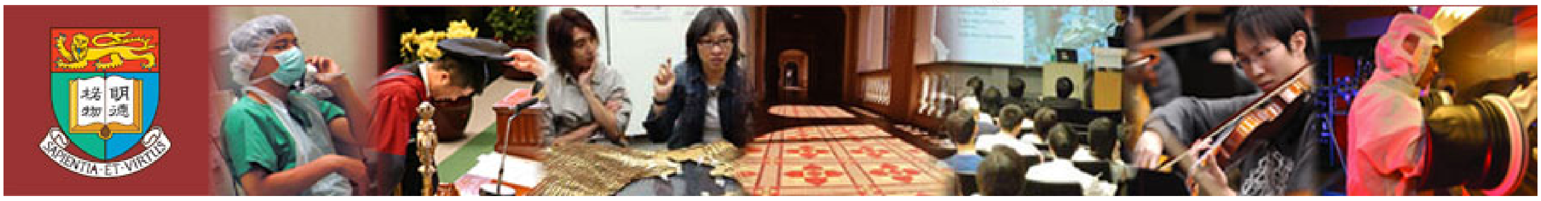

Abstract of thesis entitled

"Large genome *de novo* assembly with bi-directional BWT"

Submitted by

Binghang Liu

for the degree of Master of Philosophy
at The University of Hong Kong
in August, 2015

De novo genome assembly is a fundamental problem in genomics research. When assembling large genomes, time is often a very important concern, and one might have no choice but to use a more efficient assembler like SOAPdenovo2 instead of a high-quality but prohibitively slow assembler (e.g., SPAdes). Yet SOAPdenovo2 has inherent difficulty to utilize the full advantage of longer reads (say, 150bp to 250bp from Illumina HiSeq and MiSeq). Other assemblers that are based on string graphs (e.g., SGA), though less popular and also very slow, are indeed more favorable for longer reads.

In this thesis, I mainly present a new contig assembler called BASE, based on a seed-extension approach. It exploits an efficient indexing of reads to generate adaptive seeds with high probability of unique appearance in the genome and high sequencing quality. Guided by these seeds, BASE constructs extension trees and gradually removes the branches with a method called *reverse validation*, which utilizes information about read coverage and paired-end relationship to obtain consensus sequences of reads sharing the seeds. These consensus sequences are further extended to form high quality contigs.

Benchmark on several bacteria and human datasets demonstrates the performance advantage of BASE in speed and assembly quality when longer reads are used. Our first benchmark was based on two datasets of deeply sequenced bacteria genomes (240X) with read length of 100bp and 250bp. Especially for 250bp reads, BASE performs much better than SOAPdenovo2 and SGA and is similar to SPAdes in performance. Regarding speed, BASE is consistently a few times faster than SPAdes and SGA, but still slower than SOAPdenovo2. We have further compared BASE and SOAPdenovo2 using human genome datasets with read length 100bp, 150bp and 250bp. BASE consistently achieves a higher N50 for all datasets; while the improvement becomes more significant when read length reaches 250bp. SOAPdenovo2 uses relatively more memory when sequencing error is high.

BASE is an efficient assembler for contig construction, with significant improvement in quality for long NGS reads. It could be easily extended to support scaffolding in the near future.

(340 words)

Large genome *de novo* assembly with bi-directional BWT

by

Binghang Liu (劉兵行)
Department of Computer Science
The University of Hong Kong

Supervised by

Prof. Tak-Wah Lam

A thesis submitted in partial fulfillment of the requirements for the degree of *Master of Philosophy in Computer Science* at *The University of Hong Kong*

August, 2015

Declaration

I declare that this thesis represents my own work, except where due acknowledgement is made, and that it has not been previously included in any thesis, dissertation or report submitted to this university or any institution for any diploma, degree or other qualifications.

Signed: ______________________________

Binghang Liu

August, 2015

Acknowledgements

I would like to thank my supervisor Prof. Tak-Wah Lam. Prof. Lam gave me this precious chance to return school after four years work in BGI. He is so nice and patient to allow me to continue my researches even they are not well explained. I feel really sorry to miss the chances to prove myself and let him disappointed on me. He helped me to be self-confident again although the process is great of painful. I appreciate these invaluable supports.

Especially, I thank Ruibang Luo sincerely. He is my direct leader in BGI and in HKU-BGI Bioinformatics Algorithms and Core Technology Research Laboratory. I appreciate his supports in the past years. He is also a good brother for now and for the future.

I am also thankful to Dr. Hing-Fung Ting, Dr. Siu-Ming Yiu for their many useful advices and kindly help.

I thank the members of the HKU-BGI Bioinformatics Algorithms and Core Technology Research Laboratory. Li Dinghua, Liu Chi-Man and Ye Yongtao support a lot directly on my researches. Liu Xuan, my officemate, helped me to get familiar with HKU and to persist in the most difficult time. Wu Haiyang, He Guangzhu, Ou Min, Law-Wai Chun, Wang Heng, Mai Huijun, Fang Ping, I will miss them all and remember the happy days with them.

I thank my leaders in BGI, Yingrui Li, Hancheng Zheng, without their direct support, I certainly have no chance to study in Hong Kong.

Last but not least, my great thanks to my parents, my wife, my son, my two brothers and all my friends who always keep helping out and supporting me.

Remarks. This thesis is a joint work with Tak-Wah Lam, Ruibang Luo, Chi-Man Liu and Dinghua Li, which appeared in International Symposium on Bioinformatics Research and Applications (ISBRA), 2015. This research has also been submitted for a special issue of this conference in *BMC Bioinformatics* upon invitation. Specially, thanks to Dr. Hing-Fung Ting for presenting the paper on behalf of me in the ISBRA2015 conference (as I couldn't attend the conference due to visa issue).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Fast development and application of next generation sequencing (NGS) technologies including Roche 454, Illumina and Solid have generated PB level genomic data within the past 10 years. Hundreds of thousands of species are under sequenced for the first time to assemble their reference sequences. Meanwhile, the assembled sequences are also used to detect variations when given the reference sequences. Genome assembly is considered as a one of the basic computational areas in bioinformatics.

The genome assembly problem is concerned with the reconstruction of the genome sequence (denoted S below) from sequencing data, usually in the form of a large collection of short reads (denoted as R below), which randomly cover the whole genome with sufficient depth. Because of the double-helix structure of DNA, ideally, any R or its reverse complemented sequence should be a substring of S. Due to the limitation of sequencing technologies, R is usually much shorter than S, some reads are found to appear in many different places in S, and they are named as *repetitive* sequences, which cause the basic challenge of assembly. Generally, paired-end sequencing strategy is used to recover much longer repetitive regions, then genome assembly problem is divided into two typical problems, such as *contig* assembly problem, which is to layout reads by overlaps among them to obtain longer continuous sequences, and *scaffolding* problem, which is to layout the contigs with paired-end reads by determining the relative order, orientation and distance among contigs[33, 43]. Here in this thesis, I mainly talk about *contig* assembly problem.

After more than 30 years development, lots of famous contig assemblers are implemented on three main algorithms such as overlap graph/string graph, *de Bruijn* graph and seed-extension based methods. For NGS data, the past few years have witnessed a number of improved *denovo* genome assemblers, providing users choices between speed and accuracy [6]. The more recent NGS technologies have gradually increase the read length beyond 100bp (e.g., 150bp from HiSeq and 250 - 400bp from MiSeq), yet existing efficient assemblers do not have much improvement regarding accuracy, and it is still up to the challenge how to better utilize the advantage of read length to achieve a fast-and-accurate assembler.

In this thesis, I provide a new assembler BASE for longer NGS data. By better utilizing the read length information, BASE can obtain better assemblies than some famous short read assemblers. In the following chapters, I firstly review the development of assembly methods, then show the design of algorithms and its implementation in BASE, and finally I use BASE to assemble deeply sequenced bacteria sequencing datasets and human genome sequencing datasets with different read length, and compare it with some famous NGS assemblers including SOAPdenovo2, SGA and SPAdes.

1.2 Genome assembly methods

The most challenging aspect of genome assembly problem is to obtain the layout or order of *reads* which is mostly consistent with the overlaps among them. The assembled sequences by joining the layout reads, without gaps or unknown base pairs, are named as *contigs*. It has been developed for more than 30 years to solve this problem using different generations of sequencing data.

1.2.1 Overlap-Layout-Consensus (OLC) method

Contig assembly problem became acute on the basis of statistical analysis by Lander Waterman in 1988[24]. In this research, length and amount of assembled contigs could be estimated by sequencing coverage or depth (c), read length (l), and minimum overlap size. Later in 1995, Eugene W. Mayers[41] summarized the 10 years development of overlap-layout-consensus (OLC) strategy to assemble contigs and provided a graph view to layout. This method was further utilized in Celera assembler to assemble *Drosophila* genome in 2000 with Sanger sequencing data[43]. Even in 2010s, genomes, such as tomato[10], which was sequenced by Roche 454 and Sanger, are still assembled with OLC based Newbler and CABOG.

In the OLC strategy, every read is compared in two orientations to obtain overlaps among reads. If the suffix of one read has high similarity with the prefix of another read, we name the high similar regions as overlap, showing the relationship between these two reads. After constructing the overlap graph (read as vertex and overlap as edge), layout is to select the subsets of overlaps, which determine the orders of reads. Finally, multi-alignment of reads is used to form a consensus-measure on each position to generate the consensus character for this position. With the overlap graph, the global relationship between reads could be taken into account, and as mentioned before, this method performs quite well on 500-1000bp for Sanger sequencing and Roche 454 sequencing data. However, it is not so proper for Illumina and Solid sequencing data, which is shorter than even 50bp before 2009. Based on Lander-Waterman model, for shorter reads, higher sequencing depth is needed to support long assemblies. This results in much larger amount of reads to be assembled and brings greater computing challenges to build the overlap graph. Then since 2007, for assembly of short Illumina reads, this strategy has seldom been used.

In 2005, before the burst of NGS assemblers after 2008, Mayers published another research on string graph based assembly[42]. In string graph, reads overlap graph with bi-directional edges are constructed, then transitive edges are removed with a linear time reduction algorithm. After replacing the paths formed by all internal vertices (with single-in and single-out degree) with single composite edges, a read coherent string graph is formed. Then contigs sequences will be obtained after simplification of this graph. In this string graph, however, computational challenge of read alignment was still not efficiently solved, and then it had not been used for NGS reads assembly until 2010. In this year, Simpson published SGA[47], a string graph based assembler for NGS reads, could assemble human genome by constructing string graph directly from the FM-index of reads[48]. After that, some similar assemblers were developed, such as Fermi[28] and Readjoiner[14]. However, although the memory costs are much lower than *de Bruijn* Graph based assemblers, they cost too long time for large genome assembly.

Like OLC based assemblers, a proper minimum overlap size is required in string graph based assemblers to reduce the complexity of graph and to improve the connectivity of graph. Smaller minimum overlap size will increase the probability that the overlap sequence falls within a repetitive region of the genome, thus brings much more branches in the graph and might result in shorter contigs. Meanwhile, according to the Lander-Waterman model[24], larger minimum overlap size leads to a reduction of sufficient support for overlap among reads, thus enhance the demand for higher sequencing depth. Therefore, due to the variation in length of repetitive sequences in genome, it is difficult to find a fixed minimum overlap size that fits each specific case especially when the NGS read is not so long.

1.2.2 *De Bruijn* Graph method

Also in 1995, Idury and Waterman published the idea of sequence graph[18], which is to build a k-tuples' graph and perform Eulerian tours to infer the underlying sequence. Later in late 1990s and early 2000s, the group of Pevzner used *de Bruijn* graph to assemble contigs with sequencing by hybridization (SBH) data and published their assembler Euler[45]. In 2008, Danniel used their graph structure and published Velvet[53], which is generally considered as the first successful *de Bruijn* graph based assembler for NGS reads. In this research, sequencing errors and heterozygsis regions are designed to be recognized with graph structures like *tips* and *bubbles*. After that, SOAPdenovo[36], ALLPATHS-LG[13] and Abyss[49] also use *de Bruijn* graph to assemble large genomes and achieve quite well balance between assembly length, accuracy and computational efficiency.

In these DBG based assemblers, reads are chopped into a sequence of overlapping k-mers with two adjacent k-mers overlapped by k-1 nucleotides. In *de Bruijn* graph, vertexes are k-mers and edges are the transitions among k-mers[33, 53]. Reads are not directly used except helping simplify the DBG graph. The DBG based method works well for assembly of deeply sequenced short reads, especially for avoiding calculating overlaps between reads. But it cannot handle well the repetitive sequences that are longer than k. With various lengths of repetitive sequences in genome, larger k size is promised to assemble longer repetitive regions. Based on Lander-Waterman model, increased k size needs higher sequencing depth to support the continuous coverage of k-mers. Meanwhile, increased k size will worsen the influence of sequencing errors and heterozygous regions, which could increase the memory cost to construct the graph and shorten the assembly length[36]. This problem brings great challenge in practice to find the best proper k size for assembly.

To solve this problem, multiple k-mer strategies like IDBA-UD[44] and SPAdes[4] were developed. In these methods, *de Bruijn* graphs will be rebuilt for many times with k size increasing distinctly. In this way, small k-mers are used to remove most of sequencing errors, treat heterozygous regions and avoid the break of continuous in low coverage regions. Large k-mers are used to solve longer repetitive regions. Yet this requires multiple constructions of DBG and much longer running time, limiting their application for the assembly for relative large genome.

When NGS reads are longer, it is natural to consider using a larger k or using the multiple k-mer strategies. For large genome assembly, SOAPdenovo2 currently only supports maximum 127mer, which is a great waste for 250bp or 300bp reads. For multiple k-mer methods, it's not an easy task to determine the increase of k sizes and the maximum k size. Considering the variation of real sequencing depth and variation of repetitive length for different regions, there is no method to find a ideal k size for each region to balance the requirement from sufficient coverage and repetitive assembly.

1.2.3 Seed-Extension method

Besides these two graph based methods, seed extension based assembly method also has a long history. Amount of assemblers are based on it, such as the first one for assembly of NGS reads, SSAKE[50] *et al.* In SSAKE, reads are kept in hash table with unique sequencing reads as key and frequency as value. A prefix tree is constructed to accelerate the collecting of overlapped reads for a suffix of the extending read. Then in one unassembled read, it finds a suffix, collects reads containing this suffix with prefix tree, obtains the extended sequences, finds a new suffix in the extended sequences and repeats the process until no more extension is possible or conflictions are met, i.e. two or more un-overlapped reads could continue this extension.

This extension method is termed as greedy, because it generally makes decisions only to optimize the local objective function and does not lead to a global optimization[46]. A heuristic algorithm is usually used to determine the appropriate extension, which relays on the overlap with highest quality (longer in length and fewer in mismatches). This might bring false-assembly in some repetitive regions.

Recently, machine-learning methods such as supporting vector machine are used to determine extension[55], which might be a promising direction for further development of this strategy to obtain high quality assembled contigs.

1.2.4 Assembly challenge of repetitive sequences

Treatment of repetitive sequences is a common challenge for any assembly problem. As shown in Figure 1.1, there is a repetitive unit with two copies A and B in genome. After sequencing, we obtain reads r1-7 related with these two copies. Then we discussed the treatment ideas from these three assembly methods.

From this figure, we can summarize the factors that determine the assembly of repetitive sequences:

1) Read length. This is the basic factor determining how long repetitive sequences could be well assembled. Longer reads is better in the view of

repetitive sequences, but read accuracy is also important to choose algorithms and further determines the length of assemble sequences.

2) Minimum overlap size or k-mer size. When minimum overlap size is larger, there will be no confliction after r4. But there will also be only r6 left overlapped with r1, which might result in false assembly. When k size is larger than repetitive sequence, Copy B will be well assembled, but Copy A will be probably lost. Then in practice, many different values of k size or minimum overlap size will be tested to find the optimal one, which can obtain the longest assemblies.

3) Sequencing depth. When sequencing reads cover the genome efficiently enough, there would be kinds of reads covering Copy A like Copy B. Then there might be more reads having longer overlaps with r1, so larger minimum overlap size or k size could be used to solve longer repetitive regions. There might also be reads like r5 connecting a-Copy A-b, so even we use small k size, we can also solve this repeat with Rock-Band algorithm[53].

4) Features of repetitive sequences. As shown in the supplementary file of SOAPdenovo2[36], genomes of different species have different content of exact copied sequences and different content of inexact copied sequences. The later one can be easily noised by sequencing errors and heterozygous regions when the mismatches or indels are accepted in overlap regions. Topology structures like "Tips", "bubbles" are recognized in *de Bruijn* graph assemblers including velvet[53] and SOAPdenovo[33] and are used to treat sequencing errors and assembly of heterozygous regions. Simpson in SGA also implemented this idea as mentioned in his PhD thesis. This treatment is really important to obtain high quality contigs.

In summary, when we know the sequenced genome is rich in repetitive sequences, we should improve the sequencing depth and read length to use larger k size in *de Bruijn* graph assemblers or larger minimum overlap size in overlap/string graph assemblers. However, not all the genome regions will be covered equally especially for the reasons of sequencing bias [5, 9] and most of the repetitive lengths are not the same, so same k size or minimum overlap size will not be the best choice for local assembly of different regions.

1.3 Application of bi-directional BWT

1.3.1 Introduction to bi-directional BWT

Barrow and Wheeler invented a data transformation for data compression named as Burrows-Wheeler transform (BWT) in 1994[7]. For a string T with length n over an alphabet Σ, we add a unique special character '\$' as the last character of T, which is the smallest among characters in Σ. Then we obtain all the suffixes of T, sort them lexicographically and obtain suffix array SA[0, n-1] of T such that SA[i] stores the starting position of the i-th-lexicographically smallest suffix. The BWT of T is a string with same length to T and BWT[i] is equal to T[SA[i]-1].

Later in 2000, Ferragina and Manzini used BWT to support string pattern matching[12]. Let S as the substring of T, and is the shared prefix of suffixes between $SA[i]$ and $SA[j]$. Here range [i, j] is named as SA range of string S. Given the SA range [i, j] of S, for string matching, the challenge is how to find the updated SA range [p,q] of cS, in which c is a character in Σ. Ferragina and Manzini proved that:

$$p = C(c) + occ(c, i-1) + 1$$
$$q = C(c) + occ(c, j)$$

Where $C(c)$ is the total amount of characters in T, which are smaller than c. Occ(c, i) is the total occurrence of c in BWT[0..i]. If p is no larger than q, cS is a substring of T. $C(c)$ could be pre-computed, and Occ(c, i) could be obtained in a constant time using the data structure introduced by Ferragina and Manzini[22]. With this text indexing method (named as FM-index), it is possible to obtain the SA range of cS in a constant time. This process is the backward search with BWT.

To support backward searching, forward searching and interleaving between them, which might be used in approximate pattern matching, Lam *et al.* introduced bi-directional BWT in 2009[23]. It is formed by BWT of T and BWT of T^R (the reversal of T). Then for substring S, we also define SA'

range of S^R w.r.t. T^R. Now for S, given SA range and SA' range, for any character c, they proved to obtain the updated SA range and SA' range of cS (backward search), as well as the SA range and SA' range Sc (forward search) in constant time. In this way, BWT could be used in NGS mapping to genome references using edit distance, which allows insertion and deletions, while keeping the cost of memory reasonably small.

Besides for reference sequences, many researches have been focusing on the construction of BWT of NGS reads. In 2010, Simpson firstly developed a method to building the FM-index for reads R and FM-index for the reversed reads R', after constructing the SA for R and SA' for R' respectively[47]. However, this method took 32 hours with 132 processors to construct the FM-index of near 40X human NGS reads. In 2014, Liu Chi-Man [35] developed a method to construct the BWT of human 30X sequencing data costing only 6 hours with the acceleration of GPU. BWT of NGS reads was firstly used for genome assembly and recently has been used for variation calling[20]. In this research, we used the improved version of BWT construction method of Liu Chi-Man to assemble NGS reads.

1.3.2 String matching with bi-directional BWT

Within the past many years, read mapping to reference sequences is the main application of BWT in bioinformatics. In 2007, Hon *et al.*[17] introduced a method to construct BWT and could construct BWT of human references within 50 min using <1 GB peak memory. Later Lam *et al.* developed BWT-SW in 2008[22] to speed up the Smith-Waterman local alignments between DNA sequences. In 2009, BWA from Li Heng[29] and SOAP2 (based on bi-directional BWT) from Li Ruiqiang[31] were developed to mapping NGS reads to human reference sequences. Both of them were times faster than previous hash-based NGS reads mapper MAQ[30] and SOAP[32]. Until now, BWT based aligners such as Bowtie2[25], BWA[29] and SOAP (SOAP2[31], SOAP3[34] and SOAP3-dp[37]) are the main tools for mapping NGS reads to reference.

The first application of BWT of reads is for read-read alignment. With the FM-index of reads R, for read X with length l, we can obtain the reads overlapped with the suffix of X and the overlap length is larger than minimum overlap length τ. Using backward search, we can start at $X[l\text{-}1]$ and find the SA ranges for $X[l\text{-}1]$, $X[l\text{-}2,l\text{-}1]$, $X[l\text{-}3...l\text{-}1]$, ... , $X[l-\tau...l\text{-}1]$ gradually. If the size of SA range [i, j] for $X[l\text{-}\tau...l\text{-}1]$ is larger than 0, then we say there are (j-i+1) reads overlapped with read X. This idea is used in SGA for collecting overlapped reads and constructing string graph directly[47].

1.3.3 Contig assembly with BWT of reads

Given the BWT of NGS reads, string graph method has already been implemented in SGA[47] and Fermi[28]. They both have the parameter minimum overlap size and take long time for contig assembly. MegaHit[27], a new assembler developed recently in our lab, used the BWT to construct *de Bruijn* graph. By utilizing the multi-kmer idea from IDBA[44], it can do meta-assembly quite well. Then the question is how to design an algorithm to better use the advantages of BWT of reads.

One of the most important advantage of bi-directional BWT of reads is that it can be used to obtain the SA ranges of any sequences (no longer than reads) in linear time and the size of SA ranges is the number of reads containing this sequence and its reversed-complement form. Given the length of a sequence k (<= read length l), I define the depth of this sequence d_k is the number of reads containing this sequence and is equal to the size of SA ranges in bi-directional BWT. As discussed above, we can also use this depth d_k to infer the uniqueness of sequence with length k. If this sequence is unique, it means this sequence only has one copy in genome. This means for any overlapped sequences among reads, it is possible to infer the uniqueness with bi-directional BWT.

In this way, with bi-directional BWT, it is possible to increase the threshold of minimum overlap size until it is longer than length of repetitive sequence in overlap region. Comparing to multi-kmer method, which using the discrete size of k-mers to solve repetitive sequences shorter than them, with bi-directional BWT, we can find the proper k size to solve repetitive sequences in a continuous way.

One of our aims is to use this advantage in our new assembler. The straightforward idea is to construct overlap graph or string graph. In order to tolerant sequencing errors within reads, we need to infer sub-sequences covering all the bases in reads. To reduce time complexity for large genome assembly, I imply the seed-extension strategy. Seeds have adaptive length to make sure they are inferred to be unique or say having only one copy in the whole genome. With bi-directional BWT, reads containing this seeds can be collected and recovered. While, instead of re-generate the read sequences, I build up a tree structure with branches to show the inconsistence among reads when backward searching with BWT. Then the extension process is to remove the branches in tree and obtain the final consensus sequences as newly extended regions. In the following chapters, the details of this method will be further explained and discussed.

1.4 Contributions

In this thesis, I proposed a new large genome contig assembler BASE, which fits for the longer NGS reads.

With bi-directional BWT, using seed-extension strategy, BASE finds seeds by increasing the length continuously until they are inferred to be unique in genome. In the assembly of each local region, the seed length is determined by balancing the requirement from the real sequencing depth and the local repetitive sequence length. This overcomes the shortage of fixed k-mer size in *de Bruijn* graph based assemblers and fixed minimum overlap size in overlap/string graph based assemblers, thus it can obtain relatively long contigs. Compared with traditional seed-extension assemblers, BASE acquired the *reverse-validation* idea to recognize the sequencing error, repetitive sequence and heterozygous sites caused special structures which are also recognized in *de Bruijn* graph based assemblers, so BASE can obtain accurate contigs. Compared with multi-kmer methods, in which k-mer size is practically increased discretely, by considering the local real sequencing depth, BASE avoid the over-treating of repetitive sequences with inefficient coverage for certain k size. So BASE can obtain contigs with higher accuracy while costing less time.

In practice, as shown in the following chapters of this thesis, assembly of deeply sequenced bacteria datasets suggested the high accuracy assembly of BASE and the performance on four human datasets showed that BASE performs better than contig assembly of SOAPdenovo2 especially on 250bp HiSeq dataset. Recent practices in BGI, the largest sequencing center in world, BASE can assemble some plant genomes and obtain contigs much longer than those from SOAPdenovo2. New version of BASE is still under-developed to obtain better assembly for a plant with 14Gbp genome size.

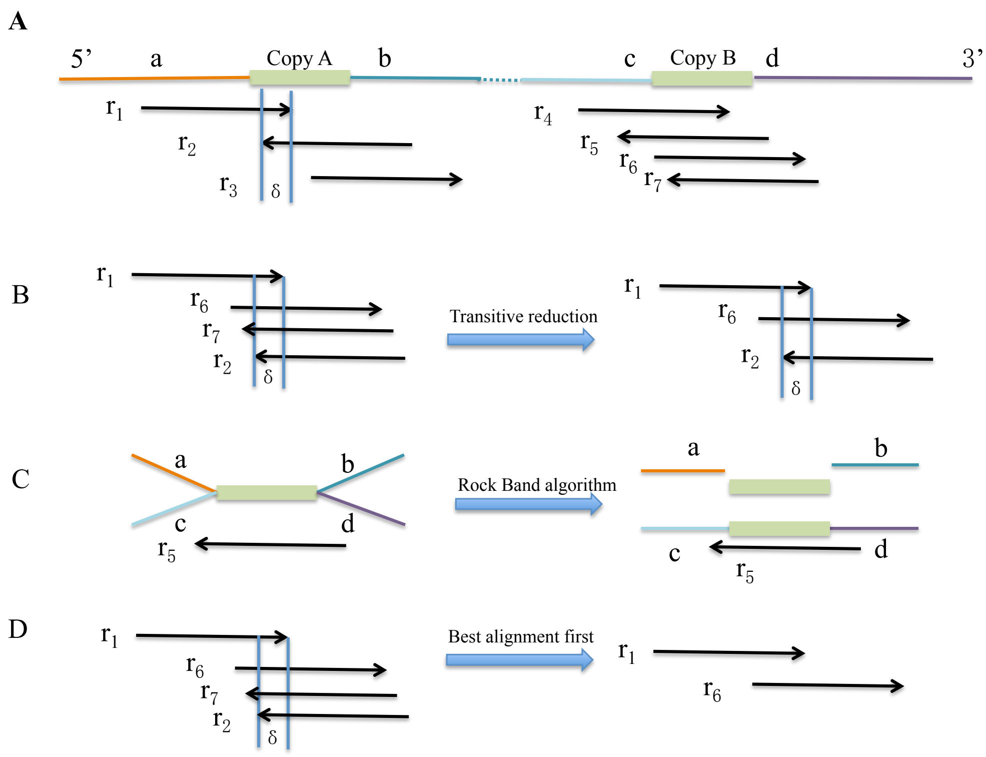

FIGURE 1.1: Comparison of three main assembly algorithms. (A) There is one repetitive sequence with two same copies (Copy A and Copy B) in the whole genome. After sequencing, we get three reads (r1, r2, r3) covering Copy A and four reads (r4, r5, r6 and r7) covering Copy B. The minimum overlap size is δ. (B) In overlap/string graph based assemblers, given minimum overlap size δ, for r1, there would be r2, r6 and r7 overlapped with it. Because r2 is not overlapped with r6 and r7 is overlapped to r6, after transitive reduction, only r2 and r6 are left for r1. For this confliction, contigs will stop here. This is the reason why minimum overlap size should be larger than the length of repetitive sequences. When the minimum overlap size is larger than the overlap size between r1 and r6, there will be no reads overlapped with r1 left and contig will also stop here. (C) In *de Bruijn* graph based assemblers, a butterfly structure will be formed around repetitive sequence, theoretically with two out and two in. Because r5 covers Copy B and connects branch *c* and branch *d* together, using Rock-Band algorithm, we can obtain a contig connecting *c*-Copy B-*d* together. (D) In seed-extension assemblers, for r1, we collect overlapped reads r2, r6 and r7. However, we dont know whether r2 is overlapped with r6 or r7. In the simplest extension strategy, we choose r6, which has the best alignment with r1. But this is sure a false extension for r1.

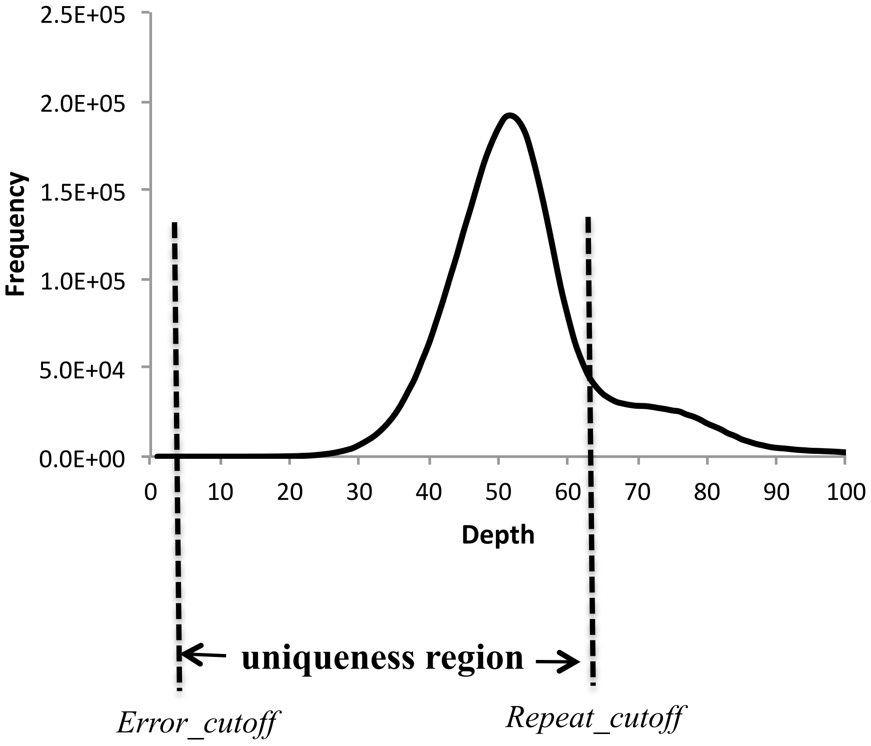

FIGURE 1.2: Depth distribution of k-mers in NGS reads. When the depth values are smaller than *Error_cuttoff*, k-mers might contain error bases. When the depth values are larger than *Repeat_cutoff*, they might have a quite high probability from the repetitive regions of the genome. Then the k-mers, of which the depth values fall in the uniqueness region, are *inferred-unique* k-mers.

Chapter 2

Algorithm

2.1 Preliminary

Given a set of reads R =R_0, R_1, ..., R_{n-1}, where each R_i has a length l and is terminated with a sentinel symbol '\$' (i.e., $R_i[l]$ = \$). For convenience, we also define R_i[-1] = \$, and for tie-breaking purpose, $R_i[l]$ is smaller than $R_j[l]$ if i is smaller than j.

Let Suff_R = R_i[j...l] | 0 < i < n and 0 < j < l be all possible suffices of reads in R. The suffix array SA_R of R is defined as SA_R[k] = (i, j) if R_i[j...l]is the k-th lexicographical smallest suffix in Suff_R. The BWT of R is an array defined by BWT_R[k] = R_i[j-1] if SA_R[k] = (i, j).

Given a string P, the SA range [$l_R(P)$, $u_R(P)$] of P w.r.t. R is defined as follows:

- $l_R(P)$ = min { k | SA_R [k] = (i, j) and P is a prefix of R_i [j...l] }
- $u_R(P)$ = max { k | SA_R [k] = (i, j) and P is a prefix of R_i [j...l] }

In any context, SA range with $l_R(P) > u_R(P)$ means that P is not a substring of any reads in R.

For any string P, let P' denote its reverse sequence and $RC(P)$ denote its reverse complement sequence. We store the BWT of R, as well as the revBWT of R' (the reverse of R). The SA' range of a string P is defined as [$l'_R(P')$, $u'_R(P')$]. Given the BWT of R and revBWT of R' forming bi-directional BWT, the following operation can be done in constant time[23]:

- For any string P and a character c in {A, C, G, T, \$}, calculate the SA range of cP from the SA range of P.
- For any string P and a character c in {A, C, G, T, \$}, calculate the SA range and the SA' range of cP (or Pc) from the SA range and the SA' range of P.

Due to the double-strand nature of DNA, we introduce the term intact SA range (ISR) of a DNA sequence P, which is the combination of: a) the SA range of P in R, b) the SA range of RC(P)in R, and c) the SA' range of RC(P) in R. Then the intact SA range is denoted by ISR(P) = [$l_R(P)$, $u_R(P)$, $l_R(\mathrm{RC}(P))$, $u_R(\mathrm{RC}(P))$, $l'_R(\mathrm{RC}(P))$, $u'_R(\mathrm{RC}(P))$]. With ISR(P), we define the depth of P (with respect to the set R) as follows:

$$\mathrm{Dep}(P) = \max\{0, u_R(P) - l_R(P)+1\} + \max\{0, u'_R(\mathrm{RC}(P)) - l'_R(\mathrm{RC}(P))+1\}$$

According to the functionality of bi-directional BWT described above, for P=P[0...m], the depth values of $P[m]$, $P[m\text{-}1...m]$, ..., $P[0...m]$ can be calculated incrementally by updating their ISRs. Symmetrically, the depth of $P[0]$, $P[0...1]$, ..., $P[0...m]$ can also be calculated incrementally by updating the ISRs of RC($P[0]$), RC($P[0...1]$), ..., RC($P[0...m]$) incrementally.

To make bi-directional BWT of reads better fit for genome assembly, the team in our lab also make the following modifications based on the published version[35].

(a) A base is encoded with 4 bits using the last 3 bits to encode A, C, G, T or the read terminal symbol \$, and the first 1 bit to indicate whether this base has a high sequencing quality with respect to a user-defined threshold.

(b) Read ID (2i, 2i+1) is defined by the i-th pair of reads, and an auxiliary table is used to record the mapping between a read ID and the position of the \$ in the BWT w.r.t this read. This enables fast recovery of read sequences and qualities in linear time. However, this method requires that all reads have equal length.

(c) A new CPU-only implementation was developed and is compared the GPU version.

2.2 Seed-extension assembly framework

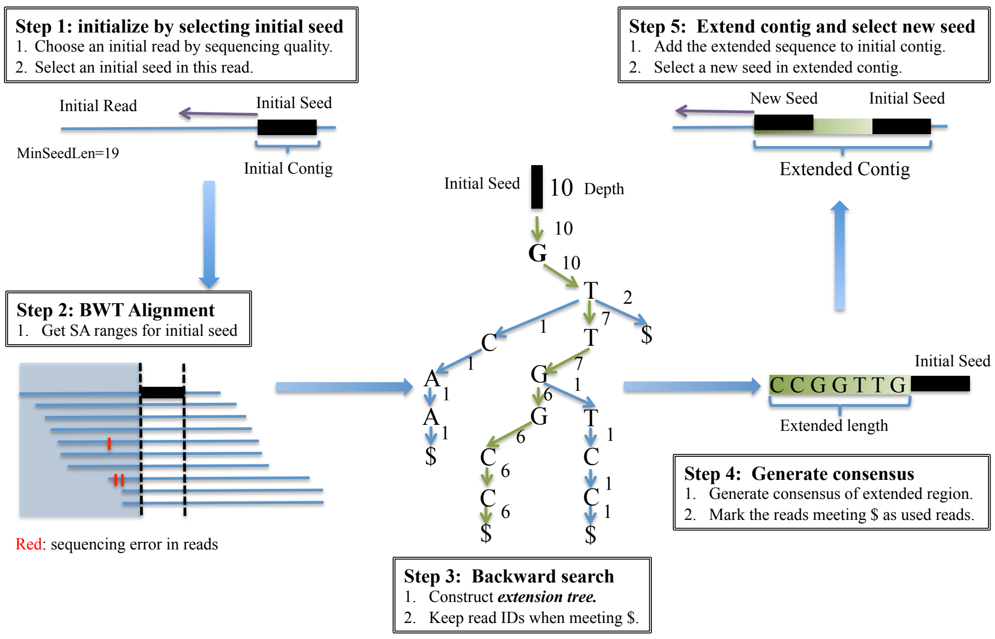
Step 1: initialize by selecting initial seed
1. Choose an initial read by sequencing quality.
2. Select an initial seed in this read.
Initial Read
Initial Seed
MinSeedLen=19
Initial Contig
Step 2: BWT Alignment
1. Get SA ranges for initial seed
Red: sequencing error in reads
Initial Seed
10 Depth
Step 3: Backward search
1. Construct extension tree.
2. Keep read IDs when meeting $.
Initial Seed
CCGGTTG
Extended length
Step 4: Generate consensus
1. Generate consensus of extended region.
2. Mark the reads meeting $ as used reads.
Step 5: Extend contig and select new seed
1. Add the extended sequence to initial contig.
2. Select a new seed in extended contig.
New Seed
Initial Seed
Extended Contig

FIGURE 2.1: Overview of the seed-extension strategy in BASE. There are five steps for one direction extension. Firstly, we choose an initial read by order and find an initial seed in this read. Then we use bi-directional BWT to get the SA ranges of this seed using backward exact matching. Thirdly, we build up a backward extension tree by adding bases to continue the backward matching. After removing erroneous branches and heterozygosis branches, we obtain the consensus sequence of the extended region. Finally, we continue to find a new seed in the extended region and extend iteratively.

To implement the idea invoked in Chapter 1, here I adopted the seed-extension strategy and developed a new assembler BASE. As shown in Figure 2.1, there are five steps: initialize contigs by finding an starting seed, obtain the SA ranges of this seed, construct an extension tree using backward searching with bi-directional BWT, generate consensus sequences from the extension tree and find a new seed in the extended regions to extend iteratively. Technically, these five steps could be divided into two parts: seed selection (including selection of initial seed and extended seed) and seed extension (including construction of extension tree and its simplification to obtain consensus of extended sequence). Then in the following paragraphs, these two parts will be discussed in detail.

2.3 Seed selection

An initial seed is a sub-sequence of a read and is used to initialize an extension. An extended seed is a sub-sequence of an extended sequence and is used to initialize a new iterative extension. The main idea of our seed selection strategy is to select the seeds that have only one occurrence in the genome to be assembled. In the context of *de novo* assembly, there is no way to calculate the exact number of occurrences of a seed in the genome. Then I developed the following method to guarantee a high probability to select one-occurrence seeds, which we call *inferred-unique* seeds.

Let d be the average sequencing depth of a genome, and each read has length l. Here I define the expected depth of a sub-sequence P with length k to be $d_k = (l - k + 1) * d / l$ [5]. If Dep(P) (which is the depth of P calculated according to ISR(P)) is no larger than z*d_k, in which z is a user-defined parameter, P is defined as an *inferred-unique* sequence, which means it is likely to occur only once in the genome.

To find an *inferred-unique* seed in a read R_i or a previously extended sequence, as shown in Figure 2.2, starting at the end and by using backward search mentioned above, we can update the ISRs and calculate the depth incrementally until it achieves *inferred-unique*. For example, we find a seed in read R_i of length l, we calculate the ISRs and depth of R_i[l-1], R_i[l-2,l-1],..., R_i[1...l-1] and R_i[0...l-1]. Meanwhile, We also update the expected depth d_{l-j} with j decreasing from l-1 to 0. Then there would be two cases for the changes of depth from these sub-sequences:

Case 1: The depth of R_i[j...l-1] is reduced to less than user-defined depth threshold τ. This region R_i[j...l-1] would contain sequencing error or too long repetitive sequence. Then we will further try to find seed in the substring R_i[0...j-1].

Case 2: The depth of R_i[j...l-1] is no larger than $z * d_{l-j}$. Then sub-sequence R_i[j...l-1] meets the requirement of *inferred-unique* and it will be treated as a seed, then no more sub-sequences will be checked.

Each *inferred-unique* sub-sequence will be further checked to make sure the all the bases in seed have high quality scores (using the 1-bit base quality stored in BWT). Finally, we can obtain a high quality *inferred-unique* seed to invoke the extension step.

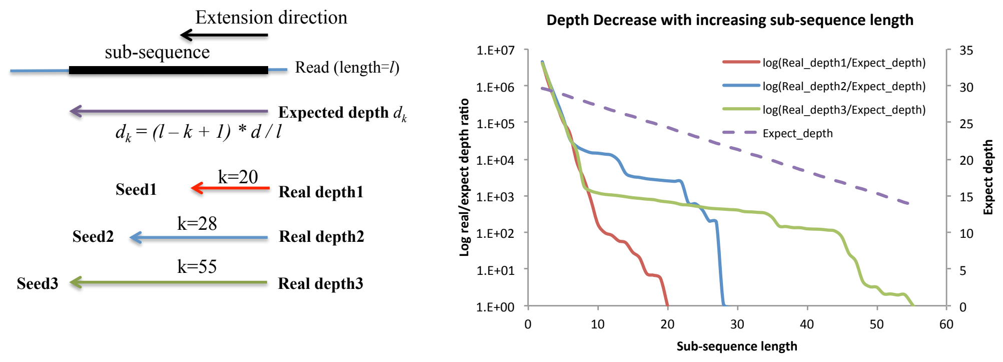

Figure 2.2: Decreasing of depth with the increasing of seed length. Using backward searching in BWT, with the increasing of k size, the real depths of seed1, seed2 and seed3 decrease gradually. In the meantime, we also calculate the expected depth d_k, and compare the real depth and expected depth in time. When the real depths are suddenly similar to their expected depths, we stop extensions to obtain the final initial *inferred-unique* seeds. For regions with different repeat features and length, we can obtain seed1, seed2 and seed3 with different length, which are longer than their repetitive sequences respectively.

2.4 Extension tree and its simplification

Given a pattern P with Dep(P) > 0 and a character c, we define cP as a valid backward extension of P if and only if Dep(cP) > 0. For a seed S, by adding characters in the head base by base, we can construct a backward extension tree T_s whose nodes are tagged with characters A, C, G, T, except for the root node, which is tagged with Seed S. The label of a node v, denoted by $L(v)$, is the concatenation of tags from v to the root; $W(v)$ is the weight of the node v which is equal to depth of $L(v)$.

Backward extension tree is built recursively. The root tagged with S is firstly created. For each newly created node v, if $cL(v)$ is a valid backward extension of $L(v)$ for some character c in A, C, G, T, a new node is created as a child of v and is tagged with c. Note that the label of a node will not be longer than the read length, the depth of the tree is limited by the read length minus the seed length. Moreover, for any node v in the tree, if Dep($\$L(v)$) > 0, we obtain the IDs of reads which have $L(v)$ as a shared prefix and mark these reads to avoid redundant assembly.

The consensus sequence for the backward extension tree is constructed by walking down the tree from the root to a certain node. This process is called *consensus-walk*. When visiting a node with only one child, the walk moves on to that child. Otherwise we have to select a branch to move on or stop immediately. A greedy algorithm, which chooses the child with the largest weight, is straightforward but error-prone. Therefore, we introduce another strategy, which we call *reverse validation*, to improve the probability of choosing the correct branch. For simplicity, we describe our method for the case of two branches. As shown in Figure 2.3, let v be the node that the consensus is currently processing to, a and b be two children of v, tagged with t_a and t_b respectively. Let C=$L(v)$ be the consensus sequence we have already constructed. The method incrementally calculates the depth of t_a, $t_aC[0]$, $t_aC[0...1]$, $t_aC[0...2]$, etc. and t_b, $t_bC[0]$, $t_bC[0...1]$, $t_bC[0...2]$, etc.

Below τ denotes a user-defined threshold. There are mainly two cases below to determine the child used for further extension.

Case 1. If Dep($t_aC[0...i]$) $< \tau$ and Dep($t_aC[0...i]$) > 0 for some i, we immediately conclude that a is an erroneous branch and b is authentic if it demonstrates the following properties:

- Dep($t_aC[0...i-1]$) is significantly larger than Dep($t_aC[0...i]$).
- Dep($t_bC[0...i]$)is significantly larger than Dep($t_aC[0...i]$).
- The expected depth d_{i+2} is significantly larger than Dep($t_aC[0...i]$).

Case 2. if Dep($t_aC[0...i]$) $= 0$ for some i, we conclude that the initial seed is a false positive *inferred-unique* seed and a is near another copy of this seed

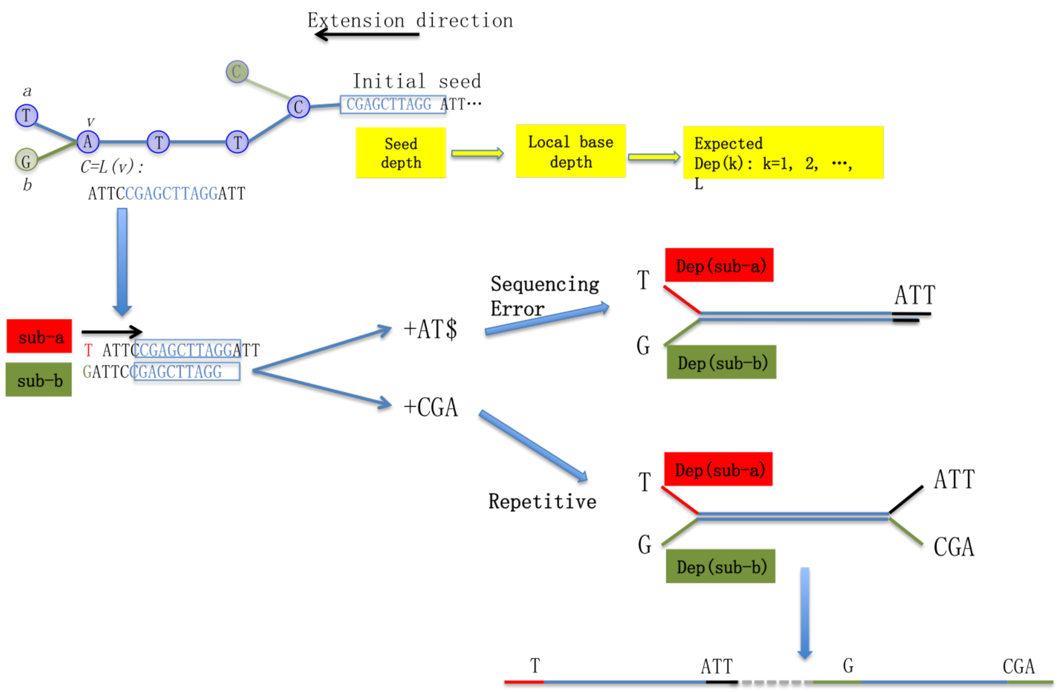

FIGURE 2.3: Remove branches in backward extension tree. In the backward extension tree, we try to remove erroneous branches, repetitive branches and heterozygosis branches to obtain the consensus sequences of the extended region. As an example, we meet node v with two child nodes a and b. Firstly, combined with $L(v)$, we obtained $TL(v)$ for a and $GL(v)$ for b to detect erroneous branches between a and b. We incrementally calculate the depth of sub-sequences of a(sub-a_i with length i): T, TA, TAT, ..., and b(sub-bi with length i): G, GA, GAT, ... until the depth of sub-a is less than user-defined threshold τ. At the same time, if Dep(sub-a_i) is significantly smaller than Dep(sub-a_{i-1}), Dep(sub-a_i) is significantly smaller than d_i and Dep(sub-b_i) is significantly larger than Dep(sub-a_i), then branch a will be treated as a erroneous branch or repetitive branch. When there is no erroneous signal, we will further try to remove the branch, which might be caused by heterozygous. After obtaining two sequences representing the consensus sequences of the sub-trees rooted at a and b respectively, we compare the two sequences to find the matched region and get the depth of it. Then we use this depth to calculate base depth and compare to the base depth calculated by depth of initial seed. If the two sequences have high similarity and the two depths are similar to each other, we will treat a as heterozygous branch if $W(a)$ is smaller than $W(b)$.

in the genome, and a is named as a repetitive branch if it demonstrates the following properties:

- Dep(t_aC[0...i-1]) is significantly larger than 0.

- Dep(t_bC[0...i]) is significantly larger than 0.
- d_{i+2} is significantly larger than 0.

If we fail to identify the above two cases, an additional step will be introduced to estimate whether the branches are due to heterozygous sites. Starting from a and b, we use a greedy algorithm mentioned above to obtain two sequences representing the consensus of the sub-trees rooted at a and b, respectively. If the similarity of these two sequences is high enough, we make a prediction that these two branches are caused by heterozygous sites, and walk to the child with larger weight. Otherwise, the *consensus-walk* stopped at node v.

If the *consensus-walk* does not stop at the root of the tree, i.e. the consensus sequence has been extent by at least one base pair from the seed, a new inferred unique seed will be chosen from the prefixes of the consensus sequences to start a new round. The process of seeding, backward extension and consensus is repeated until the *consensus-walk* stops at the root of the extension tree in some round. Then a series of symmetric processes follow, which forward extend the initial seed. Finally the contig containing this initial seed, which is the concatenation of the consensus sequences in both directions, is obtained when the forward extension completes.

2.5 Application of paired-end information

Most of the NGS reads for genome assembly are sequenced from the two terminates of DNA fragments forming paired-end reads. The length of the DNA fragment is named as insert size. Paired-end information was designed to solve repetitive sequences shorter than insert sizes. In the process of extension, we store the read IDs, which have been used in the extended regions. Paired-end reads have adjacent read IDs in bi-directional BWT. This makes it possible to use paired-end information to resolve longer repetitive regions.

When the *consensus-walk* stops at the root of the extension tree, as shown in Figure 2.4, suppose we have two children nodes a and b of root node v, reads with \$ falling in the sub-tree of a and sub-tree of b regions are divided into $R(a)$ and $R(b)$. We check whether the paired reads of $R(a)$ or $R(b)$ have been

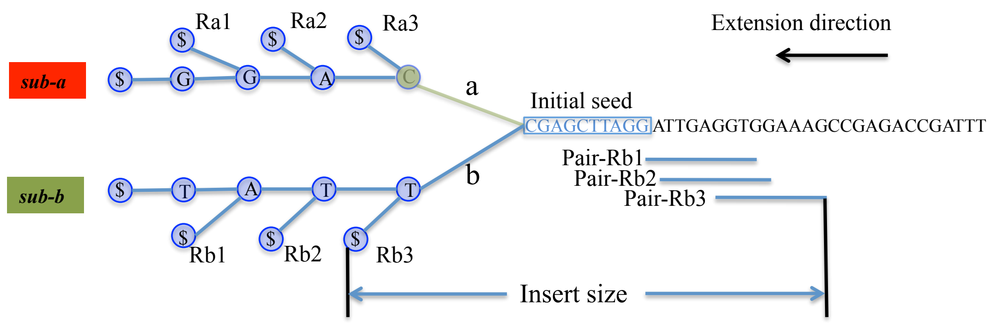

FIGURE 2.4: Application of paired-end information to remove branches in extension tree. This method will be invoked only when the above methods fail to remove the branches. And current branches are formed by the children nodes of root node. Starting from a and b, we use a greedy algorithm mentioned above to obtain the sub-trees rooted at a and b respectively, and keep the read IDs with \$ falling in each sub-tree. Then we check whether their paired reads have been used in extended regions. If the paired read ID(Pair-Rb3) is found in used read IDs and the distance between Rb3 and Pair-Rb3 falls in the 3*SD region of *Insert_size*, then we say branch b is supported by Rb3 paired-end relation. If branch a is not supported by any paired-end relations and branch b is supported by more than 2 paired-end relations, we will only keep branch b, and remove branch a, so the extension will turn to branch b.

used in the previous extended regions and whether the distances are proper as estimated by their positions in this contig and their insert sizes. Without loss of generality, if only paired reads of $R(b)$ are found and the number is larger than user-defined threshold τ mentioned above, the child node a will be removed.

This method could be used to assemble repetitive sequences longer than read length and obtain longer contig sequences. This is also an important point compared with previous seed-extension based NGS assemblers.

2.6 Complexity

Given base number N as the total length of reads, read number R as the count of reads, read length l, genome size G, Let c_i be the number of overlaps for read R_i, and C be the sum of c_i. Here I will discuss the complexity of our methods and compare it with other assemblers.

In overlap/string graph based assemblers, the first step is to calculate the complete set of overlaps among each read, which is the main bottleneck for this kinds of assemblers. Simply, it will take time $O(N^2)$. With q-grams mentioned in Mayers paper[42], a better performance could be achieved by a time-space tradeoff to complexity $O(N^2/D)$, in which D is the amount of available memory. In SGA, Simpson used FM-index to compute overlaps in $O(N+C)$ time[47]. Removing transitive is another challenge, but Simpson has already solve it costing $O(N)$ time. So overlap/string graph based assemblers still have the time complexity $O(N+C)$ for contig assembly.

In *de Bruijn* graph based assembler, such as SOAPdenovo, reads are firstly spliced into k-mers using $O(R*(l\text{-}k+1))$ time. Once the graph has been constructed, it will obtain a graph containing $O(G)$ nodes and $O(G)$ edges, for ideally repetitive-free genome and error-free sequencing data. Then the raw contigs can be assembled with $O(G)$ time. So totally, the time complexity is $O(R*(l\text{-}k+1))$.

For BASE, the backward searching for a seed is $O(s)$ when the length of seed is s. To construct the extension tree, for its maximum depth is l-s, it only need $O(4*(l\text{-}s))$ time. To simplify the extension tree, we need maximum time O(l). Let x to be expected distance between two adjacent seeds, we need to find $O(G/(x+s))$ seeds and finish $O(G/(x+s))$ extensions, so the global time complexity is $O(l*G/(x+s))$. Then, in the worst case, we can only extend for one base each time, we need to extend for $O(G)$ times and the time complexity would be $O(l*G/(s+1))$. This would happen if the sequencing depth N/G is really high or the sequencing quality is too poor. Generally, $l/(x+s)$ would be less than 10, so the time complexity will be smaller than those of *de Bruijn* graph based assemblers and string graph based assemblers.

Chapter 3

Implementation

I implemented the method mentioned above in BASE, to assemble longer NGS reads. Before assembly, we firstly need to construct the bi-directional BWT with Liu Chi-Man′s method. To make it convenient, it can load the assembly configure file for SOAPdenovo. Then given the bi-directional BWTs, we can use BASE to assemble large genomes or deeply sequenced bacterial genomes. In the process of implementation, to assemble large genomes including human genome, we mainly treated two problems to improve the contig coverage by controlling the abundance and the speed to assemble with multiple threads. These two points will be further explained in the following paragraphs.

3.1 Abundance controlling

In overlap/string graph, all the reads will be treated as nodes. In *de Bruijn* graph based assemblers, k-mers are treated as nodes. In these assemblers, to obtain contig sequences is to traverse all the nodes. However, in seed-extension method, there are no such nodes and its a challenge to determine when the read could be marked as used and when to stop finding a new seed.

One simple method is to mark the reads collecting by seed as used. When all the reads are marked as used, we say the assembly has been finished. However, in our method, there are surely false positive unique (FPU) seeds, as shown in Figure 3.1. These seeds are falsely inferred as unique sequences, and the reads

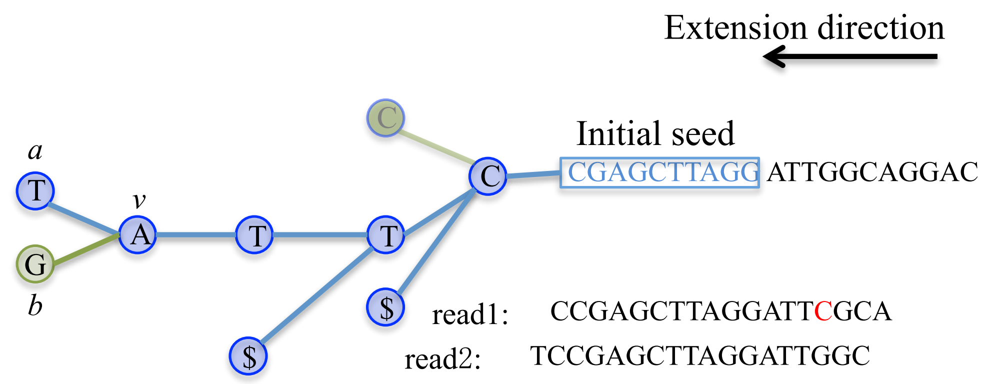

FIGURE 3.1: Marking used reads in seed-extension. For read1 and read2, we will recover the part after initial seed region, such as ATTCGCA and ATTGGC. Because there is a mismatch between read1 and extended sequence, read1 is not marked as used, but read2 is marked as used. When the initial seed is a true positive unique seed, then read1 will be treated as reads containing sequencing error, and would not form another extended region longer than read length. When the initial seed is a false positive unique seed and the mismatch C is from another copy region of initial seed, then there is a possibility to assemble another copy when the near-C region is covered by enough reads.

might be incorrectly collected and marked as used, which will result in the low genome coverage of assembled contigs. This problem will become worse in the following cases:

1) Reduce the starting overlap size to 19bp by default for large genome. It means, there might be seeds with 19bp and more seeds might be inferred falsely as unique. Then more reads will be falsely marked as used.

2) Increase the read length. When the sequencing depth is not changed, the amount of reads will decrease for longer reads. Then more incorrectly marked reads will cause more regions fail to be assembled, thus reduce the genome coverage of contigs.

3) For genomes with higher repetitive content, such as plant genomes, or when the sequencing bias is a bit severe, the rate of FPU reads will increase, resulting lower genome coverage of assembled contigs.

To maintain the genome coverage of contigs, we need to avoid marking the reads, which are inconsistent to the regions of assembled sequences. As shown in Figure 3.1, we recover the regions of reads behind the seed, and compare with extended regions to check the consistence. The advantage of this treatment is to keep reads from another repeat copy as unused and could be assembled later.

With this treatment, reads containing sequencing errors or heterozygous regions will be not marked, until finding initial seeds in these reads. Even though seeds could possibly be found in these reads, it is not easily to obtain extended regions with length up to read length. So to control assembly abundance, extended sequences only with length no less than l will be kept as assembled sequences. Another shortage of this treatment is the time for assembly will become longer. Theoretically, we need to recover all the reads containing the seed, and compare parts of them to the extended region. This part could be improved in the future.

3.2 Multiple threads

To speed up contig assembly, I also developed the multi-thread version, which mainly contains three parts. The first part is to find a read, in which we can find a initial seed. The second part is to mark a read as used and the third one is to remove the marks of reads used in the thread with lower priority when it meets another thread.

To improve efficiency, with the help from our lab, I used compare_and_swap(CAS) operation to avoid locking in threads. Then the challenge is how to recognize and treat the thread conflictions. Likely to graph-based assemblers, when two or more threads treat one read at the same time, the thread with higher priority will be continued and other threads will be terminated. To support these operations, as shown in Figure 3.2, we used the unused bits in bi-directional BWT, by dividing them into quality-bits, thread-bits, used-bits and current-bits. For one read with length l, there will be l bits left in BWT and l bits left in revBWT. l-1 bits in BWT are used to keep the quality bits, in which 1 means the base has high sequencing quality by a user-defined threshold. The last eight bits in revBWT are used to keep the used mark (whether the read

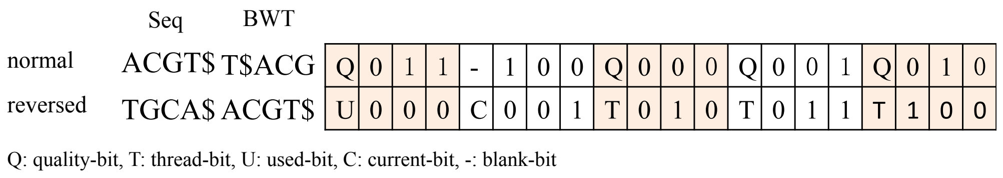
Seq
BWT
normal
ACGT$ T$ACG
Q 0 1 1 - 1 0 0 Q 0 0 0 Q 0 0 1 Q 0 1 0
reversed
TGCA$ ACGT$
U 0 0 0 C 0 0 1 T 0 1 0 T 0 1 1 T 1 0 0
Q: quality-bit, T: thread-bit, U: used-bit, C: current-bit, -: blank-bit

FIGURE 3.2: Bits for bi-directional BWT. For each read, there is a BWT for the read sequence and a revBWT for its reversed sequence. Each character in BWT or revBWT is stored with four bits, with last three bits for characters A,C,G,T and $. The first bits in BWT store sequencing quality information for each character, except that the first bit for $ is unused currently. The last six first-bits in revBWT store the thread id which is transformed into bits (*thread_id*<64), the last seventh first-bit is used to mark whether the read is currently under-processed in one thread and the last eighth first-bit is used to mark whether read is used in assembly before. So for 100bp read, 92 first-bits in revBWT is still blank-bits. If a used read is met in the extension of one thread, with current-bit, we can determine whether there are two threads meet together.

have been assembled already), current mark (whether the read is being used in one thread) and thread id.

In the first part, to find an initial read, which might contains an initial seed to start an extension, we need to check whether the read has been used or not. To avoid meeting the same read in different threads and to make it easy to use multiple BWTs, each thread only checks the *thread_id**i-th reads in BWTs (i is a integer). Then we get the values in the last eighth first-bit in revBWT to check whether this read has been assembled. If it has been assembled, then this thread will evaluate *thread_id**(i+1)-th read.

In the second part, when one extension has been finished and the checked reads should be marked as used, we mark the used-bit, current-bit and thread id-bits at the same time with CAS operation shown as Figure 3.3.

In the third part, we need to recognize the case multiple threads meet at one read. Here when one read is going to be marked in one thread, we firstly recognize whether it is already assembled with used-bit equal 1. If yes, one case is that it has been used in one thread and the extension has finished with current-bit equal 0. Then we only stop extension of current thread because we might meet a region, which has already been assembled. The other case is that it is used in another thread and the extension in this thread has not

```
set_read_sparse_thread ( idx2BWT , id , threadId )
bwt <- idx2BWT -> rev_bwt
new_word <- ((unsigned int)(1) << 31) | ((unsigned int) (1) << 27)
new_word |= id_to_sparse_word[ threadId ]
blank_word <- ~((unsigned int) (id_to_sparse_word[ 63 ]))
while (1)
        oldvalue <- bwt -> bwtCode[ id ]
        if(bwt->bwtCode[ id ] >> 31)
                return 0
        newvalue <- oldvalue & blank_word | new_word
        if( __sync_bool_compare_and_swap ( &(bwt->bwtCode[ id ]), oldvalue, newvalue) )
                return 1
```

Figure 3.3: CAS operation to mark read as used.

yet finished with current-bit equal 1. In this case, we need to further check whether this read has just been used in current thread before, so we obtain the thread id kept in thread-bits and compare it with current thread id. If the two thread ids are the same, we need to stop extension for current thread to avoid endless loop. If they are different, this means two threads meet at this read. So we stop current thread, clear the used-bits and current-bits of reads used in current thread using CAS operation. To support this operation, we temperately keep the read ids, their positions in current extending contigs. This will also support the application of paired-end information mentioned in Chapter 2. The initial read of this thread will invoke a new thread later if it is still not used then.

The multiple threads version is especially useful to assemble contigs of large genomes. Because it is still quite difficult to terminate a thread by recognizing the sufficient enough coverage of genome by assembled contigs, it is suggested not to use the multiple threads version when assembling small genomes.

Chapter 4

Benchmarks and evaluation

4.1 Datasets

To test the performance of BASE, I used several sets of real data, including two bacterial *Staphylococcus aureus MW2* 240X HiSeq 100bp reads (SRR857914)[16], *V. parahaemolyticus* 240X MiSeq 250bp reads (DRX016227)[40], and four human sequencing data sets including YH Solexa 100bp reads[36], YH HiSeq 150bp reads (BGI), NA12878D HiSeq X Ten 150bp data (DNAnexus.com) and NA12878 HiSeq 250bp data (SRR891258, SRR891259). All raw sequencing data were pre-processed with SOAPfilter[36] to remove reads containing excessive amount of Ns or adapters, low quality reads and duplicated reads. The four human datasets were further corrected with SOAPec[36] using 23-mer.

4.2 Evaluation

Using reference genomes for *Staphylococcus aureus MW2* [1] and *V. parahaemolyticus* (RIMD2210633), I evaluated the accuracy of assembly using the GAGE pipeline[39], in which metrics such as correct N50 size, mismatch, align rate and coverage were assessed. N50 value is the length of the shortest contig such that the total length of contigs no shorter than it can cover 50% of the genome. If the reference genome is known, after mapping contigs to reference sequences, we break contigs at each error site, calculate N50 of these broken sequences, and get correct N50. Align rate is the length of contigs, which could be mapped to reference sequences, divided by the length of all contigs. Low align rate generally means there are false assembled contigs. Coverage is the length of reference regions, to which contigs can be mapped, divided by the total length of reference sequences. Low coverage generally means some regions are failed to be assembled. For YH and NA12878, I mapped the assembled contigs to Hg19 with LAST[19] and evaluated the alignment rate, reference coverage and repeat-masked reference coverage. To prove the performance of BASE, I compared the assembly performances of BASE to those of some popular assemblers, including SGA (first string graph NGS reads assembler), SOAPdenovo2 (a popular large genome *de Bruijn* graph based assembler), SPAdes (a powerful multi-kmer *de Bruijn* graph based assembler).

4.3 Contig assembly of deeply sequenced bacterial genomes

I randomly fished 240X from the raw datasets for two bacterial genomes. Then I tried different parameters for SOAPdenovo2 and SGA to obtain their best assembly results.

As shown in the bacterial assembly (Table 4.1), BASE obtains contigs with the highest accuracy among all evaluated assemblers and is the only assembler that achieve 100% alignment rate. Except four trans-locations of SPAdes in dataset of *V.para*, trans-locations assembled by BASE, SGA and SOAPdenovo2 are all caused by circular DNA and are not shown.

For the 100bp dataset of *S.aureus*, the correct N50 statistics of BASE is much shorter than that of SPAdes and is only a bit longer than that of SGA and SOAPdenovo2. Further analysis showed that BASE's improvement over SGA and SOAPdenenovo2 is mainly due to the usage of paired-end information. For the 250bp dataset of *V.para*, the correct N50 from BASE is indeed comparable to that of SPAdes and is much longer than that of SGA and SOAPdenovo2. Increased minimum overlap size to 149bp also increased the assembly length of SGA, but larger minimum overlap size will further reduce the assembly

Table 4.1: Contig assembly of deeply sequenced bacterial genomes

	Tools	Parameters	Correct N50	Mismatch /Indel	Aligned Rate(%)	Coverage (%)	Time (sec)
	SPAdes	51,63,85	299,305	134/6	99.79	100.00	1239
S.aureus MW2	SOAP2	87-95	82,495	40/0	99.84	99.27	25;16
(240X, 100bp)	SGA	29;91	74,584	7/0	99.81	99.98	1228;1149
	BASE	4	**92,706**	**0/0**	**100.00**	**99.97**	**161; 93**
	SPAdes	33,55,65,75,85,99	169,978	118/45	99.97	99.97	4616
V.para (240X,	SOAP2	125	88,858	23/30	99.98	99.98	110;1
250bp)	SGA	29;149	95,711	58/26	99.80	99.97	2478;2884
	BASE	4	**159,715**	**29/29**	**100.00**	**99.75**	**676;388**

S.aureus MW2 has its real reference with length 2.8Mb and *V.para* has its species reference with length 5.1Mb and two chromosomes. GAGE validation pipeline was used to calculate the corrected contig N50, base errors, structural errors, contig aligned rate and reference coverage. Except BASE used single thread for contig assembly part, and other the assemblies were all performed with 24 threads. The time before semicolon is for index building and after semicolon is for assembly. For SGA, indexing time contains the time used in the indexing after error correction and filtering; assembly time contains the time used in the overlap and assembly. In this table, we use SOAP2 to stand for SOAPdenovo2.

length (not shown). For SOAPdenovo2, currently the longest k size could be 127mer, which is still shorter than 149bp used in SGA.

SPAdes can obtain the longest assembly, with the most amounts of mismatches and costing the longest time as expected. As shown in Table 4.1, BASE takes slightly longer time in building index and assembling contigs than SOAPdenovo2, but is much faster than SPAdes and SGA. The coverage of contigs from BASE is relatively low, which could be improved by devoting more time to initialize more extension or by scaffolding like SOAPdenovo2.

In summary, for deeply sequenced bacterial, BASE can obtain long assemblies, with high accuracy and quite fast speed, showing its can achieve better balance of assembly metrics on longer NGS reads.

4.4 Contig assembly of human genomes

I further tested human genome assembly with four datasets: YH 100bp 35X, YH 150bp 63X, NA12878 XTen 150bp 35X and NA12878 HiSeq 250bp 45X. With 30X 100bp reads, it already took SGA more than 2 days [15] and Fermi nearly five days [16] to output the contigs. So here I only compared BASE with SOAPdenovo2.

TABLE 4.2: Summary of human contig assembly

	YH,100bp		YH,150bp		NA12878, 150bp		NA12878, 250bp	
	SOAP2 k=41	BASE	SOAP2 k=61	BASE	SOAP2 k=41	BASE	SOAP2 k=61	BASE
Contig num(M)	3.42	3.32	2.28	2.15	8.07	1.93	1.42	1.51
Contig size(Gbp)	2.67	2.88	2.76	2.95	2.44	2.90	2.60	2.94
Contig N50	2,244	2,279	3,008	3,126	1,140	3,823	3,368	4,199
Contig aligned rate(%)	99.10	97.07	98.87	95.96	99.40	97.62	99.34	96.33
Genome coverage(%)	90.36	93.76	93.12	93.90	84.11	95.58	89.55	94.09
RepeatMasked coverage(%)	97.05	96.13	97.28	95.32	93.94	97.38	95.60	95.99
Exon coverage(%)	93.76	91.51	95.73	94.13	91.48	96.84	93.90	91.49
Mismatch base(Mbp)	2.74	3.48	2.91	3.84	2.30	3.46	2.54	3.75
Mismatch ratio(%)	0.103	0.121	0.105	0.130	0.094	0.119	0.098	0.128
Indel num	340,930	327,469	358,358	334,989	259,190	322,214	327,695	372,941
Indel base(Mbp)	1.41	1.59	1.69	1.74	1.09	1.60	1.40	1.95
Indel ratio(%)	0.053	0.057	0.062	0.061	0.045	0.057	0.054	0.069

We mapped the raw contigs to Hg19. Aligned rate is the contig-aligned length divided by total contig length. To calculate genome coverage, the length of gap regions in Hg19 has been removed. For unique coverage, the repetitive regions have been further removed. For SOAPdenovo2 contig assembly, we all used single-kmer method and M1 to treat heterozygous regions. In this table, we use SOAP2 to stand for SOAPdenovo2.

TABLE 4.3: Mismatches analysis for human genome assembly

Dataset	Assembler	Whole genome				Whole Exon			
		Total	VariantCall	PublicSNP	Novel	Total	VariantCall	PublicSNP	Novel
YH	SOAP2	2,735,141	2,423,482	108,044	203,615	47,926	40,701	1,590	5,635
100bp	BASE	3,479,046	2,872,396	208,351	398,299	47,515	42,124	1,724	3,667
YH	SOAP2	2,911,990	2,613,670	113,037	185,283	46,002	43,151	1,148	1,703
150bp	BASE	3,839,110	3,075,913	256,217	506,980	52,561	46,420	1,822	4,319
NA12878	SOAP2	2,301,111	2,025,220	109,497	166,394	39,702	35,644	1,740	2,318
150bp	BASE	3,459,648	3,052,269	129,933	277,446	49,711	45,361	1,151	3,199
NA12878	SOAP2	2,554,785	2,122,144	130,806	291,835	42,744	35,890	1,936	4,918
250bp	BASE	3,751,887	2,613,065	604,805	534,017	48,635	37,853	5,068	5,714

We mapped the assembled contigs to Hg19 and got the mismatches between each contig and reference sequence. Then we used the detected SNPs and SNPs from published SNP databases to analysis these mismatches in whole genome and exon regions respectively. "Total" means total number of mismatches between contigs and reference sequences, of which "VariantCall" could be detected by GATK-UnifiedGenotyper pipeline. For the left mismatches, published SNP of YH(from BGI) and NA12878(from Illumina) further covers "PublicSNP" mismatches and the left uncovered are noted as "Novel", which are probably caused by mis-assembly. In this table, we use SOAP2 to stand for SOAPdenovo2.

In all four human datasets, shown in Table 4.2 the N50 statistics of BASE improves as read length increases, while SOAPdenovo2 does not show such degree of improvement. BASE′s improvement over SOAPdenovo2 becomes significant for 250bp reads. Similar to bacterial assembly, BASE′s genome coverage, with repeat masked, is lower that of SOAPdenovo2. But BASE has a overall higher genome coverage in each dataset. This suggests that BASE is able to assemble more repetitive regions.

Table 4.4: Performance for human genome assembly

		SOAPdenovo2			BASE		
		Wall Time (h)	CPU time (h)	Peak Mem (GB)	Wall Time (h)	CPU time (h)	Peak Mem (GB)
YH (36X, 100bp)	Index	4	46	163	5	18	200
	Contig	2	2	41	4	53	140
	Total	6	48	163	9	71	200
YH (64X, 150bp)	Index	6	75	201	11	36	192
	Contig	1	1	33	5	80	225
	Total	7	76	201	16	116	225
NA12878D (30X, 150bp)	Index	9	141	**477**	9	34	194
	Contig	1	1	24	7	144	142
	Total	10	141	**477**	16	178	194

For X Ten data, we used a different machine with larger memory to finish SOAPdenovo2 and BASE assembly, so it is improper to compare the time usage of this dataset to other dataset. Other datasets were all performed in the same machine with 24 CPU.

As shown in Table 4.3, assemblies of BASE usually have larger amount of mismatches, most of which are consistent to SNPs detected by GATK-UnifiedGenotyper pipeline, and published SNPs(for the same set of reads). The left unknown mismatches are generally considered to be caused by mis-assembly. However, for more repetitive regions are assembled, BASE might generate more mismatches and indels, which are not easy to be proved whether they are new assembly errors or not. Further analysis shows that in exon regions, amounts of mismatches especially novel ones are quite similar for BASE and SOAPdenovo2.

As shown in Table 4.4, for the 35X 100bp YH dataset, both BASE and SOAPdenovo2 (in single-kmer mode) took only about half a day to obtain the contigs. To assemble X Ten data (150bp reads), BASE used much less memory than SOAPdenovo2 on indexing and contig assembly (Table 4.4). This is probably due to the high error rate of the X Ten data, as shown in 17mer depth distribution of the three datasets in Figure 4.1. For the reason of sequencing error, there is some inconsistence between theoretical complexity and real time consuming.

4.5 Influence of read length on assembly

Currently, the sequencing read length has grown to 250bp for Illumina HiSeq and more than 300bp for Illumina MiSeq. Then question is how long reads

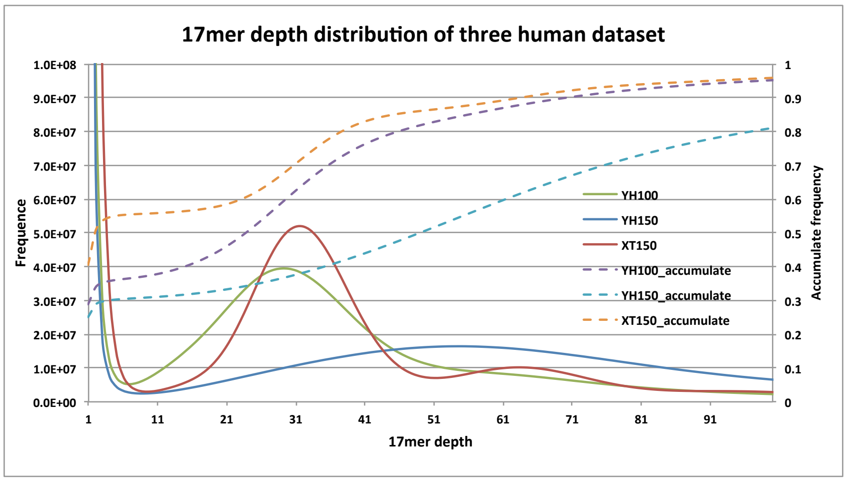

FIGURE 4.1: 17mer depth distribution of three human sequencing dataset. After counting the depth of all 17mers in the sequencing reads, we can calculate the frequency of each depth. About 35% 17mers of YH 100bp reads, 30% 17mers of YH 150bp reads and 53% 17mers of NA12878D XTen reads having depth no more than 3. Then we say the NA12878D XTen reads should have more sequencing errors left after raw data filter and correction than other datasets.

BASE could assemble and whether the performance will become worse with the growing of read length.

I simulated 30X reads with length from 100bp to 1kb for *E.coli* genome, using wgsim[2]. Then I constructed the bi-directional BWT of these reads and assembled contigs with BASE respectively. As shown in Table 4.5, it took more time to construct the bi-directional BWT of reads with increasing of read length. But the time usage was quite stable for contig assembly. As expected, contig N50 increased with the growing of read length, especially when the read length was longer than 250bp. It is necessary to mention that for different species, this value would change for the different lengths distribution of repetitive sequences.

TABLE 4.5: Assembly of *E.coli* simulated data

Read length	Insert size	Read number	CPU Index time(sec)	GPU Index time(sec)	Contig time(sec)	Contig N50	Coverage (%)
100	170	675,000	57.46	50.539	32.9	62,872	98.59
150	250	450,000	71.36	58.776	29.89	68,555	98.62
250	450	270,000	101.72	84.391	28.53	133,550	98.86
500	950	135,000	185.27	124.033	28.1	178,941	99.42
750	1,450	90,000	263.03	211.259	28.48	179,327	99.63
1000	1,950	67,500	348.04	272.534	33.01	191,128	99.60

CPU Index time stands for the time used by improved CPU version to construct bi-directional BWT.
GPU Index time stands for the published GPU version to construct bi-directional BWT.

Chapter 5

Conclusion

The primary objective of this research is to study whether a seed-extension approach to contig assembly, coupled with reverse validation, can give a performance (accuracy and N50) comparable to SOAPdenovo2 and SGA. As shown in the previous section, the new approach gives clear advantage for longer reads, and with speed much higher than SGA and comparable to SOAPdenovo2, and stable memory usage (i.e., not sensitive to error rate of the reads). The contigs obtained by BASE are longer and cover more repetitive sequences than those from SOAPdenovo2 and SGA.

Based on the high quality contigs assembled by BASE, one could use less accurate third generation reads or paired-end reads with longer insert size for scaffolding and gap closing. This approach has been used in a recently published assembler DBG2OLC[52], which assembles second level contigs onto high accurate DBG contigs. Indel or SV could also be detected with these contigs using established methods[28].

With the increasing length of high quality Illumina reads, it is of computational interest how to fully utilize the read length information in contig assembly. SGA, Fermi and our tool BASE both build an index of the reads and make it possible to assemble high-depth short reads without splitting them into k-mers. Although SGA and Fermi could finish assembly with less memory, they need much longer time. As noted in MEGAHIT[27], the requirement for big memory machine can be circumvented. For future bioinformatics analysis including assembly, it is time and robustness that matter most. I plan to further reduce the running time of BASE and build up the pipelines for BASE to support assembly solutions for especially large genomes.

Chapter 6

Discussion and Future Work

High quality contig assembly is the first step for the applications of NGS data. In the following paragraphs, I will describe the genome assembly solutions using BASE, one with paired-end reads and the other with third generation sequencing reads. These will be implemented and used in the quite near future.

Compared with SOAPdenovo2, which we have published in the end of 2012[36], BASE is designed especially for the increasing sequencing length of NGS data. In the process to assemble oyster genome[54] and YH fosmid[8], we have met a great challenge to obtain the high quality scaffolds using paired-end NGS reads and highly overlapped contigs. Contigs from BASE have longer expected overlap lengths than contigs from SOAPdenovo2, so it is also facing the challenge to obtain better scaffolds. To solve this problem, as shown in Figure 6.1, we designed and implemented the *Uniqer* and paired-end reads *Mapper* for BASE. *Uniqer* is to find the minimum unique length for each position in BASE contigs, and *Mapper* ensures that the mapping length of short reads to contig should be no shorter than this minimum unique length. The idea of minimum unique length was already used in the mutation detection tool using BWT of reads[20]. In this way, we can obtain high quality contig relations for scaffolding, by avoiding false mapping caused by repetitive sequences. After obtaining high quality contig relations, we use *SOAPscaf* to build the scaffolds, which is separated from SOAPdenovo2 and modified to fit for the feature of BASE contigs. In our immature practice, we can obtain the

10M scaffolds N50 for YH genome using dataset YH100bp. More improvement of this part will be further developed in the future.

With the development of the third generation sequencing (TGS) technologies like PacBio and Nanopore, more and more genomes are also sequenced and willing to be assembled with them. The advantage of TGS reads currently is kbp read length. But the shortage is their poor sequencing accuracy 85%[21] or even much lower[26, 38]. To directly assemble Nanopore raw data with Celera Assembler, it might even fail to output a single contig[15]. So, by using NGS and TGS reads together, there are two main candidate solutions for this hybrid assembly. One is to use NGS reads to correct the sequencing errors in TGS reads[3, 21, 38]. The other one is to assemble NGS reads at first and then map the assembled sequences to TGS reads to further assemble second-level contigs[51, 52] or close the gaps within scaffolds[11].

In the first solution, the earliest research directly mapping the NGS reads to TGS reads, which costs lots of computation and memory[21] and is improper for large genome assembler. Later, researchers in BGI tried to use SOAPdenovo assembled contigs to correct PacBio reads, which is much more efficient. In Madoui's research[38], NGS reads were also firstly mapped to Nanopore reads and named them as *seed-reads*. Then, seeds is found in these reads and is used to recruit more similar NGS reads as *recruited-reads*. These two types reads are assembled with OLC methods to obtain contigs, these contigs will build a contig graph and weighted by their relations in Nanopore reads. In this way, Madoui's method could be treated as the combination of the two solutions, but one of the key problems is still the assembly of NGS reads.

Most of the left un-assembled large genomes are generally not easy to treat, such as Peony with 14Gbp genome size and amounts of repetitive sequences. For these species, only sequenced by NGS reads can not obtain an acceptable results and most projects aim to hybrid assembly introduced above. To provide an efficient solution for hybrid assembly of large genomes, our efforts are focusing on two points. The first is to assemble NGS reads in high quality with BASE. Given 250bp and 150bp reads, BASE is expected to obtain longer contigs than SOAPdenovo2 and PacBio reads could be better mapped to them. This mapping efficiency is surely higher than mapping NGS reads to TGS reads. The other effort is *Uniqer*. Most of confusions in assembly are caused by repetitive sequences. And it is really a challenge to recognize the repetitive sequences in NGS reads, TGS reads and NGS assembled contigs. *Uniqer*, using the bi-directional BWT of NGS reads, with the uniformity hypothesis of sequencing, can be used to infer the minimum unique length in any sequences. This should be able to reduce the repetitive-caused false mapping, thus plays a positive role in the following correction of TGS reads, or second-level assembly of contigs, or close the gaps within scaffolds.

In summary, it is possible to support a whole large genome assembly solution with BASE and I will put more effort to use BASE to solve the assembly of more genomes.

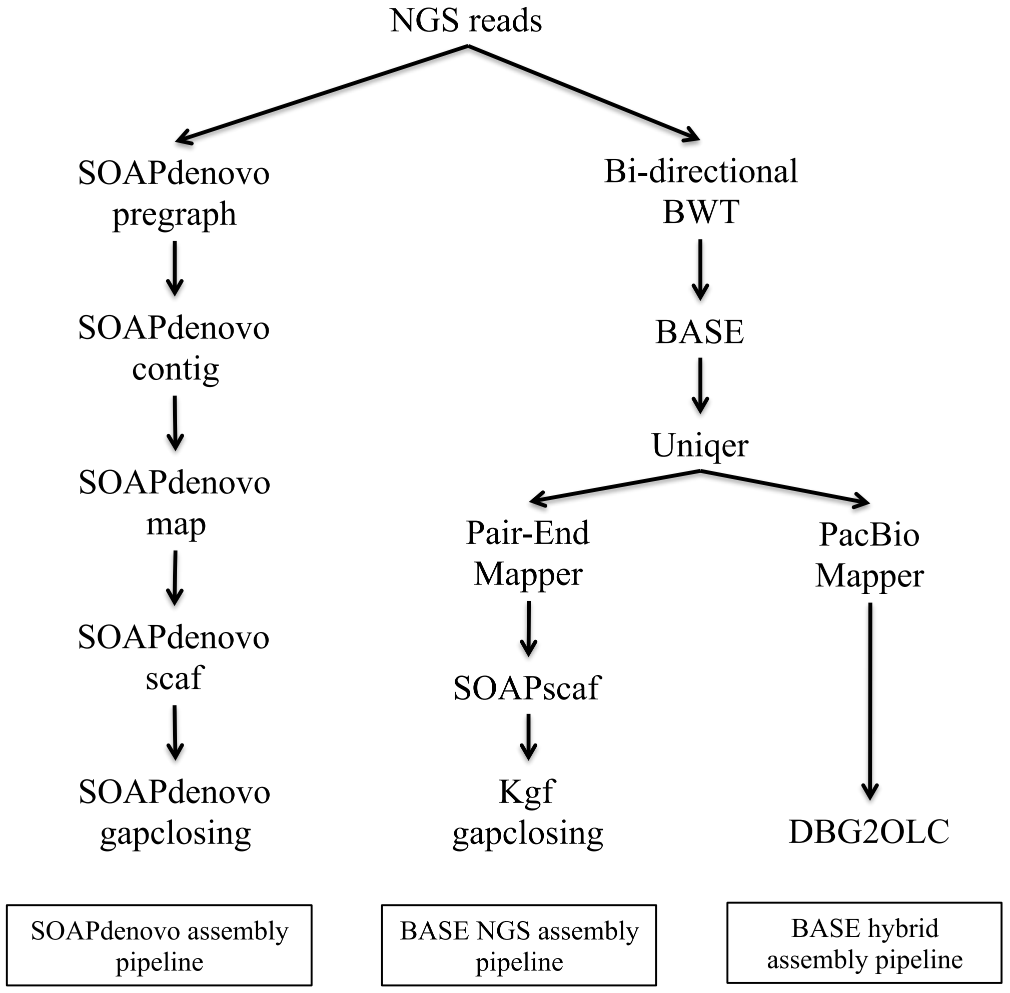

Figure 6.1: Two pipelines for genome assembly using BASE. The standard assembly pipeline of SOAPdenovo2 includes more than five steps. For BASE contigs to obtain final assembly results, we have two candidate pipelines. With paired-end short reads, we developed *Uniqer* and *Mapper* to map reads to contigs to obtain contig relations. With these contig relations and specially modified *SOAPscaf*, we can obtain scaffolds. Using *Kgf* developed by me in BGI, we can close the gaps of BASE scaffolds to obtain final assembly results. With PacBio sequencing data, we are developing a new mapper combined with *Uniqer* to obtain the relations between contigs and PacBio reads. Then we can obtain final assemblies with DBG2OLC or similar second level assemblers.

Bibliography

[1] Reference of staphylococcus aureus mw2. www.genomic.ch/edena/results2013/ReferenceSequences.

[2] wgsim. https://github.com/lh3/wgsim.

[3] Kin Fai Au, Jason G Underwood, Lawrence Lee, and Wing Hung Wong. Improving PacBio Long Read Accuracy by Short Read Alignment. *PLoS ONE*, 7(10):e46679, Oct 2012.

[4] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, Alexey V Pyshkin, Alexander V Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A Alekseyev, and Pavel A Pevzner. SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *Journal of Computational Biology*, 19(5):455–477, May 2012.

[5] Liu Binghang, Shi Yujian, Yuan Jianying, Hu Xuesong, Zhang Hao, Li Nan, Li Zhenyu, Chen Yanxiang, Mu Desheng, and Fan Wei. Estimation of genomic characteristics by analyzing k-mer frequency in de novo genome projects. *http://arxiv.org/pdf/1308.2012*, Aug 2013.

[6] Keith R Bradnam and Fass. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience*, 2(1):10, Jul 2013.

[7] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, 1994.

[8] Hongzhi Cao, Honglong Wu, Ruibang Luo, Shujia Huang, Yuhui Sun, Xin Tong, Yinlong Xie, Binghang Liu, Hailong Yang, Hancheng Zheng,

Jian Li, Bo Li, Yu Wang, Fang Yang, Peng Sun, Siyang Liu, Peng Gao, Haodong Huang, Jing Sun, Dan Chen, Guangzhu He, Weihua Huang, Zheng Huang, Yue Li, Laurent C A M Tellier, Xiao Liu, Qiang Feng, Xun Xu, Xiuqing Zhang, Lars Bolund, Anders Krogh, Karsten Kristiansen, Radoje Drmanac, Snezana Drmanac, Rasmus Nielsen, Songgang Li, Jian Wang, Huanming Yang, Yingrui Li, Gane Ka-Shu Wong, and Jun Wang. De novo assembly of a haplotype-resolved human genome. *Nature Biotechnology*, 33(6):617–622, Jun 2015.

[9] Yen-Chun Chen, Tsunglin Liu, Chun-Hui Yu, Tzen-Yuh Chiang, and Chi-Chuan Hwang. Effects of GC bias in next-generation-sequencing data on de novo genome assembly. *PLoS ONE*, 8(4):e62856, 2013.

[10] The Tomato Genome Consortium. The tomato genome sequence provides insights into fleshy fruit evolution. *Nature*, 485(7400):635–641, May 2012.

[11] Adam C English, Stephen Richards, Yi Han, Min Wang, Vanesa Vee, Jiaxin Qu, Xiang Qin, Donna M Muzny, Jeffrey G Reid, Kim C Worley, and Richard A Gibbs. Mind the Gap: Upgrading Genomes with Pacific Biosciences RS Long-Read Sequencing Technology. *PLoS ONE*, 7(11): e47768, Nov 2012.

[12] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398, 2000. doi: 10.1109/SFCS.2000.892127.

[13] Sante Gnerre, Iain MacCallum, Dariusz Przybylski, Filipe J. Ribeiro, Joshua N. Burton, Bruce J. Walker, Ted Sharpe, Giles Hall, Terrance P. Shea, Sean Sykes, Aaron M. Berlin, Daniel Aird, Maura Costello, Riza Daza, Louise Williams, Robert Nicol, Andreas Gnirke, Chad Nusbaum, Eric S. Lander, and David B. Jaffe. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, 108(4):1513–1518, 2011. doi: 10.1073/pnas.1017351108. URL http://www.pnas.org/content/108/4/1513.abstract.

[14] Giorgio Gonnella and Stefan Kurtz. Readjoiner: a fast and memory efficient string graph-based sequence assembler. *BMC Bioinformatics*, 13:82, 2012.

[15] Sara Goodwin, James Gurtowski, Scott Ethe-Sayers, Panchajanya Deshpande, Michael Schatz, and W Richard McCombie. Oxford nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. *bioRxiv*, 2015. doi: 10.1101/013490.

[16] David Hernandez, Ryan Tewhey, Jean-Baptiste Veyrieras, Laurent Farinelli, Magne sters, Patrice Franois, and Jacques Schrenzel. De novo finished 2.8 mbp staphylococcus aureus genome assembly from 100 bp short and long range paired-end reads. *Bioinformatics*, 30(1):40–49, 2014. doi: 10.1093/bioinformatics/btt590. URL http://bioinformatics.oxfordjournals.org/content/30/1/40.abstract.

[17] Wing-Kai Hon, Tak-Wah Lam, Kunihiko Sadakane, and Wing-Kin Sung. Constructing compressed suffix arrays with large alphabets, 2007.

[18] RAMANA M IDURY and Michael S. Waterman. A New Algorithm for DNA Sequence Assembly. *Journal of Computational Biology*, 2(2):291–306, Jan 1995.

[19] Szymon M Kiełbasa, Raymond Wan, Kengo Sato, Paul Horton, and Martin C Frith. Adaptive seeds tame genomic sequence comparison. *Genome Research*, 21(3):487–493, Mar 2011.

[20] Kouichi Kimura and Asako Koike. Ultrafast SNP analysis using the Burrows–Wheeler transform of short-read data. *Bioinformatics*, Jan 2015.

[21] Sergey Koren, Michael C Schatz, Brian P Walenz, Jeffrey Martin, Jason T. Howard, Ganeshkumar Ganapathy, Zhong Wang, David A Rasko, W Richard McCombie, Erich D Jarvis, and Adam M Phillippy. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature Biotechnology*, 30(7):693–700, Jul 2012.

[22] T W Lam, W K Sung, S L Tam, C K Wong, and S M Yiu. Compressed indexing and local alignment of DNA. Aug 2008.

[23] T.W. Lam, Ruiqiang Li, A. Tam, S. Wong, E. Wu, and S.M. Yiu. High throughput short read alignment via bi-directional bwt. In *Bioinformatics and Biomedicine, 2009. BIBM '09. IEEE International Conference on*, pages 31–36, Nov 2009. doi: 10.1109/BIBM.2009.42.

[24] Eric S. Lander and Michael S. Waterman. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics*, 2(3): 231–239, Apr 1988.

[25] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–359, Apr 2012.

[26] T Laver, J Harrison, P A O'Neill, A Farbos, K Paszkiewicz, and D J Studholme. Assessing the performance of the Oxford Nanopore Technologies MinION. *Biomolecular Detection and Quantification*, Feb 2015.

[27] Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics*, 31(10):1674–1676, May 2015.

[28] Heng Li. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, Jul 2012.

[29] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, Jul 2009.

[30] Heng Li, Jue Ruan, and Richard Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research*, Apr 2008.

[31] R Li, C Yu, Y Li, T W Lam, S M Yiu, K Kristiansen, and J Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, Jul 2009.

[32] Ruiqiang Li, Yingrui Li, Karsten Kristiansen, and Jun Wang. SOAP: short oligonucleotide alignment program. *Bioinformatics*, Jan 2008.

[33] Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, Shengting Li, Gao Shan, Karsten Kristiansen, Songgang Li, Huanming Yang, Jian Wang, and Jun Wang. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265–272, Feb 2010.

[34] Chi-Man Liu, Thomas Wong, Edward Wu, Ruibang Luo, Siu-Ming Yiu, Yingrui Li, Bingqiang Wang, Chang Yu, Xiaowen Chu, Kaiyong Zhao, Ruiqiang Li, and Tak-Wah Lam. SOAP3: ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics*, 28(6):878–879, Mar 2012.

[35] Chi-Man Liu, Ruibang Luo, and Tak-Wah Lam. GPU-Accelerated BWT Construction for Large Collection of Short Reads. Jan 2014.

[36] Ruibang Luo, Binghang Liu, Yinlong Xie, Zhenyu Li, Weihua Huang, Jianying Yuan, Guangzhu He, Yanxiang Chen, Qi Pan, Yunjie Liu, Jingbo Tang, Gengxiong Wu, Hao Zhang, Yujian Shi, Yong Liu, Chang Yu, Bo Wang, Yao Lu, Changlei Han, David W Cheung, Siu-Ming Yiu, Shaoliang Peng, Zhu Xiaoqian, Guangming Liu, Xiangke Liao, Yingrui Li, Huanming Yang, Jian Wang, Tak-Wah Lam, and Jun Wang. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, 1(1):18, Dec 2012.

[37] Ruibang Luo, Thomas Wong, Jianqiao Zhu, Chi-Man Liu, Xiaoqian Zhu, Edward Wu, Lap-Kei Lee, Haoxiang Lin, Wenjuan Zhu, David W Cheung, Hing-Fung Ting, Siu-Ming Yiu, Shaoliang Peng, Chang Yu, Yingrui Li, Ruiqiang Li, and Tak-Wah Lam. SOAP3-dp: fast, accurate and sensitive GPU-based short read aligner. *PLoS ONE*, 8(5):e65632, 2013.

[38] Mohammed-Amin Madoui, Stefan Engelen, Corinne Cruaud, Caroline Belser, Laurie Bertrand, Adriana Alberti, Arnaud Lemainque, Patrick Wincker, and Jean-Marc Aury. Genome assembly using Nanopore-guided long and error-free DNA reads. *BMC Genomics*, 16:327, 2015.

[39] Tanja Magoc, Stephan Pabinger, Stefan Canzar, Xinyue Liu, Qi Su, Daniela Puiu, Luke J Tallon, and Steven L Salzberg. GAGE-B: An Evaluation of Genome Assemblers for Bacterial Organisms. *Bioinformatics*, Aug 2013.

[40] Mari Miyamoto, Daisuke Motooka, Kazuyoshi Gotoh, Takamasa Imai, Kazutoshi Yoshitake, Naohisa Goto, Tetsuya Iida, Teruo Yasunaga, Toshihiro Horii, Kazuharu Arakawa, Masahiro Kasahara, and Shota Nakamura. Performance comparison of second- and third-generation sequencers using a bacterial genome with two chromosomes. *BMC Genomics*, 15:699, 2014.

[41] E W Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of computational biology : a journal of computational molecular cell biology*, 2(2):275–290, 1995.

[42] Eugene W Myers. The fragment assembly string graph. *Bioinformatics*, 21 Suppl 2:ii79–85, Sep 2005.

[43] Eugene W Myers, Granger G Sutton, Art L Delcher, Ian M Dew, Dan P Fasulo, Michael J Flanigan, Saul A Kravitz, Clark M Mobarry, Knut H J Reinert, Karin A Remington, Eric L Anson, Randall A Bolanos, Hui-Hsien Chou, Catherine M Jordan, Aaron L Halpern, Stefano Lonardi, Ellen M Beasley, Rhonda C Brandon, Lin Chen, Patrick J Dunn, Zhongwu Lai, Yong Liang, Deborah R Nusskern, Ming Zhan, Qing Zhang, Xiangqun Zheng, Gerald M Rubin, Mark D Adams, and J Craig Venter. A Whole-Genome Assembly of Drosophila. *Science*, 287(5461):2196–2204, Mar 2000.

[44] Yu Peng, Henry C M Leung, S M Yiu, and Francis Y L Chin. IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, Apr 2012.

[45] P A Pevzner, H Tang, and M S Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, Aug 2001.

[46] Mihai Pop. Genome assembly reborn: recent computational challenges. *Briefings in Bioinformatics*, 10(4):354–366, Jul 2009.

[47] Jared T Simpson and Richard Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, 26(12):i367–73, Jun 2010.

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

[49] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J.M. Jones, and nan Birol. Abyss: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009. doi: 10. 1101/gr.089532.108.

[50] R L Warren, G G Sutton, S J M Jones, and R A Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4): 500–501, Feb 2007.

[51] René L Warren, Chen Yang, Benjamin P Vandervalk, Bahar Behsaz, Albert Lagman, Steven J M Jones, and Inanc Birol. LINKS: Scalable, alignment-free scaffolding of draft genomes with long reads. *GigaScience*, 4:35, 2015.

[52] Chengxi Ye, Chris Hill, Sergey Koren, Jue Ruan, Zhanshan, Ma, James A Yorke, and Aleksey Zimin. DBG2OLC: Efficient Assembly of Large Genomes Using the Compressed Overlap Graph. Oct 2014.

[53] Daniel R Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, Aug 2008.

[54] Guofan Zhang, Xiaodong Fang, Ximing Guo, Li Li, Ruibang Luo, Fei Xu, Pengcheng Yang, Linlin Zhang, Xiaotong Wang, Haigang Qi, Zhiqiang Xiong, Huayong Que, Yinlong Xie, Peter W H Holland, Jordi Paps, Yabing Zhu, Fucun Wu, Yuanxin Chen, Jiafeng Wang, Chunfang Peng, Jie Meng, Lan Yang, Jun Liu, Bo Wen, Na Zhang, Zhiyong Huang, Qihui Zhu, Yue Feng, Andrew Mount, Dennis Hedgecock, Zhe Xu, Yunjie Liu, Tomislav Domazet-Lošo, Yishuai Du, Xiaoqing Sun, Shoudu Zhang, Binghang Liu, Peizhou Cheng, Xuanting Jiang, Juan Li, Dingding Fan, Wei Wang, Wenjing Fu, Tong Wang, Bo Wang, Jibiao Zhang, Zhiyu Peng, Yingxiang Li, Na Li, Jinpeng Wang, Maoshan Chen, Yan He, Fengji Tan, Xiaorui Song, Qiumei Zheng, Ronglian Huang, Hailong Yang, Xuedi Du, Li Chen, Mei Yang, Patrick M Gaffney, Shan Wang, Longhai Luo, Zhicai She, Yao Ming, Wen Huang, Shu Zhang, Baoyu Huang, Yong Zhang,

Tao Qu, Peixiang Ni, Guoying Miao, Junyi Wang, Qiang Wang, Christian E W Steinberg, Haiyan Wang, Ning Li, Lumin Qian, Guojie Zhang, Yingrui Li, Huanming Yang, Xiao Liu, Jian Wang, Ye Yin, and Jun Wang. The oyster genome reveals stress adaptation and complexity of shell formation. *Nature*, 490(7418):49–54, Oct 2012.

[55] Xiao Zhu, Henry C M Leung, Francis Y L Chin, Siu-Ming Yiu, Guangri Quan, Bo Liu, and Yadong Wang. PERGA: a paired-end read guided de novo assembler for extending contigs using SVM and look ahead approach. *PLoS ONE*, 9(12):e114253, 2014.