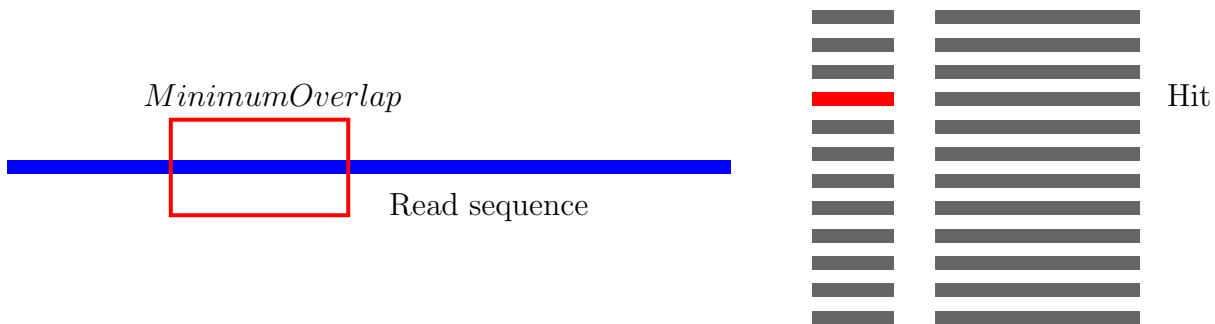
Figure 3.3: Minimum Overlap Window.

Read r :   **TCAGA****CGCTAATGCA****GCCATTATTAGAACACAGAT**
Read r_h :   **CGCTAATGCA****GCCATTATTAGAACACAGAT****GCTAA**

Figure 3.4: Overlap in an edge.

3.3.3   Removing transitive edges

Transitive edges contain redundant information making the graph more complex. There is an algorithm to reduce transitive edges in linear time given by Myers [24]. Myers' algorithm performs an iteration over all edges in the graph $G = (V, E)$. For each edge e it marks all its neighbours w as *inplay*. Then it goes on checking all neighbours of w with an increasing order of the length of the string spelled by the edge (v, w), marking those neighbours as *eliminated* if they are already in status *inplay*. In the end, the program removes all edges (v, x) with x is marked with *eliminated*. Reusing the set of reads shown in Figure 2.7 in Chapter 2, the overlap graph is shown again for convenience in Figure 3.5, and then in Figure 3.6 after removing transitive edges. The algorithm is shown in Algorithm 2.

3.3.4   Compressing paths

We notice that in the reduced graph, some nodes have only one in-edge and one out-edge, such as nodes r_2, r_3, r_5, r_6, r_8 and r_9 in Figure 3.6. A path consisting only of such vertices except the beginning and the end is a simple path and can be compressed so that only the initial and final vertices are kept, the rest are removed, and the obtained edge stores the entire sequence spelled by all the reads involved: such a new edge is called a *composite edge*. An edge that is not composite is called *simple edge*. Compressing simple paths to obtain composite edges for the graph in Figure 3.6 produces the graph in Figure 3.7 that contains four vertices and three composite edges.

3.3.5   Bubbles and dead-ends remove

No error correction software can correct all errors in genome reads. Even after error correction by MetaRACER, reads are not error free in regarding of the huge size and the unexpectedness of the genome sequence. Most errors in genome reads will cause
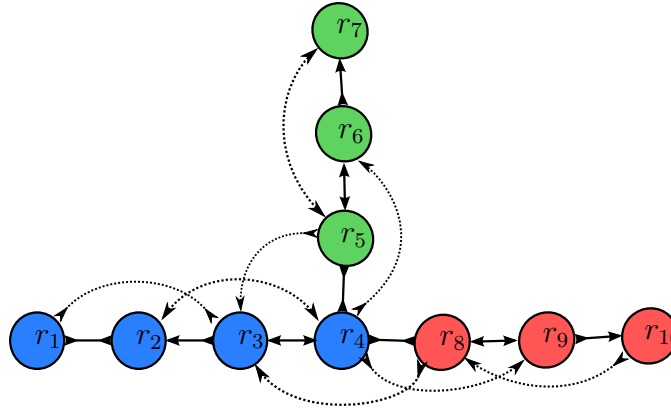
Figure 3.5: Before removing transitive edges.
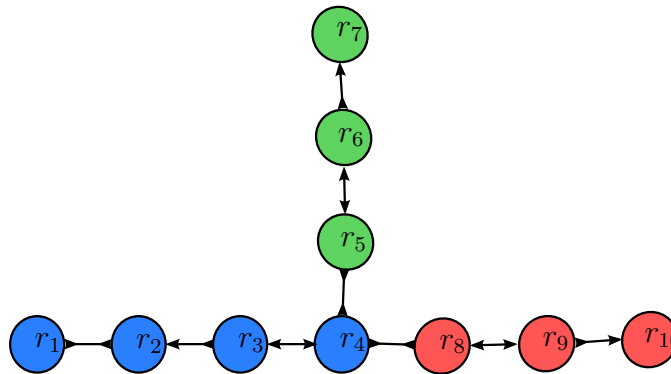
Figure 3.6: After removing transitive edges.
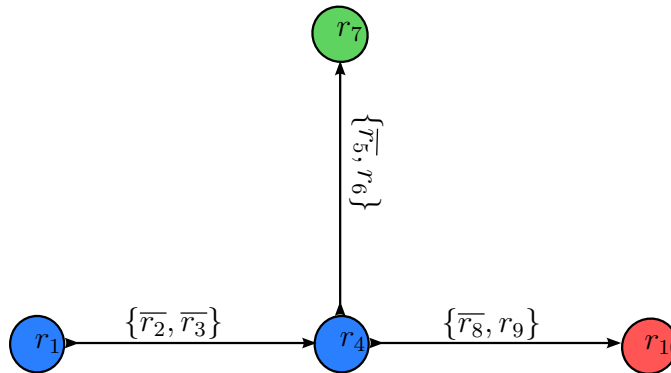
Figure 3.7: Composite graph.

dead-ends in the overlap graph. Usually dead-ends are short because it is very unlikely that a sequencer generates many errors in the same region. In MetaSAGE, a composite edge containing more than 5 reads will not be considered as a dead-end. Assume that the sequencing of each read is independent, and the error rate on one base is p. If a composite edge has 5 reads on it, the possibility that all these 5 reads contain an error

---

Algorithm 2: Linear time transitive edge reduction

    input : Overlap graph $G = (V, E)$

    output: Transitively reduced overlap graph

1 **for** *each* $v \in V$ **do**

2 $mark[v] \leftarrow vacant$ **for** *each* $(v, w) \in E$ **do**

3 | $reduce[(v, w)] \leftarrow false$

4   **end**

5 **end**

6 **for** *each* $v \in V$ **do**

7   **for** *each* $(v, w) \in E$ **do**

8 | $mark[w] \leftarrow inplay$

9   **end**

10   **for** *each* $(v, w) \in E$ *in increasing order of length of the string spelled* **do**

11   **if** $mark[w] \leftarrow inplay$ **then**

12   **for** *each* $(w, x) \in E$ *in increasing order of length of the string spelled* **do**

13   **if** $mark[x] = inplay$ **then**

14 | $mark[x] \leftarrow eliminated$

15   **end**

16   **end**

17   **end**

18   **end**

19   **for** *each* $(v, w) \in E$ **do**

20   **if** $mark[w] = eliminated$ **then**

21 | $reduce[(v, w)] \leftarrow true$

22   **end**

23 $mark[w] \leftarrow vacant$

24   **end**

25 **end**

26 **for** *each edge* $e \in E$ **do**

27   **if** $reduce[e] = true$ **then**

28 | Remove e from E

29   **end**

30 **end**

31 **return** G

---

on the same position is $(\frac{p}{3})^5(1 - p)^{5(l-1)}$, where l is the read length. According to Table 2.1, the error rate of NGS is less than 2% and the read length l is around 100. So the possibility of a composite edge containing 5 or more reads being a dead-end is very small; about 6×10^{-16} according to our formula. Figure 3.8 (from [12]) shows that how a dead-end appears in the overlap graph. In that graph, reads r_{11}, r_{12}, r_{13}, and r_{14} are dead-ends

caused by errors in reads.

Figure 3.8: Dead-end removing.
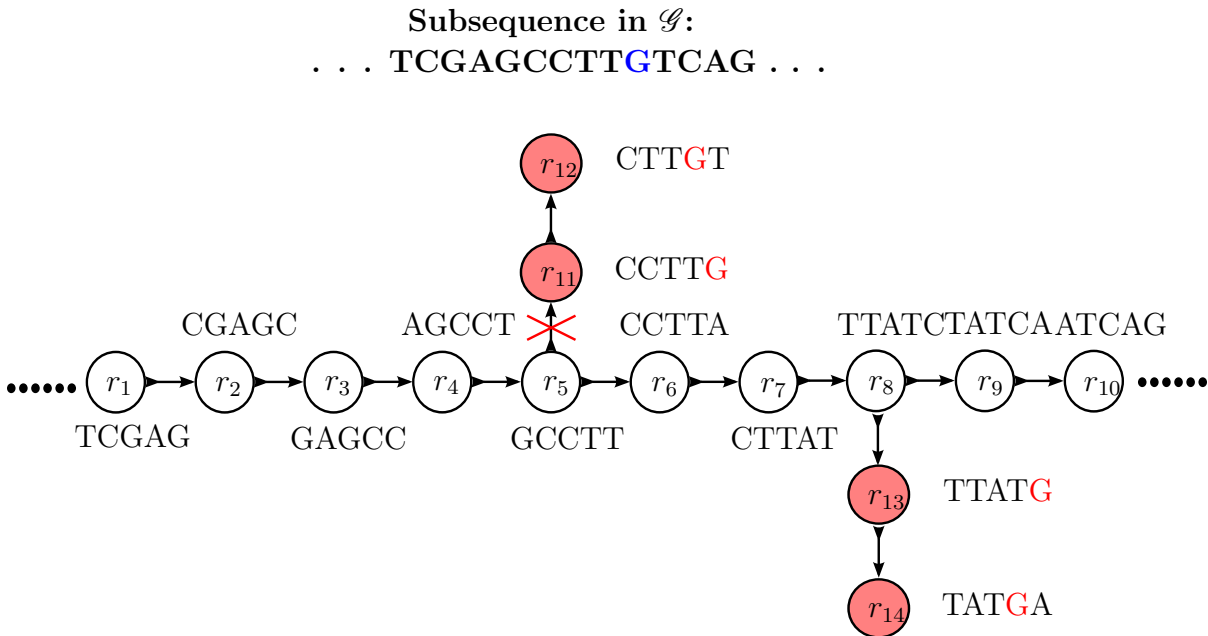
Apart from dead-ends, another complication in the overlap made by errors is the bubble. A bubble is made from two (or more) edges sharing both the beginning and the end on the overlap graph. Bubbles occur due to various reasons. Some of them are caused by errors in reads while some of them occur because of the repeats in reference genome. We only reduce bubbles caused by errors. Edges of low coverage will be considered as erroneous and set as removable in bubbles. Figure 3.9 gives an example of bubble made by errors in reads. The reference genome sequence is ACGCGTATCCGGTATC on this area. However, the edge in the above spells ACGCGTAGCCGGTATC. In this case MetaSAGE will detect the coverage of both edges, and keep the edge with higher coverage because the higher coverage has lower possibility to be erroneous.

The algorithms for removing dead-ends and bubbles [12] are shown in Algorithm 3 and Algorithm 4.

Subsequence in $\mathscr{G}$:

. . . ACGCGTATCCGGTATC . . .

Figure 3.9: Bubble removing.

Algorithm 3: Remove Dead Ends

input : Overlap graph $G = (V, E)$

output: Overlap graph after removing dead-ends

1 **for** *each node $u \in V$* **do**
2      $inDegree \leftarrow 0$
3      $outDegree \leftarrow 0$ **for** *each neighbour v of u* **do**
4   **if** *(u, v) has more than 5 reads in it* **then**
5              $inDegree \leftarrow 0$
6              $outDegree \leftarrow 0$
7   break
8   **end**
9   **if** *(u, v) is an in-edge of u* **then**
10             $inDegree \leftarrow inDegree + 1$
11   **end**
12   **else**
13             $outDegree \leftarrow outDegree + 1$
14   **end**
15   **end**
16   **if** *$inDegree = 0$ and $outDegree > 0$* **then**
17   Remove u and all its edges from G
18   **end**
19   **if** *$inDegree > 0$ and $outDegree = 0$* **then**
20   Remove u and all its edges from G
21   **end**
22 **end**
23 **return** G

Algorithm 4: Remove Bubbles

    input : Overlap graph $G = (V, E)$

    output: Overlap graph after removing bubbles

1 **for** *each pair of edge $e = (u, v)$ and $e' = (u, v) \in G$ with the same endpoint* **do**

2   **if** *string spelled by $e \approx$ string spelled by e'* **then**

3   **if** *number of reads in $e \leq \frac{1}{2}$ number of reads in e'* **then**

4   Remove e from G

5   **end**

6   **else if** *number of reads in $e' \leq \frac{1}{2}$ number of reads in e* **then**

7   Remove e' from G

8   **end**

9   **end**

10 **end**

11 **return** G

3.4   Chimeric node splitting

After dead-end removing and bubble reducing, we have the step of splitting of reads coming from different species. However, it is hard to tell apart whether two sequences are coming from different species. Our approach is to use the coverage difference in different genomes. Because genome sequencers generate reads uniformly on the reference genome, if one genome is more abundant than others in the sample, sequences from that genome are hopefully in a higher coverage. Using the genome coverage, we can tell apart two reads or two edges in the overlap graph that are coming from different reference genomes if they have a big difference in their coverage. That is not to say that two reads or two edges having the same coverage come from the same reference genome.

Similar to Equation 2.1, the coverage of an edge can be defined as follows. Suppose there is a composite edge $e = (u, v)$, and the reads on this edge are $r_1, r_2, \ldots, r_n$. If the read length is l and length of edge e is L, then the coverage of this edge is

$$coverage(e) = \frac{nl}{L} \tag{3.1}$$

After computing the coverage of each edge, MetaSAGE will split intersections consisting of edges of mixed coverage, see Figure 3.10. A chimeric node is the node whose outgoing edges and incoming edges have mixed coverage. By connecting edges with sim-

ilar coverage, chimeric nodes can be reduced into different paths; see Figure 3.10 for an example.
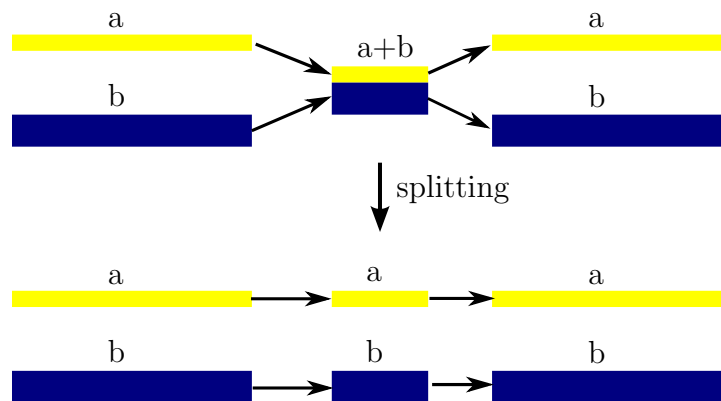
Figure 3.10: Chimeric node.

However, in metagenomic assembly, due to the mixing of different species in different abundance, chimeric nodes will not be as simple as the case shown in Figure 3.10. When dealing with those complex chimeric nodes, we split pairs of edges gradually. Suppose there is a chimeric node like the one shown in Figure 3.11. On the left side, there is a set of incoming edges $\{u_1, u_2, \ldots, u_8\}$ and another set of outgoing edges $\{v_1, v_2, \ldots, v_6\}$ on the right side. In the first figure (u_3, v_3) and (u_3, v_5) are two pairs of edges having similar coverage c, coloured in yellow. Those edges in black color are in different coverage. In this case, we cannot decide how to split this chimeric node because the merging of (u_3, v_3) and (u_3, v_5) are both applicable. In the second figure, only (u_3, v_3) are coloured in yellow which means they have some similar coverage that is different from those edges coloured in black. So we can split (u_3, v_3) from the original graph and merge them into a new edge. In the algorithm, the similarity of coverage between edges will be checked by a threshold; two edges are considered to have similar coverage if the coverage difference is below the threshold. Precisely, if the coverage of an edge is c_e, then, for two edges e_1 and e_2, the coverage difference was computed as $2\left|\frac{c_{e_1}-c_{e_2}}{c_{e_1}+c_{e_2}}\right|$ and the experimentally determined threshold was 0.05. The algorithm of splitting chimeric node is shown in Algorithm 5.

can not merge more than one possible edge pairs (u_3, v_3) and (u_3, v_5)

(u_3, v_5) will be merged
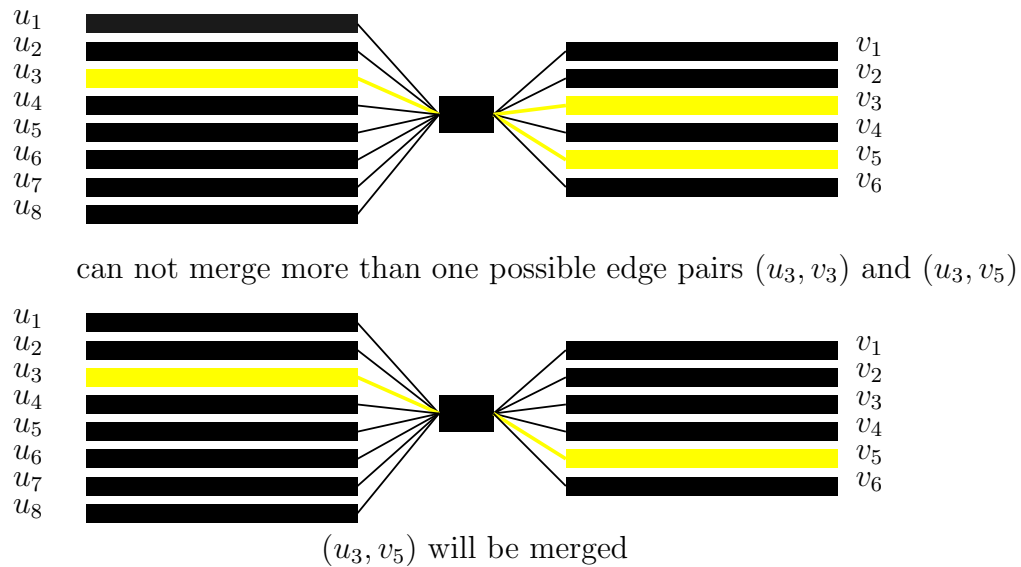
Figure 3.11: Chimeric node splitting.

Algorithm 5: Chimeric node Splitting

input : Overlap graph $G = (V, E)$

output: Overlap graph after chimeric node splitting

1 **for** *each node n in G* **do**
2 $\quad$ Compute the coverage of all incoming and outgoing edges contacting on this node.
3 **end**
4 **for** *each node n G* **do**
5 $\quad$ **for** *each edge e_{in} incoming to node n* **do**
6 $\quad\quad$ **for** *each edge e_{out} outgoing from node n* **do**
7 $\quad\quad\quad$ **if** $difference(coverage(e_{in}), coverage(e_{out})) \leq threshold$ **then**
8 $\quad\quad\quad\quad$ **if** *no other edges with similar coverage* **then**
9 $\quad\quad\quad\quad\quad$ Mark edge e_{in} and edge e_{out}
10 $\quad\quad\quad\quad$ **end**
11 $\quad\quad\quad$ **end**
12 $\quad\quad$ **end**
13 $\quad$ **end**
14 $\quad$ Merge all marked pairs of edges
15 **end**
16 **return** G

3.5 Copy count estimation

Even after the bubble reducing and chimeric node splitting, the overlap graph is still a criss-cross graph because the genome sequence contains repeats itself. To assemble genomes well, assemblers must know how many times one piece of a subsequence appears in the original reference genome sequence. The number of times a certain subsequence appears is called its copy count. In MetaSAGE, the copy count is estimated by the combination of the minimum cost flow model and the convex cost function $c_e(k)$. The convex cost function represents the maximum likelihood of the sequence e appearing k times in the reference sequence. Details of this process will be described in the following sections.

3.5.1 Minimum cost flow

The minimum-cost flow problem is to find the cheapest possible way of sending a certain amount of flow through a flow network. Given a network $G = (V, E)$ and the cost $c(u, v)$ on each edge (u, v) in this graph, the minimum cost theory wants to produce the flow $f(u, v)$ on each edge such that the total cost $\sum_{(u,v) \in E} f(u, v) c(u, v)$ is minimum subject to the following constraints:

Capacity constraint

$f(u, v) \le flowUpperBound(u, v)$

Skew symmetry

$f(u, v) = -f(v, u)$

Flow conversation

$\sum_{w \in V} f(u, w) = 0$ all $u \ne s, t$ and

Required flow

$\sum_{w \in V} f(s, w) = d$ and $\sum_{w \in V} f(w, t) = d$

3.5.2   Cost function

The maximum likelihood genome assembly was proposed by Medvedev et al. [19, 20, 21, 22]. This algorithm uses the bidirected network flow to model double-stranded DNA structure and aims to assemble a genome that is most likely the source of a given set of reads. In genome assembly, each subsequence is supposed to be assembled the same number of times as it appears in the reference genome. However the number of times each subsequence appears in the reference genome is unknown. If a subsequence is present multiple times in the reference genome sequence, reads from that sequence are more likely to be sampled more often than the other subsequences in the reference genome. The maximum likelihood genome assembly is looking for the most likely copy count of each subsequence. A convex min-cost function $c_e(k) : \mathbb{N} \to \mathbb{R}$, is associated with every edge e reflecting the likelihood that the sequence represented by e appears k times in the genome for each $k \leq 1$. The goal is to compute a flow function f that minimizes $\sum_e c_e(f(e))$, where the flow through edge e is $f(e)$. The value we compute for $f(e)$ is the approximation of the number of times the sequence associated with e occurs in the genome.

Consider a set R of n reads from a genome $\mathcal{G}$ of length L. Let x_r denote the times that read $r \in R$ appears in the dataset R and c_r denote the times that read r appears in the genome $\mathcal{G}$. The actual copy counts c_r are unknown. What is known is the observed values x_r. Then, if n reads are sampled uniformly from $\mathcal{G}$, the probability that r is sampled x_r times is

$$Prob(Freq(r) = x_r) = \binom{n}{x_r} \left(\frac{c_r}{L}\right)^{x_r} \left(1 - \frac{c_r}{L}\right)^{n-x_r} \tag{3.2}$$

Here, $Freq(r)$ denotes the number of times read r appears in the dataset R. This is also the likelihood that the copy count of read r is c_r, given the observed values x_r. This likelihood needs to be maximized, which is the same as minimizing the negative log of Equation 3.2.

$$-log(Prob(Freq(r) = x_r)) = K + c_r(c_r)$$

where,

$$K = -log\binom{n}{x_r} + nlogL$$

and

$$c_r(c_r) = -x_r logc_r - (n - x_r)log(L - c_r) \tag{3.3}$$

Note that K does not depend on the number of times that the read r appears in the genome, so it can be regarded as constant for our genome copy count estimation here. Equation 3.3 is used as the convex function.

There are several algorithms for solving the minimum cost flow problem in directed graphs. In MetaSAGE, software called CS2 [9] is used. We refer the reader to [20] for all details on the flow approach.

3.5.3   Flow bounds

In SAGE [14], the authors used a generalization of the A-statistics of Myers [24] to estimate the upper bound and lower bound of copy count before they calculate the minimum cost flow. The A-statistics were based on the assumption that the coverage is uniform. One of the most important differences between single genome and metagenome assembly is that the coverage of the latter in *not* uniform and therefore the A-statistics of SAGE cannot be used. We have used in MetaSAGE the lower bounds as 1 and the upper bounds as 1000.

3.6   Tree reductions

After computing the flow on each edge in the overlap graph, MetaSAGE will perform *in-tree* and *out-tree* reductions. In the overlap graph, a node that has only one outgoing edge and more than one incoming edges is called an *in-node*, and this structure in the graph is called *in-tree*; *out-node* and *out-tree* are defined similarly. In Figure 3.12, on the left is an in-tree. The flow on its outgoing edge is 2 while the two incoming edges have flow 1. This in-tree can be simplified into the tree on the right of the graph.

Besides in-tree and out-tree reduction, MetaSAGE also removes loops in the graph.
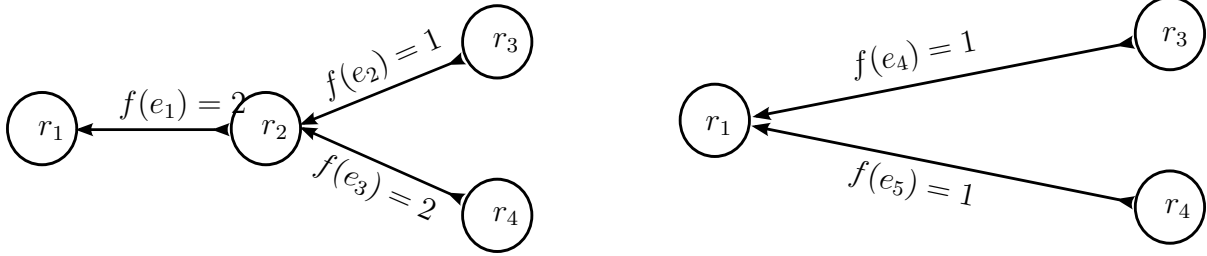
Figure 3.12: In-tree simplification.

In Figure 3.13, on the left an edge having the same end points is called a loop in the overlap graph. If there is only one valid path travel through these edges, MetaSAGE replaces these edges with an new edge, as shown in the right figure in Figure 3.13. The algorithm for in-tree/out-tree reduction and loop reduction [12] is shown in Algorithm 6.
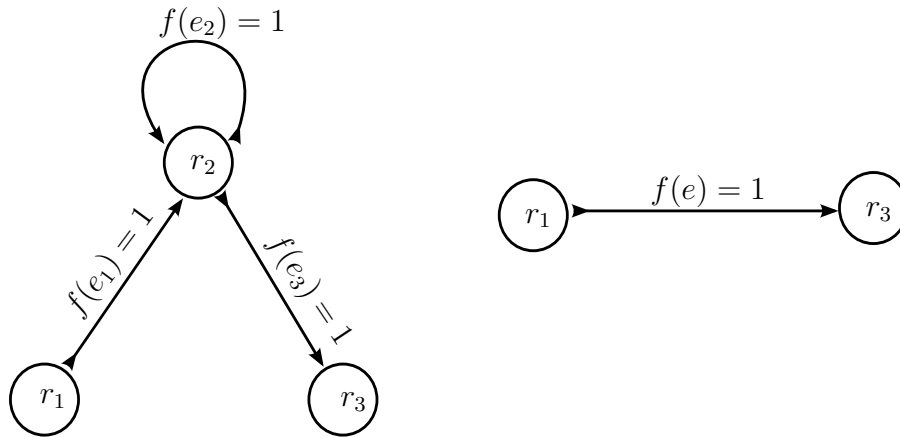
Figure 3.13: Loop reduction.

3.7   Mate-pair information

After tree reduction, the overlap graph will be less complex. However there might be many "ambiguous" nodes which could be merged in more than one way. In Figure 3.15 node r_2 is an ambiguous node because there are two ways to merge the edges adjacent to r_2. The edge $e_1 = (r_1, r_3)$ can be merged with any of the two outgoing edges $e_3 = (r_3, r_4)$ and $e_4 = (r_3, r_5)$ and similar situation happens for all the edges. Using

---

Algorithm 6: In-tree, out-tree and loop reduction

    input : Overlap graph $G = (V, E)$

    output: Tree after reduction

1 **for** *each node $u \in V$* **do**

2 $inDegree \leftarrow$ number of incoming edges incident on u $outDegree \leftarrow$ number of outgoing edges incident on u **if** *$inDegree = 1$ and $outDegree > 1$* **then**

3   **for** *each out-edge $e_1 = (u, u_{out})$* **do**

4   merge edges $e = (u_{in}, u)$ and e_1

5   **end**

6   **end**

7   **else if** *$inDegree > 1$ and $outDegree = 1$* **then**

8   **for** *each in-edge $e_2 = (u, u_{in})$* **do**

9   merge edges $e = (u, u_{out})$ and e_2

10   **end**

11   **end**

12 **end**

13 **for** *each loop (u, u) in G* **do**

14   **if** *u has only two edges (x, u) and (u, y) incident on it* **then**

15   **if** *there is only one possible path $p = x, u, u, y$ according to the flow* **then**

16   Remove p from G Add (x, y) to G

17   **end**

18   **end**

19 **end**

20 **return** G

---

mate-pair information is a good way to solve this problem.

As mentioned in Section 2.3.2, a mate pair refers to two reads coming from the same subsequence of the reference genome. One of them is the suffix and the other is the prefix. In sequences generated by NGS, mate-pairs are stored together in the output file. MetaSAGE uses this mate-pair information to resolve ambiguous nodes in the overlap graph. Suppose the reads are generated from a reference genome shown in Figure 3.14. The reads on edge e_1 and on edge e_3 are mate-pairs, while, the reads on edge e_2 and edge e_4 are also mate-pairs. With this information we can merge e_1 with e_4 and e_2 with e_3, then remove the ambiguous node r_2 from the overlap graph.
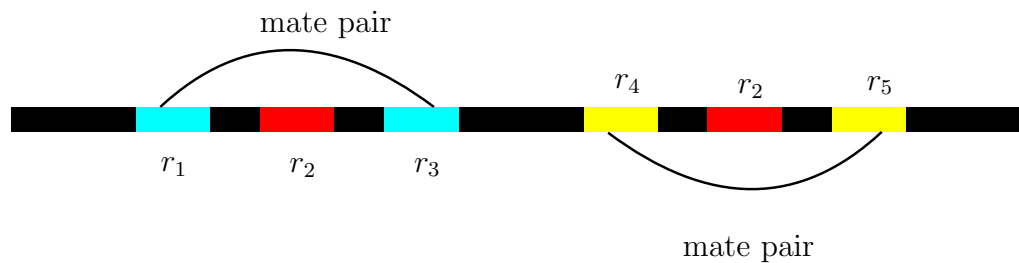
Figure 3.14: Ambiguous node.
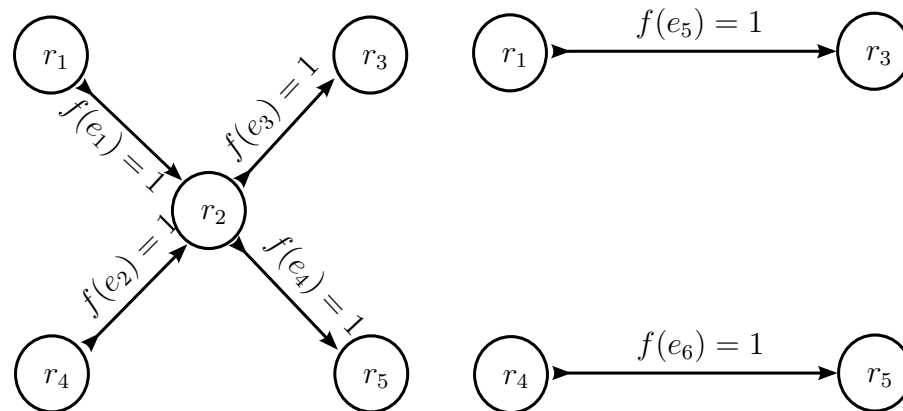
Figure 3.15: Ambiguous node.

3.8   The algorithm

The overall algorithm of MetaSAGE is shown in Algorithm 7. (The FASTA file format is described in the next chapter.) We have described in detail all new steps of MetaSAGE and included the most important steps of SAGE that were carried over into MetaSAGE such that the structure of MetaSAGE becomes clear; however, we refer the reader to [12] and [14] for the complete description of the steps of MetaSAGE that were inherited from SAGE.

Algorithm 7: MetaSAGE

---

    input : Read set $R = \{r_1, r_2, \ldots, r_n\}$, minimum overlap $minOverlap$

    output: Set $C = c_1, c_2, \ldots, c_k$ of contigs

1 **call** $RACER(R)$

2 build overlap graph

3 **repeat**

4   composite graph

5   remove bubbles and dead-ends

6 **until** *no edge is removed from* G;

7 split chimeric node

8 estimate genome size L

9 compute minimum cost flow on each edge

10 repeat

11   do tree reduction

12   remove loops

13 **until** *no edge is removed from* G;

14 merge contigs using pair-end information

15 extract contigs in to C

16 write C into FASTA file

Chapter 4

Experiments

In this chapter a comparison is made between MetaSAGE and three other metagenomic assembly programs, IDBA-UD [29], MetaVelvet [26] and Omega [11]. In our tests, Genovo [16] failed to give a result within a tolerable time on our datasets, so we do not discuss it here. IDBA-UD is an upgraded version of meta-IDBA [28] released in 2013. MetaVelvet is software based on Velvet [40], released in 2012. They both use the de Bruijn graph. Omega is a newly released program. It is also based on the overlap graph approach of SAGE, similar to MetaSAGE. In this comparison, four datasets were generated using sequencing simulation software MetaSim[32]. The results obtained by the competing programs were compared using several criteria. In this chapter, the datasets used in the experiments and the evaluation criteria are introduced, and the performance of the four metagenome assemblers is compared.

4.1   Datasets

As mentioned before, metagenomic data was sequenced directly from a sample of living community, usually bacteria, obtained from the environment. Because there is no reference of those genome sequences, we cannot make a full evaluation on those datasets. In order to make a careful evaluation of our program, artificial datasets were used in this thesis. Those artificial datasets were generated by MetaSim[32]. MetaSim is a genome sequence simulator released in 2008. It can generate sequences based on different param-

eters and error models. In this thesis, the read length was set at 100 bp, and the error model was set as the error model of Illumina [8]. The average and standard deviation of insert size were set at 500 bp and 50 bp.

To test the performances on various taxonomic levels of diversity, the test datasets were set in four different taxonomic levels of diversity, that is, "order-level", "family-level", "genus-level" and "species-level". In general, on lower taxonomic level, the genomes are more similar to each other. So it is more difficult to do assembly on a lower taxonomic level. In each level, 20 species genomes were selected randomly in different coverage.

We followed the procedure indicated in the MetaVelvet paper [26]. Here are the steps of our procedure:

1. Download the simulator MetaSim [32].

2. Download the database of bacteria from the NCBI[1] and imported it into MetaSim.

3. Set the profiles in MetaSim, indicating which genome sequence I would use and the coverage ratio between each reference sequence.

4. Set the read length, total number of reads, set the error module as Illunima; the error module is set by a profile in MetaSim.

5. Generate datasets by the MetaSim.

The details of the datasets are given in Table 4.1 and Table 4.2.

The input file format for a genome assembler is usually FASTQ. FASTQ format is a text-based format for storing both a biological sequence (usually nucleotide sequence) and its corresponding quality scores. Both the sequence letter and quality score are encoded with a single ASCII character for brevity. An example of FASTQ file is shown in Figure 4.1 (from: http://drive5.com/usearch/manual/fastq_files.html).

The output is FASTA, which is very similar to FASTQ except that there are no quality scores (there are only two lines instead of four for each sequence) and the "@" at the beginning of the description is replaced by ">".
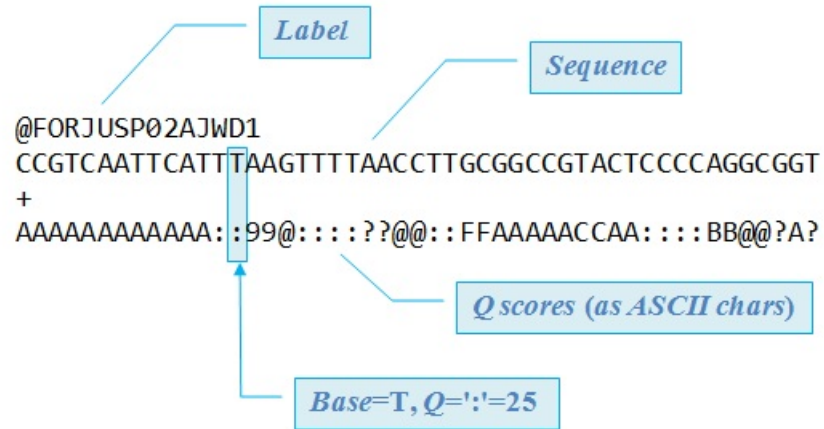
---

[1]ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/all.fna.tar.gz

Figure 4.1: The FASTQ file.

Order-level dataset

Domain	Phylum	Class	Order	Family	Genus	Species	Strain	Length	coverage
Bacteria	Proteobacteria	Alphaproteobacteria	Caulobacterales	Caulobacteraceae	Caulobacter	crescentus	NA1000	4,042,929	89.22
Bacteria	Proteobacteria	Alphaproteobacteria	Parvularculales	Parvularculaceae	Parvularcula	bermudensis	HTCC2503	2,902,643	59.50
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Rhizobiaceae	Rhizobium	etli	CFN 42	4,381,608	29.74
Bacteria	Proteobacteria	Alphaproteobacteria	Rhodobacterales	Rhodobacteraceae	Dinoroseobacter	shibae	DFL 12	3,789,584	227.99
Bacteria	Proteobacteria	Alphaproteobacteria	Rhodospirillales	Rhodospirillaceae	Azospirillum	sp.	B510	3,311,395	49.51
Bacteria	Proteobacteria	Alphaproteobacteria	Sphingomonadales	Erythrobacteraceae	Erythrobacter	litoralis	HTCC2594	3,052,398	59.39
Bacteria	Proteobacteria	Gammaproteobacteria	Acidithiobacillales	Acidithiobacillaceae	Acidithiobacillus	ferrooxidans	ATCC 23270	2,982,397	356.54
Bacteria	Proteobacteria	Gammaproteobacteria	Aeromonadales	Aeromonadaceae	Aeromonas	hydrophila	hydrophila ATCC 7966	4,744,448	29.70
Bacteria	Proteobacteria	Gammaproteobacteria	Alteromonadales	Alteromonadaceae	Alteromonas	macleodii	Deep ecotype	4,480,937	59.47
Bacteria	Proteobacteria	Gammaproteobacteria	Cardiobacteriales	Cardiobacteriaceae	Dichelobacter	nodosus	VCS1703A	1,389,350	44.50
Bacteria	Proteobacteria	Gammaproteobacteria	Chromatiales	Ectothiorhodospiraceae	Alkalilimnicola	ehrlichei	MLHE-1	3,275,944	29.71
Bacteria	Proteobacteria	Gammaproteobacteria	Enterobacteriales	Enterobacteriaceae	Escherichia	coli	K-12 substr. MG1655	4,641,652	143.67
Bacteria	Proteobacteria	Gammaproteobacteria	Legionellales	Legionellaceae	Legionella	pneumophila	str. Paris	3,503,610	29.68
Bacteria	Proteobacteria	Gammaproteobacteria	Methylococcales	Methylococcaceae	Methylococcus	capsulatus	str. Bath	3,304,561	133.63
Bacteria	Proteobacteria	Gammaproteobacteria	Oceanospirillales	Halomonadaceae	Chromohalobacter.	salexigens	DSM 3043	3,696,649	29.67
Bacteria	Proteobacteria	Gammaproteobacteria	Pasteurellales	Pasteurellaceae	Pasteurella.	multocida	multocida str. Pm70	2,257,487	99.20
Bacteria	Proteobacteria	Gammaproteobacteria	Pseudomonadales	Pseudomonadaceae	Pseudomonas	putida	W619	5,774,330	89.12
Bacteria	Proteobacteria	Gammaproteobacteria	Thiotrichales	Piscirickettsiaceae	Thiomicrospira	crunogena	XCL-2	2,427,734	99.03
Bacteria	Proteobacteria	Gammaproteobacteria	Vibrionales	Vibrionaceae	Vibrio	cholerae	O1 biovar eltor str. N16961	2,961,149	173.23
Bacteria	Proteobacteria	Gammaproteobacteria	Xanthomonadales	Xanthomonadaceae	Xanthomonas	campestris	pv. campestris	5,079,002	272.32

Family-level dataset

Domain	Phylum	Class	Order	Family	Genus	Species	Strain	Length	coverage
Bacteria	Firmicutes	Bacilli	Bacillales	Alicyclobacillaceae	Alicyclobacillus	acidocaldarius	acidocaldarius DSM 446	3,018,755	34.94
Bacteria	Firmicutes	Bacilli	Bacillales	Bacillaceae	Bacillus	subtilis	subsp. subtilis str. 168	4,215,606	59.84
Bacteria	Firmicutes	Bacilli	Bacillales	Listeriaceae	Listeria	monocytogenes	str. 4b F2365	2,905,187	39.84
Bacteria	Firmicutes	Bacilli	Bacillales	Paenibacillaceae	Brevibacillus	brevis	NBRC 100599	6,296,436	54.80
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Bartonellaceae	Bartonella	bacilliformis	KC583	1,445,021	124.60
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Bejerinckiaceae	Methylocella	silvestris	BL2	4,305,430	29.87
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Bradyrhizobiaceae	Nitrobacter	hamburgensis	X14	4,406,967	29.92
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Brucellaceae	Brucella	suis	1330	2,107,794	29.83
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Hyphomicrobiaceae	Hyphomicrobium	denitrificans	ATCC 51888	3,638,969	29.92
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Methylobacteriaceae	Methylobacterium	chloromethanicum	CM4	5,777,908	483.58
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Phyllobacteriaceae	Mesorhizobium	loti	MAFF303099	7,036,071	149.52
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Rhizobiaceae	Rhizobium	etli	CFN 42	4,381,608	30.56
Bacteria	Proteobacteria	Alphaproteobacteria	Rhizobiales	Xanthobacteraceae	Azorhizobium	caulinodans	ORS 571	5,369,772	99.80
Bacteria	Proteobacteria	Gammaproteobacteria	Alteromonadales	Alteromonadaceae	Alteromonas	macleodii	Deep ecotype	4,480,937	29.27
Bacteria	Proteobacteria	Gammaproteobacteria	Alteromonadales	Colwelliaceae	Colwellia	psychrerythraea	34H	5,373,180	413.70
Bacteria	Proteobacteria	Gammaproteobacteria	Alteromonadales	Ferrimonadaceae	Ferrimonas	balearica	DSM 9799	4,279,159	119.75
Bacteria	Proteobacteria	Gammaproteobacteria	Alteromonadales	Idiomarinaceae	Idiomarina	loihiensis	L2TR	2,839,318	59.80
Bacteria	Proteobacteria	Gammaproteobacteria	Alteromonadales	Pseudoalteromonadaceae	Pseudoalteromonas	haloplanktis	TAC125	3,214,944	39.86
Bacteria	Proteobacteria	Gammaproteobacteria	Alteromonadales	Psychromonadaceae	Psychromonas	ingrahamii	37	4,559,598	140.61
Bacteria	Proteobacteria	Gammaproteobacteria	Alteromonadales	Shewanellaceae	Shewanella	oneidensis	MR-1	4,969,811	49.84

Table 4.1: Order-level and Family-level datasets

				Genus-level dataset					
Domain	Phylum	Class	Order	Family	Genus	Species	Strain	Length	coverage
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Candidatus**	*Moranella*	*endobia PCIT*	*538,294*	*256.39*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Citrobacter**	*koseri*	*ATCC BAA-895*	*4,720,462*	*31.32*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Cronobacter**	*sakazakii*	*ATCC BAA-894*	*4,368,373*	*31.41*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Dickeya**	*dadantii*	*Ech703*	*4,679,450*	*104.53*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Edwardsiella**	*ictaluri*	*93-146*	*3,812,301*	*83.66*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Enterobacter**	*asburiae*	*LF7a*	*4,812,833*	*78.45*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Erwinia**	*billingiae*	*Eb661*	*5,100,167*	*31.31*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Escherichia**	*coli*	*str. K-12 substr. MG1655*	*4,641,652*	*83.63*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Klebsiella**	*variicola*	*At-22*	*5,458,505*	*31.39*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Pantoea**	*vagans*	*C9-1*	*4,024,986*	*250.92*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Pectobacterium**	*atrosepticum*	*SCRI1043*	*5,064,019*	*31.40*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Photorhabdus**	*asymbiotica*	*subsp. asymbiotica ATCC 43949*	*5,064,808*	*67.97*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Proteus**	*mirabilis*	*HI4320*	*4,063,606*	*88.90*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Rahnella**	*sp.*	*Y9602*	*4,864,217*	*62.70*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Salmonella**	*bongori*	*NCTC 12419*	*4,460,105*	*31.34*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Serratia**	*proteamaculans*	*568*	*5,448,853*	*104.53*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Shigella**	*exneri*	*5 str. 8401*	*4,574,284*	*83.63*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Sodalis**	*glossinidius*	*morsitans*	*4,171,146*	*99.32*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Xenorhabdus**	*bovienii*	*SS-2004*	*4,225,498*	*177.62*
Bacteria	*Proteobacteria*	*Gammaproteobacteria*	*Enterobacteriales*	*Enterobacteriaceae*	**Yersinia**	*pestis*	*Angola*	*4,504,254*	*407.58*

				Species-level dataset					
Domain	Phylum	Class	Order	Family	Genus	**Species**	Strain	Length	coverage
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**amyloliquefaciens**	*DSM 7*	*3,980,199*	*91.14*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**anthracis**	*Ames*	*5,227,293*	*30.36*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**atrophaeus**	*1942*	*4,168,266*	*273.41*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**cellulosilyticus**	*DSM 2522*	*4,681,672*	*30.40*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**cereus**	*Q1*	*5,214,195*	*101.20*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**clausii**	*KSM K16*	*4,303,871*	*116.43*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**coagulans**	*36D1*	*3,552,226*	*217.61*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**cytotoxicus**	*NVH 391 98*	*4,087,024*	*101.12*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**halodurans**	*C 125*	*4,202,352*	*50.63*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Geobacillus*	**kaustophilus**	*HTA426*	*3,544,776*	*30.35*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**licheniformis**	*ATCC 14580*	*4,222,597*	*30.33*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**megaterium**	*DSM319*	*5,097,447*	*126.58*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**pseudormus**	*OF4*	*3,858,997*	*30.28*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**pumilus**	*SAFR 32*	*3,704,465*	*308.56*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**selenitireducens**	*MLS10*	*3,592,487*	*162.08*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**subtilis**	*subsp. subtilis str. 168*	*4,215,606*	*30.39*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Geobacillus*	**thermoglucosidasius**	*C56 YS93*	*3,893,306*	*45.60*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**thuringiensis**	*Al Hakan*	*5,257,091*	*242.98*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**tusciae**	*DSM 2912*	*3,384,766*	*30.38*
Bacteria	*Firmicutes*	*Bacilli*	*Bacillales*	*Bacillaceae*	*Bacillus*	**weihenstephanensis**	*KBAB4*	*5,262,775*	*30.38*

Table 4.2: Genus-level and Species-level datasets

4.2   Environment

In our test, all programs were running on the SHARCNET system. It is a distributed computing system. The nodes we used is working on Linux, containing 12 cores, 256GB RAM.

4.3   Evaluation

All genome assemblers output a collection of contigs or scaffolds which are assembled from the input reads. The goal of genome assembly is to generate long and correct DNA sequences. So the length of the contigs or scaffolds obtained from the assembler is an important criterion to judge their performance. Besides the length of contigs, another important criterion is the correctness of the output. In this thesis, we evaluate assemblers both by the length of contigs or scaffolds they generate and the correctness of their assembly. The evaluation in this thesis was executed by a software named QUAST (Quality Assessment Tool for Genome Assemblies) [10] which is the current state-of-the-art in genome assembly evaluation.

We introduce below some useful notions for evaluating an assembler.

Indels and Mismatches

An indel is either an insertion or a deletion. It refers to a character that appears in the assembled sequence while is does not appear in the reference genome, or a character in the reference genome that does not appear shows in the assembled sequence. Mismatch is a character in an assembled sequence that does not match the reference genome sequence. Figure 4.2 shows an example of indels. Sequence S_1 is the reference genome sequence and S_2 is the assembled sequence. The indels are in red and mismatches in blue.

$$S_1: \quad \text{AGCTA–GCATTTACGATAGCCGATAGCTAAATTAC}$$
$$\quad\quad |\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|\,|$$
$$S_2: \quad \text{AGCTAAGCATGTA–GATAGCCGATCGCTAAATTAC}$$

Figure 4.2: Indels and mismatches

N50

N50 is an important criterion for evaluating the length of the assembled result. It provides an overview of the length of contigs produced by an assembler. Assuming that an assembler produces a set $C = c_1, c_2, \ldots, c_k$ of contigs, where the length of the contig c_i is l_i. The N50 of the set C is defined as

$$N50(C) = \max\{\, l \mid \sum_{l_i \geq l} l_i \geq \frac{1}{2} \sum_{l_i > 0} l_i \,\}.$$

In other words, N50 is defined as the maximum length l such that the collection of contigs of length at least l includes at lest half of the total length of all contigs. N75 is defined similarly.

NG50

NG50 is defined as $\max\{l \mid \sum_{l_i \geq l} l_i \geq \frac{1}{2}L\}$, where L is the length of the reference genome. Compared to N50, NG50 only considers the collection of contigs that cover at least half the reference genome.

Misassemblies

According to the description in QUAST, missassemblies are the contigs that satisfy one of the following criteria:

- the left flanking sequence aligns over 1kbp away from the right flanking sequence on the reference;

- flanking sequences overlap on more than 1kpb;

- flanking sequences align to different strands or different chromosomes.

So the number of misassemblies indicates how many times the assembler merges contigs that are not close to being adjacent.

NGA50

NA50, NGA50 ("A" stands for "aligned") are similar to the corresponding metrics without "A", but in this case aligned blocks instead of contigs are considered. In other words, QUAST will firstly break the contigs into shorter contigs at those positions where they make misassemblies and then compute the N50 and NG50 with those broken shorter contigs. Figure 4.3 indicates the difference between N50 and NGA50.
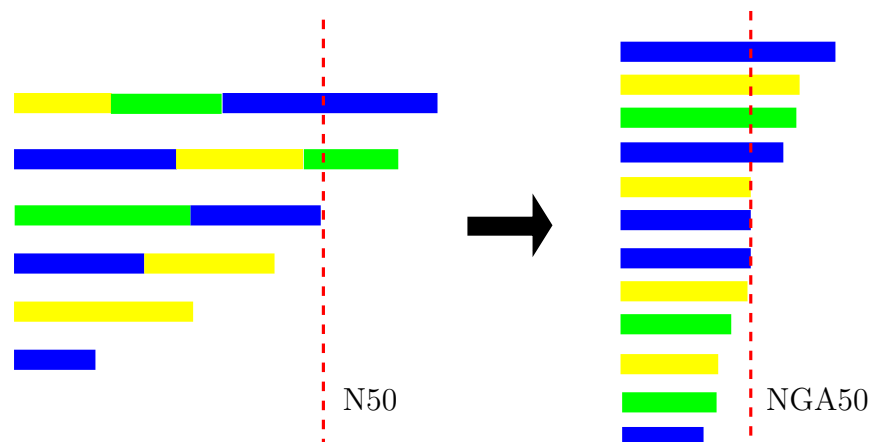
Figure 4.3: The difference between N50 and NGA50. On the left, colour change indicates a misassembly.

4.4 Results and analysis

In the tests, most parameters of comparing software are set as default, except for k-mer length in MetaVelvet and minimum overlap length in Omega and MetaSAGE. We tried several different values on those adjustable parameters, choosing the one which got highest NGA50. Table 4.3 shows the comparison of NGA50, the largest alignment, the number of misassemblies, and the number of mismatches and indels between those four programs, since these four parameters are usually considered the most important ones. The complete list of results can be found in Tables 4.4 - 4.7. Generally, on a lower taxonomic level, genome sequences are more similar with each other and it is harder for an assembler to tell apart whether reads are coming from the same genome sequence.

So, all software produce smaller values NGA50 in lower taxonomic levels.

In the comparison between software, we can see that IDBA-UD and MetaSAGE both have an obvious advantage in terms of both NGA50 and largest alignment over the other two programs. MetaVelvet has the lowest NGA50, up to six times lower than that of MetaSAGE.

As far as misassemblies are concerned, IDBA-UD and MetaVelvet produce the fewest and Omega the most by very far. MetaSAGE is in between, and its performance in this respect needs to be improved. On the other hand, MetaSAGE and IDBA-UD have the lowest number of mismatches and indels with MetaVelvet coming in the third place and Omega a distant last.

	IDBA-UD	MetaVelvet	Omega	MetaSAGE
Order Level				
NGA50	**175,984**	54,887	54,559	175,460
Largest alignment	948,840	791,046	649,618	**977,953**
Number of misassemblies	21	**20**	483	87
Mismatches & indels per 100 kbp	**3**	18	104	**3**
Family Level				
NGA50	**212,106**	36,949	53,219	180,872
Largest alignment	**1,435,467**	1,036,840	752,788	1,177,673
Number of misassemblies	**21**	22	672	68
Mismatches & indels per 100 kbp	**3**	32	97	**3**
Genus Level				
NGA50	94,287	17,613	26,246	**104,837**
Largest alignment	1,100,736	544,812	479,171	**1,326,766**
Number of misassemblies	**104**	146	1,864	282
Mismatches & indels per 100 kbp	20	51	163	**8**
Species Level				
NGA50	88,957	16,319	20,161	**94,395**
Largest alignment	1,101,923	373,412	619,278	**1,203,389**
Number of misassemblies	**98**	104	3,426	803
Mismatches & indels per 100 kbp	**21**	48	183	**21**

Table 4.3: Comparison of the four metagenome assemblers; best results in bold.

Table 4.4: Order level comparison.

Parameter	IDBA-UD	MetaVelvet	Omega	MetaSAGE
# contigs (>= 0 bp)	782	894	2,197	2,567
# contigs (>= 1000 bp)	694	593	1,941	1,494
Total length (>= 0 bp)	71,138,737	71,565,561	70,897,828	73,697,930
Total length (>= 1000 bp)	71,074,201	71,361,886	70,716,725	72,961,141
# contigs	782	894	2,197	2,567
Largest contig	2,076,502	1,525,931	1,285,269	1,366,496
Total length	71,138,737	71,565,561	70,897,828	73,697,930
Reference length	71,999,807	71,999,807	71,999,807	71,999,807
GC (%)	58	58	58	58
Reference GC (%)	58	58	58	58
N50	265,476	303,142	160,126	188,294
NG50	251,222	293,013	154,413	190,827
N75	125,344	144,808	41,706	61,599
NG75	121,771	143,896	39,408	68,255
L50	70	65	92	88
LG50	72	66	95	83
L75	170	152	330	262
LG75	175	154	351	243
# misassemblies	21	87	20	483
# misassembled contigs	16	66	14	239
Misassembled contigs length	3,484,075	11,333,649	1,990,216	26,780,944
# local misassemblies	203	577	418	6,754
# unaligned contigs	3 + 2 part	4 + 23 part	1 + 23 part	58 + 94 part
Unaligned length	2,710	16,399	17,913	85,239
Genome fraction (%)	99	99	98	99
Duplication ratio	1	1	1	1
# N's per 100 kbp	17	126	127	11
# mismatches per 100 kbp	2	2	16	102
# indels per 100 kbp	1	1	2	2
Largest alignment	948,840	977,953	791,046	649,618
NA50	181,638	176,960	87,551	51,867
NGA50	179,939	175,460	86,205	54,559
NA75	94,797	94,717	34,684	13,339
NGA75	92,050	92,578	32,369	14,543
LA50	105	113	186	288
LGA50	107	114	192	272
LA75	238	251	518	1,052
LGA75	245	255	542	961

Table 4.5: Family level comparison.

Parameter	IDBA-UD	MetaVelvet	Omega	MetaSAGE
# contigs (>= 0 bp)	1,008	4,524	3,858	1,009
# contigs (>= 1000 bp)	885	3,880	2,373	714
Total length (>= 0 bp)	83,401,681	82,787,516	87,248,861	83,955,104
Total length (>= 1000 bp)	83,312,562	82,330,312	86,220,769	83,760,804
# contigs	1,008	4,524	3,858	1,009
Largest contig	1,435,467	1,205,272	2,008,080	1,752,732
Total length	83,401,681	82,787,516	87,248,861	83,955,104
Reference length	84,622,471	84,622,471	84,622,471	84,622,471
GC (%)	53	53	54	53
Reference GC (%)	53	53	53	53
N50	294,468	81,152	176,064	330,866
NG50	283,448	76,026	189,475	324,189
N75	132,286	20,348	55,721	155,698
NG75	128,250	18,701	64,971	152,468
L50	77	188	110	68
LG50	79	199	102	69
L75	182	742	324	162
LG75	189	812	291	166
# misassemblies	21	22	672	68
# misassembled contigs	21	15	329	54
Misassembled contigs length	4,666,233	2,943,935	38,948,050	14,342,351
# local misassemblies	266	506	9,539	954
# unaligned contigs	4 + 3 part	0 + 19 part	88 + 154 part	6 + 32 part
Unaligned length	4,875	9,379	152,702	26,536
Genome fraction (%)	99	98	99	99
Duplication ratio	1	1	1	1
# N's per 100 kbp	23	134	14	163
# mismatches per 100 kbp	2	30	95	1
# indels per 100 kbp	1	2	2	2
Largest alignment	1,435,467	1,036,840	752,788	1,177,673
NA50	218,651	53,666	49,302	183,011
NGA50	215,196	51,573	53,219	180,872
NA75	97,409	18,472	12,482	84,319
NGA75	94,917	17,305	14,483	83,384
LA50	101	339	329	121
LGA50	104	356	303	123
LA75	245	1,019	1,252	287
LGA75	255	1,096	1,106	293

Table 4.6: Genus level comparison.

Parameter	IDBA-UD	MetaVelvet	Omega	MetaSAGE
# contigs (>= 0 bp)	4,086	7,393	6,403	10,304
# contigs (>= 1000 bp)	2,796	4,512	4,784	3,677
Total length (>= 0 bp)	83,163,523	81,883,399	89,312,843	88,119,228
Total length (>= 1000 bp)	82,242,237	79,876,033	88,146,342	83,742,000
# contigs	4,086	7,393	6,403	10,304
Largest contig	1,566,923	1,685,916	1,897,906	1,583,954
Total length	83,163,523	81,883,399	89,312,843	88,119,228
Reference length	88,597,813	88,597,813	88,597,813	88,597,813
GC (%)	52	52	52	52
Reference GC (%)	52	52	52	52
N50	167,968	119,734	95,297	200,609
NG50	151,790	99,353	97,599	200,590
N75	57,396	26,071	26,348	51,238
NG75	42,780	14,599	27,305	49,997
L50	115	137	212	107
LG50	132	168	208	108
L75	328	502	645	324
LG75	411	764	625	331
# misassemblies	104	146	1,864	282
# misassembled contigs	94	79	852	210
Misassembled contigs length	4,040,598	14,371,190	52,733,484	11,632,453
# local misassemblies	786	2,244	9,330	6,536
# unaligned contigs	82 + 50 part	11 + 140 part	39 + 175 part	377 + 348 part
Unaligned length	136,414	76,966	172,777	579,741
Genome fraction (%)	94	92	95	95
Duplication ratio	1	1	1	1
# N's per 100 kbp	107	650	14	1,345
# mismatches per 100 kbp	14	45	159	5
# indels per 100 kbp	6	6	4	3
Largest alignment	1,100,736	544,812	479,171	1,326,766
NA50	111,420	39,975	26,001	105,277
NGA50	100,133	34,481	26,246	104,837
NA75	45,427	14,144	8,064	35,790
NGA75	34,573	9,646	8,365	35,054
LA50	176	493	785	204
LGA50	202	584	771	206
LA75	471	1,363	2,323	561
LGA75	575	1,794	2,258	571

Table 4.7: Species level comparison.

Parameter	IDBA-UD	MetaVelvet	Omega	MetaSAGE
# contigs (>= 0 bp)	5,565	7,978	5,948	10,157
# contigs (>= 1000 bp)	4,074	5,072	4,214	4,234
Total length (>= 0 bp)	78,450,823	70,339,748	86,175,820	82,276,972
Total length (>= 1000 bp)	77,358,497	68,352,669	84,956,564	78,329,313
# contigs	5,565	7,978	5,948	10,157
Largest contig	2,107,658	707,687	916,330	3,761,321
Total length	78,450,823	70,339,748	86,175,820	82,276,972
Reference length	85,451,411	85,451,411	85,451,411	85,451,411
GC (%)	42	43	42	42
Reference GC (%)	42	42	42	42
N50	146,659	47,820	71,384	147,954
NG50	124,643	28,532	71,759	138,933
N75	40,928	13,462	20,787	33,990
NG75	18,590	3,251	21,328	23,833
L50	125	330	245	110
LG50	151	538	240	121
L75	387	1,057	829	397
LG75	587	2,746	803	480
# misassemblies	98	104	3,426	803
# misassembled contigs	86	62	1,599	572
Misassembled contigs length	2,511,518	3,972,632	40,748,439	11,192,816
# local misassemblies	1,014	1,469	13,862	7,543
# unaligned contigs	72 + 71 part	15 + 24 part	69 + 214 part	212 + 527 part
Unaligned length	146,779	33,571	205,425	607,138
Genome fraction (%)	91	82	94	92
Duplication ratio	1	1	1	1
# N's per 100 kbp	144	106	24	1,739
# mismatches per 100 kbp	12	43	177	17
# indels per 100 kbp	9	5	6	4
Largest alignment	1,101,923	373,412	619,278	1,203,389
NA50	112,491	33,515	19,833	105,924
NGA50	88,997	21,324	20,161	94,395
NA75	29,163	10,310	5,750	23,659
NGA75	15,072	2,746	5,929	17,777
LA50	170	470	825	165
LGA50	205	750	807	181
LA75	514	1,422	2,941	576
LGA75	763	3,485	2,848	691

4.5   Time and Memory

The comparison with respect to time and memory consumption is presented in Table 4.8. IDBA-UD and MetaSAGE require the lowest amount of memory, significantly lower than Omega and MetaVelvet. MetaSAGE is the fastest, followed by the MetaVelvet, than IDBA-UD, and very far from the top three, Omega. For the species level test, Omega took 10 days to complete, as opposed to the two hours of MetaSAGE.

	IDBA-UD	MetaVelvet	Omega	MetaSAGE
Order Level				
Running time (h)	18.7	5.7	59.7	**1.6**
Running space (GB)	40.4	108.2	117.5	**35**
Family Level				
Running time (h)	18.7	6.2	99.8	**2.3**
Running space (GB)	51.6	133.8	165.0	**49.7**
Genus Level				
Running time (h)	9.9	6.3	96.5	**2.2**
Running space (GB)	**21.7**	119.3	132.1	39.0
Species Level				
Running time (h)	8.8	5.5	235.2	**2.0**
Running space (GB)	43.4	121.7	134.9	**40.4**

Table 4.8: Running time and space; best results in bold.

Chapter 5

Conclusions

In this thesis, a new metagenome assembler, MetaSAGE, was introduced. MetaSAGE is based on SAGE [14] and preserves its main structure. It uses the hash table to build the overlap graph from a set of reads, splits chimeric nodes according to coverage difference, estimates copy count of edges in the overlap graph by the minimum cost flow theory and builds scaffolds using pair-end information. We generated four realistic data sets in different taxonomic levels for our experiments and compared MetaSAGE against three of the top metagenomic assemblers. We show that MetaSAGE is a competitive metagenome assemblers, its main advantages being:

- Very long aligned contigs: the NGA50 and the largest alignment of MetaSAGE are often the largest compared with top metagenome assemblers.

- Lowest number of mismatches and indels, tied with IDBA-UD.

- The most memory efficient, tied with IDBA-UD and the fastest of the metagenome assemblers tested.

Room for improvement remains in the area of misassemblies. Reducing the number of misassemblies while retaining long aligned contigs will be the main focus of future research.

Bibliography

[1] http://microbe.net/simple-guides/fact-sheet-dna-rna-protein/.

[2] http://www.fragilex.org/fragile-x-associated-disorders/genetics-and-inheritance/fmr1-gene/.

[3] Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[4] Thomas J Bandy, Ashley Brewer, Jonathan R Burns, Gabriella Marth, ThaoNguyen Nguyen, and Eugen Stulz. DNA as supramolecular scaffold for functional molecules: progress in DNA nanotechnology. *Chemical Society Reviews*, 40(1):138–148, 2011.

[5] Jonathan Butler, Iain MacCallum, Michael Kleber, Ilya A Shlyakhter, Matthew K Belmonte, Eric S Lander, Chad Nusbaum, and David B Jaffe. Allpaths: de novo assembly of whole-genome shotgun microreads. *Genome Research*, 18(5):810–820, 2008.

[6] Hamidreza Chitsaz, Joyclyn L Yee-Greenbaum, Glenn Tesler, Mary-Jane Lombardo, Christopher L Dupont, Jonathan H Badger, Mark Novotny, Douglas B Rusch, Louise J Fraser, Niall A Gormley, et al. Efficient de novo assembly of single-cell bacterial genomes from short-read data sets. *Nature Biotechnology*, 29(10):915–921, 2011.

[7] S. Nicklen F. Sanger and A.R. Coulson. A DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467, 1977.

[8] Milan Fedurco, Anthony Romieu, Scott Williams, Isabelle Lawrence, and Gerardo Turcatti. Bta, a novel reagent for DNA attachment on glass and efficient generation of solid-phase amplified DNA colonies. *Nucleic Acids Research*, 34(3):e22–e22, 2006.

[9] Andrew V Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22(1):1–29, 1997.

[10] Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

[11] Bahlul Haider, Tae-Hyuk Ahn, Brian Bushnell, Juanjuan Chai, Alex Copeland, and Chongle Pan. Omega: an overlap-graph de novo assembler for metagenomics. *Bioinformatics*, page btu395, 2014.

[12] Md Bahlul Haider. *A new algorithm for de novo genome assembly*. PhD thesis, The University of Western Ontario, 2012.

[13] Jo Handelsman. Metagenomics: application of genomics to uncultured microorganisms. *Microbiology and Molecular Biology Reviews*, 68(4):669–685, 2004.

[14] Lucian Ilie, Bahlul Haider, Michael Molnar, and Roberto Solis-Oba. SAGE: String-overlap Assembly of GEnomes. *BMC Bioinformatics*, 15(1):302, 2014.

[15] Lucian Ilie and Michael Molnar. RACER: Rapid and accurate correction of errors in reads. *Bioinformatics*, page btt407, 2013.

[16] Jonathan Laserson, Vladimir Jojic, and Daphne Koller. Genovo: de novo assembly for metagenomes. *Journal of Computational Biology*, 18(3):429–443, 2011.

[17] Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, Shengting Li, Gao Shan, Karsten Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265–272, 2010.

[18] Marcel Margulies, Michael Egholm, William E Altman, Said Attiya, Joel S Bader, Lisa A Bemben, Jan Berka, Michael S Braverman, Yi-Ju Chen, Zhoutao Chen, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, 2005.

[19] Paul Medvedev and Michael Brudno. Ab initio whole genome shotgun assembly with mated short reads. In *Research in Computational Molecular Biology*, pages 50–64. Springer, 2008.

[20] Paul Medvedev and Michael Brudno. Maximum likelihood genome assembly. *Journal of Computational Biology*, 16(8):1101–1116, 2009.

[21] Paul Medvedev, Marc Fiume, Misko Dzamba, Tim Smith, and Michael Brudno. Detecting copy number variation with mated short reads. *Genome Research*, 20(11):1613–1622, 2010.

[22] Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. In *Algorithms in Bioinformatics*, pages 289–301. Springer, 2007.

[23] Michael Molnar and Lucian Ilie. Correcting illumina data. *Briefings in Bioinformatics*, page bbu029, 2014.

[24] Eugene W Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl 2):ii79–ii85, 2005.

[25] Niranjan Nagarajan and Mihai Pop. Sequence assembly demystified. *Nature Reviews Genetics*, 14(3):157–167, 2013.

[26] Toshiaki Namiki, Tsuyoshi Hachiya, Hideaki Tanaka, and Yasubumi Sakakibara. Metavelvet: an extension of velvet assembler to de novo metagenome assembly from short sequence reads. *Nucleic Acids Research*, 40(20):e155–e155, 2012.

[27] Yu Peng, Henry CM Leung, Siu-Ming Yiu, and Francis YL Chin. IDBA–a practical iterative de bruijn graph de novo assembler. In *Research in Computational Molecular Biology*, pages 426–440. Springer, 2010.

[28] Yu Peng, Henry CM Leung, Siu-Ming Yiu, and Francis YL Chin. Meta-IDBA: a de novo assembler for metagenomic data. *Bioinformatics*, 27(13):i94–i101, 2011.

[29] Yu Peng, Henry CM Leung, Siu-Ming Yiu, and Francis YL Chin. IDBA-ud: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, 28(11):1420–1428, 2012.

[30] Pavel A Pevzner, Haixu Tang, and Michael S Waterman. An eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[31] Benjamin Raphael, Degui Zhi, Haixu Tang, and Pavel Pevzner. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*, 14(11):2336–2346, 2004.

[32] Daniel C Richter, Felix Ott, Alexander F Auch, Ramona Schmid, and Daniel H Huson. Metasim—a sequencing simulator for genomics and metagenomics. *PloS One*, 3(10):e3373, 2008.

[33] Matthew B Scholz, Chien-Chi Lo, and Patrick SG Chain. Next generation sequencing and bioinformatic bottlenecks: the current state of metagenomic data analysis. *Current Opinion in Biotechnology*, 23(1):9–15, 2012.

[34] Jay Shendure and Hanlee Ji. Next-generation DNA sequencing. *Nature Biotechnology*, 26(10):1135–1145, 2008.

[35] Jay Shendure, Gregory J Porreca, Nikos B Reppas, Xiaoxia Lin, John P McCutcheon, Abraham M Rosenbaum, Michael D Wang, Kun Zhang, Robi D Mitra, and George M Church. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science*, 309(5741):1728–1732, 2005.

[36] Jared T Simpson and Richard Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Research*, 22(3):549–556, 2012.

[37] Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM
Jones, and Inanç Birol. Abyss: a parallel assembler for short read sequence data.
Genome Research, 19(6):1117–1123, 2009.

[38] Gerardo Turcatti, Anthony Romieu, Milan Fedurco, and Ana-Paula Tairi. A new
class of cleavable fluorescent nucleotides: synthesis and optimization as reversible
terminators for DNA sequencing by synthesis. *Nucleic Acids Research*, 36(4):e25–
e25, 2008.

[39] J.D. Watson and F.H.C. Crick. Molecular structure of nucleic acids: a structure for
deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.

[40] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read
assembly using de bruijn graphs. *Genome Research*, 18(5):821–829, 2008.

Curriculum Vitae

Name: Wenjing Wan

Post-Secondary University of Western Ontario
Education and London, Ontario, Canada
Degrees: 2013 - present M.Sc. candidate

        Nanjing University
        Nanjing, Jiangsu, China
        2008 - 2012 B.Sci.

Honours and   First Place of Morgan Stanley Global Programming Contest
Awards: 2014

        Third prize of China Undergraduate Mathematical Contest in Modelling
        2010

        Third prize scholarship for academic in Nanjing University
        2009,2010,2011

Related Work Teaching Assistant
Experience: The University of Western Ontario
        2013 - present

        Research Assistant
        The University of Western Ontario
        2013 - present