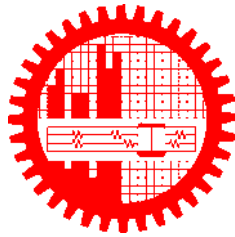


M.Sc. Engg. (CSE) Thesis

A Probabilistic Method for Filling Gaps in Genome Assemblies

Submitted by
Sumit Tarafder
0417052023

Supervised by
Dr. Atif Hasan Rahman



Submitted to
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh

in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science and Engineering

January 2022

Candidate's Declaration

I, do, hereby, certify that the work presented in this thesis, titled, "A Probabilistic Method for Filling Gaps in Genome Assemblies", is the outcome of the investigation and research carried out by me under the supervision of Dr. Atif Hasan Rahman, Assistant Professor, Department of CSE, BUET.

I also declare that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.




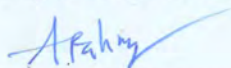
Sumit Tarafder

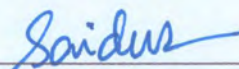
0417052023

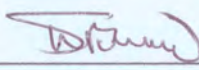
The thesis titled “A Probabilistic Method for Filling Gaps in Genome Assemblies”, submitted by Sumit Tarafder, Student ID 0417052023, Session April 2017, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents on January 25, 2022.

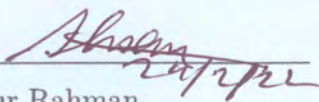
Board of Examiners

1. 

Dr. Atif Hasan Rahman Chairman
Assistant Professor (Supervisor)
Department of CSE, BUET, Dhaka
2. 

Dr. A.K.M. Ashikur Rahman Member
Professor and Head (Ex-Officio)
Department of CSE, BUET, Dhaka
3. 

Dr. Md. Saidur Rahman Member
Professor
Department of CSE, BUET, Dhaka
4. 

Dr. Md. Shamsuzzoha Bayzid Member
Associate Professor
Department of CSE, BUET, Dhaka
5. 

Dr. Ahsanur Rahman Member
Assistant Professor (External)
Electrical and Computer Engineering
North South University, Dhaka

Acknowledgement

I am thankful to Almighty for giving me the patience to fulfil this research work. I am also grateful to my mother for giving me inspiration throughout. Finally, I want to express my gratitude to the wise committee members whose invaluable guidance helped me to navigate through this work and my supervisor whose advice and continuous support this whole time at every aspect of this critical work helped me to complete my thesis.

Dhaka
January 25, 2022

Sumit Tarafder
0417052023

Contents

Candidate's Declaration	i
Board of Examiners	ii
Acknowledgement	iii
List of Figures	vi
List of Tables	vii
Abstract	viii
1 Introduction	1
1.1 Genes and genomes	1
1.2 Generations of genome sequencing technologies	2
1.3 Genome assembly and gaps	3
1.4 Contributions of this research	5
1.5 Thesis organization	6
2 Literature review	7
2.1 Long read or contig based methods	7
2.2 Short read based methods	9
2.2.1 GapCloser	10
2.2.2 Gap2Seq	10
2.2.3 GapFiller	11
2.2.4 Sealer	11
2.3 Limitations of the existing tools	13
2.4 Motivation of the work	15
3 Preliminaries	16
3.1 Sequenced reads	16
3.2 Read pairs	17
3.3 Insert size of a read pair	17

3.4	Quality of sequencing reads	19
3.5	N50 statistics of genome assembly	20
3.6	Sequencing read coverage	21
3.7	Maximum Likelihood Estimation	22
3.8	Expectation Maximization algorithm	23
4	Methods	25
4.1	Algorithm Overview	26
4.2	Aligning and parsing read pairs	26
4.3	A generative model for sequencing	27
4.4	Learning distributions	28
4.5	Gap Filling using the EM algorithm	29
4.6	Selecting the gap length	34
4.7	Finalizing the gap sequence	34
4.8	Implementation	35
4.9	Script iterations and read usage	36
4.10	Adjustment of gap filling based on heuristics	38
5	Experiments and results	43
5.1	The genome assemblies	43
5.2	The sequenced reads	44
5.3	Configurations of the tools used for comparison	44
5.4	Quality Evaluation of Filled Sequence	45
5.5	Findings on the GAGE datasets	45
5.6	Discussions	52
5.7	Time and memory usage comparison	54
5.8	Commands and configurations used to run the tools	59
6	Conclusions	60
6.1	Discussions	61
6.2	Future works	62
6.3	Availability	62
	References	63

List of Figures

1.1	Genome assembly overview	4
2.1	Overview of FGAP	7
2.2	Steps of gapFinisher algorithm	9
2.3	Schematic overview of GapFiller	12
2.4	Local assembly methods for the tools	13
2.5	Existence of multiple eulerian path in a graph	14
3.1	A graphical view of insert-size concept of a read pair.	18
3.2	Insert-size distribution for paired-end library	18
3.3	N50 calculation	20
3.4	A flowchart of EM algorithm	23
4.1	Overview of Figbird.	25
4.2	Different types of read pairs used for gap filling	27
4.3	Overview of CGAL	28
4.4	Learned insert size probabilities on ABySS2 assembly	29
4.5	Possible placement of unmapped read in different gap positions	30
4.6	Formulation of gap filling in Figbird using EM algorithm	31
4.7	Range exploration of gaps with unknown lengths	38
4.8	Consensus string at each EM iteration	40
4.9	Slow convergence due to limited overlap between reads	41
4.10	Zero alignment with both flanks	42
4.11	An illustration of discontinuous placement of reads	42
5.1	Scatter plots showing the performance of Figbird compared with four state-of-the-art gap filling tools in literature	53

List of Tables

4.1	Reads used in gap filling	36
5.1	Genomes used in evaluation	43
5.2	Read sets used in evaluation	44
5.3	Quality of the original and the gap-filled assemblies of <i>Staphylococcus aureus</i>	46
5.4	Quality of the original and the gap-filled assemblies of <i>Rhodobacter sphaeroides</i>	48
5.5	Quality of the original and the gap-filled assemblies of Human Chromosome 14	50
5.6	Gap-closing performance of the tools on draft genome assemblies of <i>Staphylococcus aureus</i>	54
5.7	Gap-closing performance of the tools on draft genome assemblies of <i>Rhodobacter sphaeroides</i>	56
5.8	Gap-closing performance of the tools on draft genome assemblies of Human Chromosome 14	57

Abstract

Advances in sequencing technologies have led to sequencing of genomes of a multitude of organisms. However, draft genomes of many of these organisms contain a large number of gaps due to repeats in genomes, low sequencing coverage and limitations in sequencing technologies. Although there exists several tools for filling gaps, many of these do not utilize all information relevant to gap filling. Here, we present a probabilistic method for filling gaps in draft genome assemblies using second generation reads based on a generative model for sequencing that takes into account information on insert sizes and sequencing errors. Our method is based on the expectation-maximization (EM) algorithm unlike the graph based methods adopted in the literature. Experiments on real biological datasets show that this novel approach can fill up large portions of gaps with small number of errors and misassemblies compared to other state of the art gap filling tools. Thus this unique probabilistic method for filling gaps is a valuable technique in terms of balance between closing gap sequence and subsequent introduction of error. The method is implemented using C++ in a software named “Filling Gaps by Iterative Read Distribution (Figbird)”, which is available at: <https://github.com/SumitTarafer/Figbird>.

Chapter 1

Introduction

1.1 Genes and genomes

In the field of genomics, the complete material that constitutes the materials of a living being called organism, i.e., all of the chromosomes of a cell is referred to as genome. Genome contains useful information for the development and functionalities of that species and this information is then passed on from one generation to the next over time. Some important experiments done between the 1920s and the 1950s showed concrete evidence that the genetic information was carried by DNA (deoxyribonucleic acid) or RNA (ribonucleic acid). There are four types of DNA nucleotides namely Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). Moreover, RNA can be represented as a string of four types of nucleotide namely Adenine (A), Cytosine (C), Guanine (G) and Uracil (U). In general, by the word “genome”, we refer to the chromosomes in the nucleus of a eukaryotic cell. But, eukaryotic cells also have organelles like mitochondria and chloroplasts that have their own DNA and referred to as the mitochondrial or chloroplast genomes to distinguish them from the nuclear genome.

Certain regions of genome termed as “genes” or coding regions were separated by the scientists from non-coding regions that were simply referred to as intergenic sequences. After many years of research, it has been established that the phenotypic traits of a species are controlled by these specific section of genome, i.e., genes [1]. The process of replication, transcription, and translation known as “Central Dogma”. With the help of this process, the genetic heredity and encoding of DNA into various gene products is maintained. These genetic products can be RNAs or proteins based upon orientation of genes. Various experiments have shown that any perturbation of these genes are responsible for the cause of various genetic disorders and diseases [2].

1.2 Generations of genome sequencing technologies

Genome sequencing of an organism, i.e., determining nucleotide base order which is correct in a DNA macromolecule using biochemical methods and sequencing machines that make up the entire genome is a prerequisite for performing experiments to study that organism's cellular functions and fundamental to understanding how different organisms relate to each other. A DNA sequencing machine outputs files which holds DNA sequences [3]. Scientists have researched for a long time to sequence the complete genomes of many organisms and till now, it has been completed for thousands of species from all domains of life and many more are to be completed soon. As sequencing technology has been going through a rapid development, there are three generations of technology that have evolved over past two decades. We will discuss about their characteristics below in brief.

- First generation of sequencing: The first generation of technologies were developed in 1977 by Sanger [4] and Maxam [5]. Sanger Sequencing is a method that terminates chain or the sequencing by synthesis method. It can create fragments or reads of length 400-900 approximately. Despite Sanger sequencing was the ultimate prevalent technique of sequencing based on the efficiency which is high and radioactivity which is low [6], it is not applied currently due to its high cost and enormous time to complete the run. For example, it took almost fifteen years for the human genome to be sequenced using the Sanger sequencing which costed approximately 100 million US dollars and a cooperation of lot of laboratories around the world.
- Second generation of sequencing: The introduction of a new flurry of sequencing technologies broke the restriction of the first generation. These new technologies are known as “Next Generation Sequencing (NGS) Technologies” or Second Generation of Sequencing (SGS). Some of the important characteristics of NGS include parallel generation of millions of short reads using amplification libraries with lower processing time and cost than Sanger sequencing. There are three main SGS approaches : 454/Roche in 2005 [7], Solexa/Illumina [8] in 2006 and in 2007 the SOLiD/ABi. After using the 454 Genome Sequencer, it took time which is close to only two months and in terms of cost it took close to one of hundred than that [9] of Sanger based methods to complete human genome sequencing project. Illumina technology has been able to produce short reads in paired-end (PE) manner in which they sequence both ends of each DNA strand. Illumina Hiseq technology has managed to generate reads up to length of 150 base pairs for both single end (SE) and paired end (PE) reads [10].

- Third generation of sequencing: Due to the requirement of PCR amplification step in SGS technologies, the execution time and price also gets higher. In addition to that, most genomes are extremely repetitive and SGS technologies can't solve these repeat regions due to relative short length of reads. The amount of repetitive sequences in human genome(*H. sapiens*) is above 60% [11]. For these reasons, “third generation sequencing” (TGS) which is a new generation of sequencing has been developed which does not require the amplification libraries. The technology of TGS consists of approaches like Single Molecule Sequencing Technology (SMRT) [12]. This approach has been the basis for sequencers such as Pacific Biosciences and Oxford Nanopore sequencing. These can generate longer read sequences of lengths up to 10 kilobase pairs (Kbp) and beyond. Although long reads are significantly larger and thus can positively affect genomic analysis, it comes with the cost of containing more error than Illumina reads. These error mostly consist of insertion errors and also sometimes deletions errors and a random distribution of errors along the long read [13] can be observed. Both PacBio and Nanopore technologies have far higher error rate (88 – 94% accuracy for Nanopore and 85 – 87% for PacBio) as opposed to 99.9% accuracy for second generation Illumina HiSeq 4000 [14].

1.3 Genome assembly and gaps

As next generation sequencing technologies have become very advanced, more and more sequencing data are getting available day by day. However, genomes can be billions of nucleotide base pairs long and there is no existing technology that can sequence the entire genome in one single run. So instead of constructing the complete genome, different generation of sequencing technologies discussed above generate millions of smaller fragments called reads. The process of reconstructing the original genome from these read sequences is known as **genome assembly**. A graphical overview of the steps of genome assembly is presented in Figure 1.1.

Genome assembly pipeline constitutes of the following steps. The first step is to stitch the read sequences generated from sequencers into a larger and contiguous sequences called **contigs**. Then using paired-end (PE) or mate-pair (MP) reads, these contigs are oriented and organized into a larger linear ordering of sequences named as **scaffolds**. A large number of de novo assemblers, i.e., novel algorithms and pipelines such as ABySS [15], Velvet [16], Allpaths-LG [17], SPAdes [18], ISEA [19], EPGA [20], EPGA2 [21] etc. have been developed for this assembly purpose. Nonetheless, the process of finding a complete genome remains a demanding task as the assemblers are

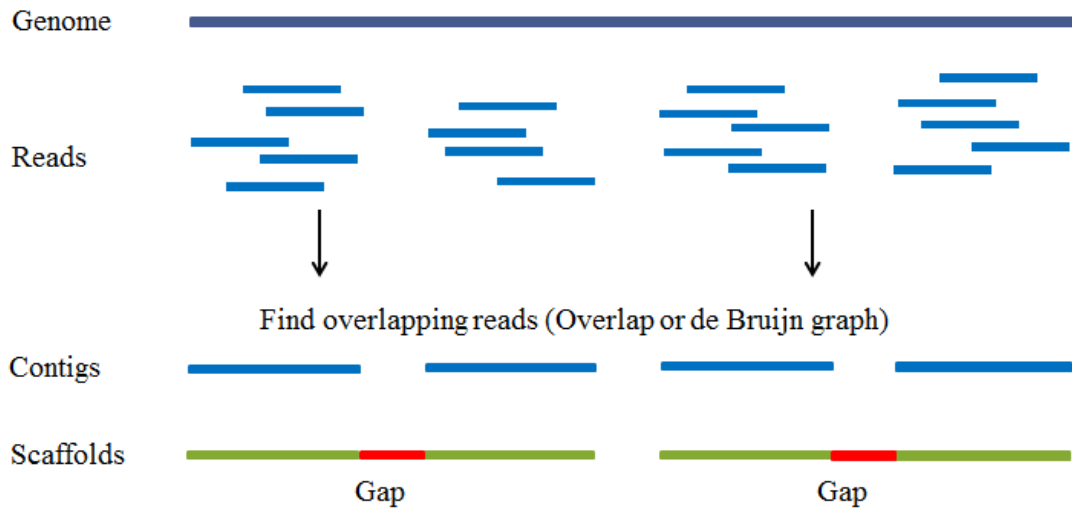


Figure 1.1: Genome assembly overview

unable to assemble parts of genome due to the presence of repetitive elements, regions with low read coverage, heterozygous alleles, sequencing errors etc. This creates a fragmented version of the genome rather than a complete assembly (Miller et al., 2010; Treangen and Salzberg, 2012). Although the presence of approximate insert distance information of read pairs may help to solve these issues [22, 23], this method will be unable to solve the deeper problems of low-coverage and repetitive elements.

So, the draft assemblies constructed with these de novo assemblers still contain thousands of intervening gaps within the assembled scaffolds as shown in Figure 1.1. This is due to the fact that contig extension is prevented due to various factors such as repeats, errors in sequencing, coverage issues and alleles [24, 25]. Filling these gaps with minimal introduction of error is one of the crucial steps in genome assembly pipeline as an error free complete genome can lead to better downstream analysis such as genotyping structural variants [26], a complete annotation of genes [27] etc. Structural variants refer to a larger variation between reference genome and donor genome in the form of insertions, deletions, duplication and inversions and identifying the type of variant is referred to genotyping. Also, it is not only the intergenic DNA region of genome that contain crucial genomic features but also the regions that are not coded and repeats are responsible for the evolution of many organisms [28, 29]. Chromosomal rearrangements, i.e., the duplication, deletion, inversion and translocation of genomic material, in most cases occur due to the implication by repetitive DNA [30, 31]. However, presence of such repetitive DNA obstructs genome assembly process hugely and in many sequencing projects, the physical co-localization of genetic loci is found in the gaps [32],

thus these rearrangements aren't fully observed. So, an error free, contiguous, gap less and close to complete genome assembly will go a long way to further improve the identification of these structural rearrangements, effector genes and co-regulated genes [33] and accurate statistical analysis [34].

1.4 Contributions of this research

Maintaining the balance between closing a large number of gaps and introducing minimal error across a variety of draft assemblies is a tedious and computationally challenging task to achieve. In this thesis work, we have developed a likelihood based probabilistic method for filling gaps in the scaffolds using second generation Illumina sequencing [35] based short reads. The reason for using reads from SGS technology instead of TGS technology is that although long reads can significantly help during the metagenomic analyses and solve repeat related problems, specially in genome assembly [36], its use in the gap filling process is still questionable due to the high error rate present in the reads as mentioned in Section 1.2. Error in sequencing data can only be managed on some levels if high coverage is available to guarantee the quality which can be an expensive process. Gap filling is the last stage of genome assembly pipeline where modification of the scaffolds are carried out in terms of insertion of new sequences. So any erroneous sequence inserted in this stage will carry over to subsequent analysis tasks and may lead to incorrect observations. For this reason, we have chosen comparatively correct second generation read sequences for gap filling purpose in this research work.

In this work, we formulate gap filling as a parameter estimation problem and develop a probabilistic method for filling gaps in scaffolds. The method is based on a generative model for sequencing proposed in CGAL [37] and subsequently used to develop a scaffolding tool SWALO [38]. The model incorporates information such as distribution of insert size of read pairs, sequencing errors, etc. and can be used to compute likelihood of an assembly with respect to a set of read pairs. We use this model to estimate the length of the gap and to find a sequence for each gap that maximizes probability of the reads mapping to that gap region. However, in this case the insert sizes of the read pairs that have only one end mapped is unknown. So, we use an iterative approach similar to the expectation-maximization (EM) algorithm [39]. We have also presented a comprehensive comparison between our proposed method and other state-of-the-art tools based on six different evaluation metrics such as misassembly, erroneous length and number of gaps remaining using QUASt [40], which is absent on most other contemporary works related to gap filling.

Our method is implemented in C++ in a tool called Figbird. An extensive comparison with a number of different gap-filling tools on bacterial genomes and human chromosome 14 data sets from GAGE [41] is also presented in this work. On *Staphylococcus aureus* dataset, Figbird shows great performance in reducing total missassembly rate on average by 4% and erroneous length by 22% over 8 different de novo assemblies of this genome which is the best among all other state-of-the-art tools. In case of Human chromosome 14 dataset, Figbird is second to GapCloser in terms of the amount of gap filled. But the additional 3% gap filled by GapCloser comes at the expense of 47% and 7% more misassemblies and erroneous length respectively compared to Figbird. Overall, our probabilistic method performs well consistently on six different metrics over variety of real draft assemblies, and is able to reduce amount of gaps substantially while keeping misassemblies and errors low. Although the method is computationally intensive, it is linear in the number of reads as well as the number and the lengths of gaps, and is thus scalable and applicable to large genomes. Achieving a significant amount of gap closure along with erroneous length reduction coupled with the introduction of extremely low missassembly than other gap filling tools, Figbird shows promising robustness of our proposed probabilistic method.

1.5 Thesis organization

We have organized our thesis work in the following way: Firstly, we have introduced a literature review on the existing gap filling methods used for genome assembly pipelines in Chapter 2. Then, the preliminary concepts of the mathematical ideas and algorithms used to develop Figbird have been described in Chapter 3. Then, in Chapter 4, we have described the entire method along with the parameters used to develop Figbird. In Chapter 5, we have compared the performance of Figbird with other gap filling methods on three datasets to solidify the concreteness of our novel approach of gap filling on real life organisms. Finally, in Chapter 6 we have concluded the thesis by discussing potential ideas for the improvement of our proposed algorithm in future.

Chapter 2

Literature review

We will discuss about the research works conducted to solve the problem of existence of gaps in genome assembly pipeline in this particular chapter. Our literature review will be summarized in the following way. We will review an array of tools or methods that use either the long read or short read sequencing data currently available in sequencing databases in Sections 2.1 and 2.2 respectively. Then we will present the limitations of these existing methods in Section 2.3 and finally in Section 2.4, we will discuss about the motivations that these limitations lead us to do this thesis work.

2.1 Long read or contig based methods

These methods are based on contigs or long reads that are aligned to the scaffolds and gap regions to find the anchoring reads. The advantages of these approaches are that the contigs or long reads can span across large gap regions and thus the resultant gap filled assembly will have better contiguity and less fragmentation. A tool named FGAP [42] utilizes nucleotide BLAST [43] as its component to align multiple contigs to draft assembly and then identifies the best sequences that overlap with gaps. A graphical overview of FGAP is given below in Figure 2.1.

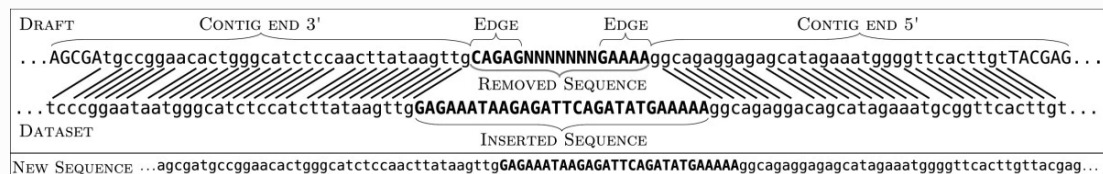


Figure 2.1: Overview of a gap filled by FGAP (This figure was taken from [42] published by BioMed Central under Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0>)).

The alignment of contigs to proposed scaffolds will be done separately. To restrict the alignments and choose the best alignment from all returned alignments, parameters like minimum score, identity of minimum and e-value are used in FGAP. Another tool named GMcloser [44] concentrates on minimization of misassemblies on the gap closing sites. A set of contigs which is pre-assembled or a long read set which is error-corrected using PacBio reads is used to close the gaps. The method incorporates splitting scaffolds and then alignment of contigs with other preassembled contigs using Nucmer [45] or BLASTn. Using likelihood-based classifications derived from the stats that represent alignment, GMcloser finds proper closure of gap regions using those reads or contigs. Also, GMcloser uses a filtering function of Coval [46] to filter mismatches between reads and haploid or heterozygous diploid genomes. Pairwise alignment has been performed with the YASS aligner [47] in case of gaps that are closed partially based on the alignment stats between Watson and Crick strand. There is a further tool that close the gaps using long reads is PBJelly [48] which uses Pacific Biosciences [49] long reads. PBJelly performs read alignment to the gap region to construct a sequence that is based on majority voting approach of those aligned reads. Then it tries to bridge gaps and if there are any erroneous sequence present in the closed sequences, it tries to make correction to those as well.

Another long read based tool in literature is called gapfinisher [50] which uses an automated gap filling pipeline using FGAP [42] algorithm and claims to achieve efficient and reliable gap filling. A detailed overview of gapFinisher is given in Figure 2.2. The tool gapFinisher currently only works on SSPACE-LongRead [51] output and requires FGAP [42], an [52] interpreter in Perl for SSPACE-LR. The basic workflow of gapFinisher involves steps such as indexing the FASTA file of draft genome and the file in FASTA format of reads that are long and generating a superscaffolds list. Then FGAP will be run for each superscaffold to fill gaps in them.

GapBlaster [53], a graphical application which uses an alignment method that uses contigs generated in the genome assembly process and performs an alignment of reads to the draft scaffold set using BLAST or Mummer [45]. Then, the tool identifies alignments which encompasses the gaps in the draft scaffolds and they are chosen by the user for gap filling purpose via graphical interface. G4ALL [54], another tool which is similar to GapBlaster is implemented using JAVA programming language. It uses BLAST to align contigs to scaffolds and use the alignment results as a choice via a graphical interface so that users can choose which scaffolds should be used to fill the gaps. Additionally, there are some assembly-merging tools such as PCAP.REP [55], CISA [56], Minimus2 [57]

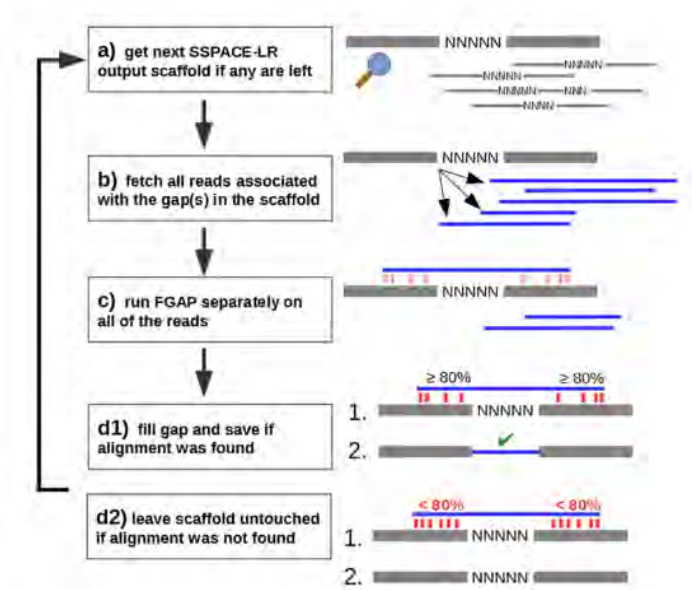


Figure 2.2: A detailed overview of gapFinisher (This figure was taken from [50] published by PLoS ONE under Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0>).)

which are sometimes used as a gap closure method by merging a set of assembled sequences. However, these methods don't fully serve the purpose of gap filling and instead used as an extension procedure of multiple assemblies.

2.2 Short read based methods

Just like long read based tools, there are also quite a few short read based tools available in literature. We will discuss quite a few of them here and mainly focus on the four tools that are the most state-of-the-art in the context of gap filling. The methods of the tools for filling gaps can be described in two categories [58] as follows. In category 1, there are methods based on either paired end reads or reads that are mate-pairs which may span a gap and a local assembly is performed to fill those gaps. IMAGE [59] is a tool that uses process based on iterative approach. It also utilizes Velvet [16] for the assembly purpose of reads whose flanking regions align with one ends of the read pairs. Only smaller sized genomes can be used to fill gaps using IMAGE [60]. FinIS [61] utilizes un-assembled reads to construct an overlap graph with reads and with the help of a program that depends on mixed-integer quadratic method, it identifies an optimal path that can be used to fill the gap. It only utilizes reads that went un-used during contig assembly and scaffolding process. Next, CloG [62], which only makes use of the VELVET assembler for the generation process of a hybrid assembly. A set of of reads

of different lengths were used in this purpose from NGS reads to contigs. Another tool named GAPPadder [63] is built for diminishing existing gaps in draft scaffolds which is similar to other tools that we discussed in the sense that a local assembly method is conducted on reads assembled to fill the gap regions. The main difference between GAPPadder and other methods is it utilizes more read information compared to other methods and considers a range of different insert sizes of read pairs. This tool is also different in the sense that the local assembly method is a combination of two steps since a two-stage local assembly is done in this case. Firstly assembly of contigs in the gap region is performed from reads and then another local assembly which is of better quality is created by merging those previous contigs. Now we will discuss about the four state-of-the-art tools in literature:

2.2.1 GapCloser

GapCloser is a stand-alone module in SOAPdenovo [64] package, which utilizes the paired reads to fill the gaps. It is designed as one of the last steps in SOAPdenovo assembler's pipeline as a finishing process but it can also be used as an independent tool. SOAPdenovo [65] is the first published version of this method and it has successfully assembled many large eukaryotic genomes. GapCloser constructs a **De Bruijn graph** on the set of available reads to perform the local assembly. A De Bruijn graph is a directed graph that represents overlap among read sequences. In this graph, the nodes are considered as $k-1$ mers extracted from the reads and an edge between two nodes represent the k -mer. It is an iterative process and considers all the available reads at that stage to build the graph. Although it works well for smaller genomes, it is highly memory inefficient for larger genomes [66] and thus not scalable. Furthermore, it only considers read pairs with insert size less than 2000 as it can not make the high insert-size read pairs to work.

2.2.2 Gap2Seq

Gap2Seq [67] is a recent computational approach for gap filling where the problem is formulated as an exact path length (EPL) problem, implemented in pseudo-polynomial time with some optimization. As studied in [68], the gap which exists between two ends of a pair of reads is considered to be reconstructive if there can be built a local assembly graph from the reads where a unique path that is shortest in the graph between the flanking nodes can be found. Such a formulation of problem where path finding is performed is called exact path length (EPL) problem [69]. In Gap2Seq, they have formulated the gap-filling problem as EPL problem and used De-Bruijn graph structure

on entire read set to perform the gap filling process.

The major drawback of such an approach of Gap2Seq is that the EPL problem is NP-hard and it uses all the available reads in dataset without considering any insert-size information and constructs a huge DBG graph on whole readset which is a huge computational approach. That's why although it works well on most small prokaryotic genomes, this approach does not scale to large genomes well and thus unable to fill large gaps as it fails to solve the NP-hard problem formulation.

2.2.3 GapFiller

GapFiller [60], is a software designed by the SSPACE-tools [23, 51] researcher and is one of the successful ones in the literature. It uses only paired-end read information and only partially aligned reads to the gaps. Gaps are not filled if the reads partially aligned do not span the entire gap and kept as it is. It assembles read pairs with one end aligned to the scaffold and the other end partially aligned to the gap region. The similarity between the gap lengths present in the scaffolds and the lengths of the newly inserted sequences predicted for a particular gap is maintained in this tool. Finally the reads are assembled using k-mer-based method exploiting a majority voting ratio strategy to fill the gaps. The limitations of GapFiller are that it does not use a lot of sequence information due to the only use of partial reads and also time inefficient for larger genomes [66] with no intermediate outputs. An overview of the GapFiller pipeline is given below in Figure 2.3.

2.2.4 Sealer

Sealer [66] is designed to close gaps in scaffolds by navigating De Bruijn graphs represented by space-efficient bloom filter data structures. Sealer boasts to be scalable for larger genomes ranging from 100 Mb to giga base pair sized genomes. Sealer is highly memory efficient and uses Konnector [70] as its core to close gaps that are present in scaffolds but does not consider the insert size of paired reads. Sealer performs three sequential functions as follows: First, gap regions and the flanking sequences are extracted and stored in separate files. Then, these flanking sequence pairs along with reads are provided to Konnector as the inputs. A set of k-mer values is given to the Konnector to find the optimal sequence. Finally, a correct sequence is used to fill the gaps. The positive side of Sealer is that the size information of gaps is ignored by Sealer as the length of the gaps provided in the scaffold file may not be always correct.

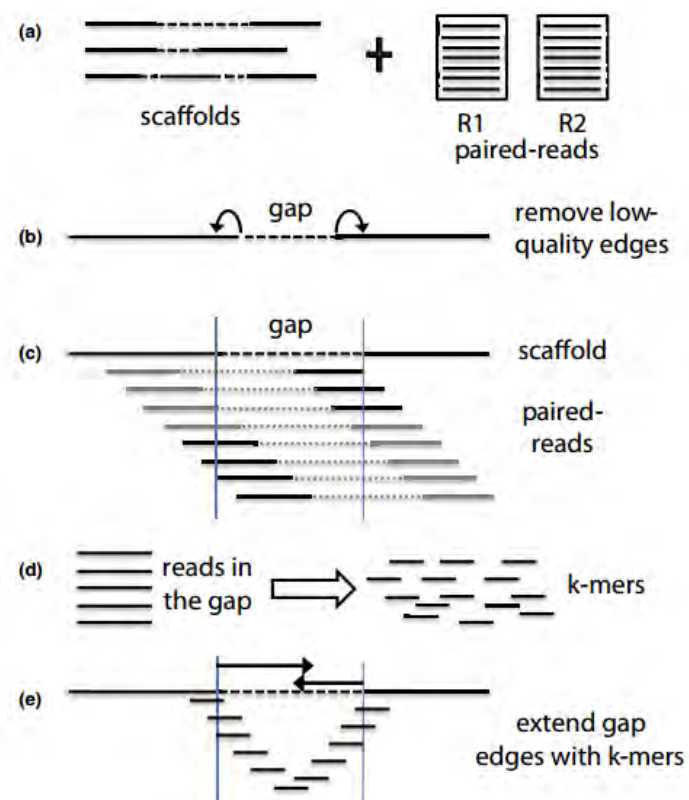


Figure 2.3: Work flow overview of the method of GapFiller . (The figure is taken from [60] published by BioMed Central under Creative Commons Attribution License (<http://creativecommons.org/licenses/by/2.0>)).

Almost all of the tools discussed above utilize a common pipeline to fill gaps: (i) Utilization of read alignment to gap regions in the scaffold; (ii) Collection of different types of read pairs to fill the gap; (iii) Formulation of a method to perform the assembly of the collected read pairs based on De Bruijn graphs, read overlap graphs or any other complex k-mer based strategies as depicted in Figure 2.4 and (iv) the repeated iteration of processes (i–iii).

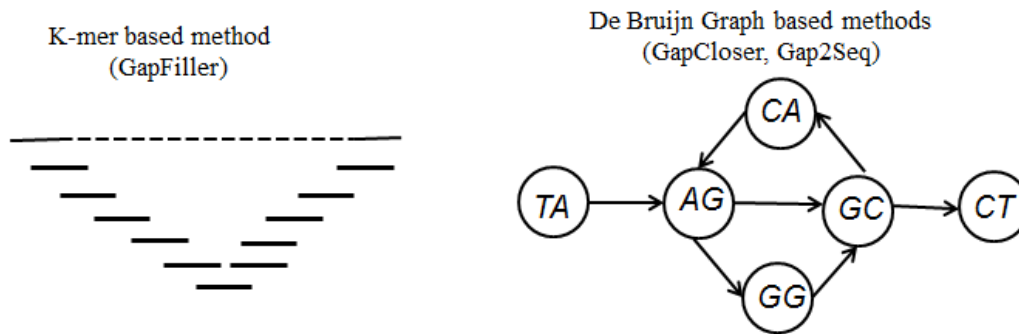


Figure 2.4: General overview of local assembly methods for the tools

2.3 Limitations of the existing tools

Almost all of the methods mentioned above use graph based formulation of the problem, most widely De Bruijn graph or read-overlap graph and then try to find an Eulerian path through the graph that will correspond to the unique gap sequence relevant to that gap. But there are some inherent complications in these graph based methods. Firstly, the reason that there are gaps that exist in genomes are identical to the reasons of the presence and creation of fragmentation of contigs. To formulate the problem, gap filling is often described as finding an optimal path from a starting node which is one flank and then to an ending node in a graph that is another flank. So there are two primary steps: (i) Graph construction is a primary step. However, as the graph becomes dense because of the huge dataset, there becomes more paths and branches in that graph. A reduction of branching may lead to a loss of nodes that represent low coverage areas. Furthermore, the reason of the presence of gaps in draft genome assembly is because the assemblers failed to solve these regions using graph based methods in the first place. So the search for a novel formulation is more appropriate in this context instead of graph based methods, (ii) While choosing an optimal path between the two cornering nodes, the branching problem, i.e., the existence of multiple such Eulerian paths present in the graph due to the presence of repetitive regions or sequencing errors [16] and only one of those paths corresponds to the true genomic sequence of the gap. Such a scenario is

illustrated in the Figure 2.5:

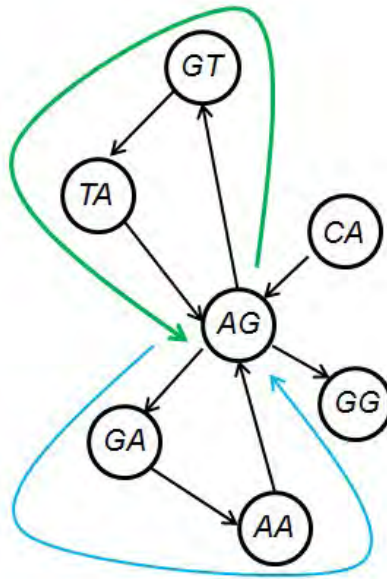


Figure 2.5: Existence of multiple eulerian path in a graph

As you can see from Figure 2.5, there are two possible Eulerian paths through the graph, one starting from node CA and then following the path of AG, GT, TA, GA etc. and another path goes around the graph constructing a sequence of CAGAAGTAGG. Finding a unique and thus correct path from these multiple choices can get complicated either due to the presence of repeats or because of the memory constraint issue due to the dense structure of the graph built from a large set of k-mers. Although GapCloser has mentioned in their paper that in case of assemblies that are very large, methods that are based on DBG will take a huge amount of memory for the step that constructs the graph with nodes. To have a better approach in SOAPdenovo2, they implemented a sparse De Bruijn graph method [71]. In this methods, reads are split into k-mers and they are stored in the memory as a group rather than uniquely. Despite this fact, GapCloser still suffers from high memory consumption issue and thus the problem still persists. Also, most of the tools, in some proposition, ignore distance information from the other end of the pair, i.e., insert size, that may help disambiguate among multiple-mapped sequences and solve repeat related issues. To resolve this problem we can use the coverage information to check the copy number of each node. But when there is a triple (or more) repeat present in the assembled sequence, even coverage will not give a unique solution. In these cases, we can try to retrieve longer reads or use the paired-end read information and thus insert-size information present in the reads to span the repeat. So, searching for the true gap sequence that can solve the above stated unsolved issues in literature is still a challenging area to be explored in genome assembly.

2.4 Motivation of the work

As we thoroughly discussed on Section 2.3 about the limitations of the existing tools in literature, it is extremely motivating to look out for a method that can give a reasonable solution to the problems. Also, keeping in mind about the fact that, a lot of methods in the literature have only managed to utilize one specific assembly for gap filling purpose or testing purpose and ignored the rest of them or it wasn't available at that time. Also, a lot of gap filling tools ignore gaps of small length for example 1 – 20 bp [50] and some have ignored larger gaps due to huge computational complexity [67]. That's why we have tried to devise a method that is completely novel in formulation and doesn't depend on graph based techniques while taking insert-size and other information in consideration. Also, we wanted to design a tool that can maintain the balance between closing a large number of gaps ranging from smallest to the largest and introducing minimal error or missassembly across a variety of draft assemblies of a genome which is a tedious and computationally challenging task to achieve.

Chapter 3

Preliminaries

In this chapter, we will introduce the preliminary concepts that will be necessary to explain the methodology and evaluation of Figbird in later sections. As Figbird is a gap filling tool which uses read sequences to fill those gaps, so most of concepts discussed below is related sequencing reads and also genome assembly and assessment metrics.

3.1 Sequenced reads

In case of DNA sequencing, as genomes are billions of base pair long, so there is no sequencing technology yet that can sequence the entire genome sequence at once. Instead, what the sequencing machines generate smaller fragments of sequences called reads. These reads can be single or in pair and they are stored in corresponding files in FASTA/FASTQ format accordingly. The characters present in the reads are usually A, C, G and T. But sometimes there can be another character present in the sequence which is 'N'. This character indicates that in that position there is a possibility of any of the four characters specified above and the particular sequencer was unsure due to the low coverage or any other reason during sequencing process.

Also, the length of the sequenced reads has great impact on biological experiments [72] such as gap filling, solving repeat related regions in genome etc. as we have discussed before. Therefore, sequencing reads length is crucial in case of bioinformatics analyses [73]. Figbird is designed and then further developed considering second generation short read libraries such as frag or mate-pair libraries which are sometimes known as jumping libraries [74]. These libraries contain reads in pair, that is they are sequenced from both strands of DNA. They are called paired end or mate pair reads based on the distance between the pairs. We will refer to both these type of reads as read pairs.

3.2 Read pairs

Most commercially available high-throughput sequencing technologies achieve the sequencing by synthesis process and synthesize in a based on the directions and mechanisms of DNA polymerase works in every living organism's cells. In conventional paired-end sequencing, the Watson strand (5'- end) is sequenced first by attaching the adapter and then sequencing using the adapter for reverse end starts once this is over. Therefore the paired-end read pairs stored FASTQ files conceptually point towards each other on opposite strands.

During the alignment of the read to the genome, one read will align to the forward strand whereas the other end to the reverse strand or reverse complement of the genome. Thus they are inwardly pointed towards each other and known as an “FR” read or forward/reverse or “innie” direction. But there are also some different technologies that incorporates large insert sizes known as jumping libraries [75], where reads are aligned in an “RF” position or reverse/forward or “outie” direction. These are derived due to the presence of circularized DNA fragments. So the two read pairs are pointing away from one another.

So, with all these, there can be a total three orientation possibilities of reads and they are categorized as: forward reverse, reverse-reverse/forward-forward (TANDEM) and reverse forward (RF). The read pairs that are oriented in a FR orientation is called paired-end reads. They have relatively small inserts (300 – 500) bp. Mate-pair fragments are usually in a RF conformation containing larger inserts (3 kb). The tandem reads are a bit different and they are the results of reshuffling during library preparation.

3.3 Insert size of a read pair

Insert size is a terminology in high-throughput sequencing genome sequencing which can be easily confused with other terms and that can lead to misunderstanding of the entire topic. We work with the reads from a sequencing run which is sequenced and is actually a piece of DNA with two adapter sequences attached, one at each end. Figure 3.1 is an outline of what a fragment, with an insert of DNA to be sequenced looks like in case of paired-end reads.

The insert size is the size of the piece of DNA of interest, without the adapters. It is the distance between the two starting points of that particular read-pair. This varies depending on library prep protocol and usually generates a distribution of inserts

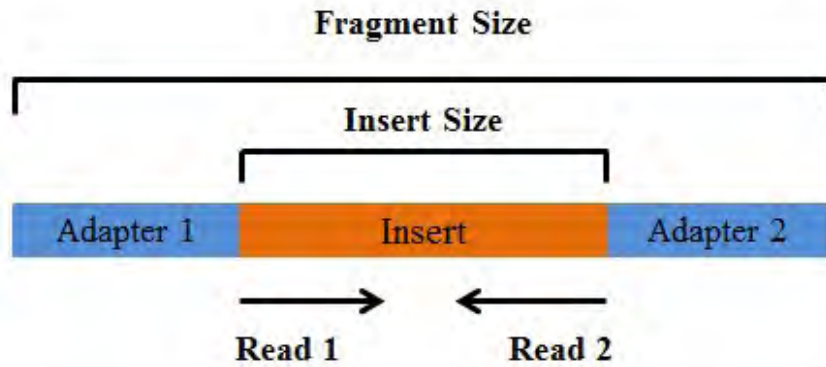


Figure 3.1: A graphical view of insert-size concept of a read pair.

of different lengths. For example, the paired-end read library (frag) collected from GAGE [41] and used in this experiment has an approximate insert size range of 150–200 bp. Identification of the range of possible insert sizes is important in our work as it limits the possible placement positions of unmapped read pairs. Although we have calculated the mean and standard deviation of this distribution in our method, we have demonstrated this distribution here graphically in figure 3.2. To do that, we mapped the read library to the reference genome using bowtie2 [76] and evaluated the distribution of insert size for all the properly mapped pairs using Qualimap [77] which is shown below in figure 3.2.

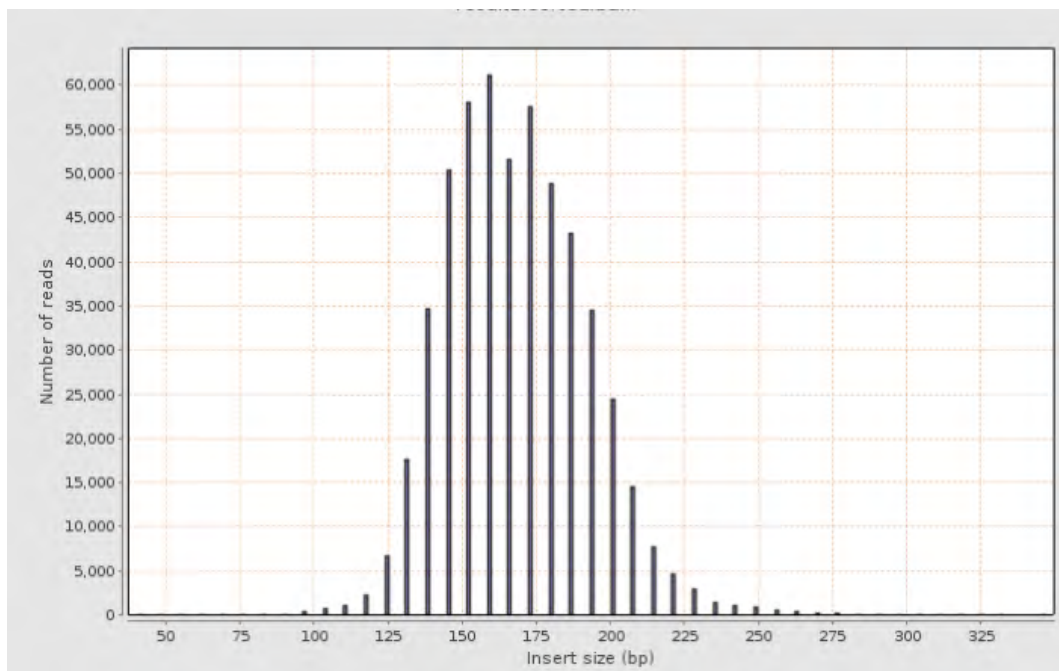


Figure 3.2: Insert-size distribution for paired-end library evaluated using Qualimap [77]

Although, the insert size of a read pair is not exactly known, with a large sample, the distribution of insert size approximately follows a normal distribution whose mean and standard deviation can be calculated [37, 38]. The advantage of getting two reads per fragment is that it can give additional information and increase the mapping accuracy the two reads to a reference genome.

3.4 Quality of sequencing reads

Each and every read sequence generated by next generation sequencing technologies can contain errors and these error probabilities can be predicted for each base indicated by quality (Q) scores or Phred scores. These Q scores are extremely helpful in case of read filtering and reducing errors in different experiments in downstream genome analysis tasks such as gap filling, marker gene sequencing experiments [78] etc. The Q score is defined as an integer which is usually in between a range of 2 to 40 indicating the base call misjudgment probability. So, $Q = 3$ indicates an error probability is 50%, while $Q=10$ indicates to an error probability of 10%. If P is the error probability, then:

$$P = 10^{-Q/10} \quad (3.1)$$

These Q scores are sometimes encoded as ASCII characters in the corresponding fastq files. There are different rules for this conversion process but ASCII BASE 33 is most prevalent in case of Illumina data.

The overall quality of a read can also be calculated using a term called expected number of errors in the read. To do that, at first we have to calculate the error probabilities P according to the Q scores in a given read from a FASTA file. The underlying assumption is that a very large number of reads have to be present that contains errors with those probabilities. Although the Q scores fail to indicate the exact amount of errors present in a given read, the Phred quality scores can indicate the average number of errors present in a huge read set where some of the reads have error and some don't. This is called the expected number of errors which is real rather than integer because it's an average and can be less than one [79]. One can calculate the expected number of errors using the formulae and their proofs from [80].

3.5 N50 statistics of genome assembly

N50 metric is represented in a large number of genome assembly papers for evaluation purpose. It is a measure that indicates or was designed to indicate the contiguity of a genome assembly but in reality it gives us information about contig length distribution. So to calculate N50, all the contigs in the assembly has to be sorted in the order of their sequence lengths as shown in Figure 3.3. Now while traversing through these sorted contigs, the contig length that constitutes half or more than half of the total assembly length is N50 of the genome assembly as shown below.

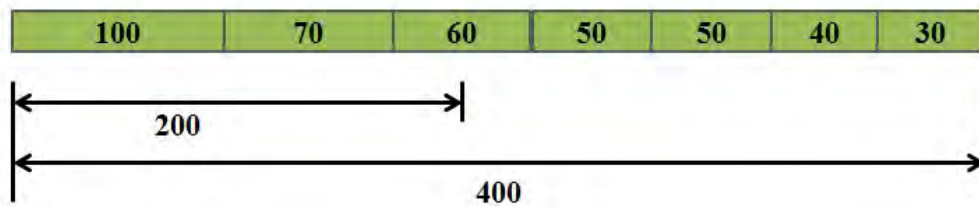


Figure 3.3: Calculation of N50 for a set of seven contigs.

So, N50 contig size refers to a length that indicates that considering a genome, half of this length is covered by this size or a length that is greater than this N50 value [81]. The reason of using N50 most often than N90 or anything else and why it describes a length instead of a number is because of it's first introduction in the original human genome paper [82] the way as it is.

3.5.1 Problem with N50

The problem with N50 is in it's highly misleading behavior in case of usage in genome assemblies. In an ideal case, there would only be a few contigs in the assembly and it will lead to a high N50 value. On the other hand, a low quality assembly would consist of a large number of fragmented contigs which will lead to a low N50. However, there can be many different scenarios where N50 value can be manoeuvred to give a false indication of high quality for a poor assembly as described in this blog [83] which is known as N50 filtering problem. Also, contigs can be incorrectly merged together to create a larger contiguous genome with higher N50 value and thus N50 can easily be inflated by manipulating the contigs which is known as N50 misassembly problem. A wrongly assembled genome with lots of chimeras is not better than one with multiple contigs and low N50, so this is important to address.

3.5.2 NGA50: A solution to both the problems of N50

The solution to N50 filtering problem is possible if the genome length of the organism is known approximately. In such a case, new metric called NG50 (G refers to genome) is derived. The difference between this metric and N50 is that instead of considering half of the total assembly length, NG50 consider half of the total genome length. That means, which ever way we filter the contigs of an assembly, it will not affect the NG50 length as reference genome length is now considered instead of independent assembly lengths. To solve the misassembly problem of N50, i.e., the erroneous joining of small contigs, multiple solutions have been proposed [40, 41, 84, 85]. There is a massive similarity among the methods as all of them require a high quality reference genome. A new metric NAx [40] is calculated, where A stands for Aligned. To compute NA50 of an assembly, sequence alignment has to be performed for the set of contigs to the genome. If there are any misassemblies, then the contig is split into aligned blocks and an independent alignment is performed. These two solutions are merged together to solve both the problems and a new metric called NGA50 derived. It is computed similarly to the NA50 but considering half of the reference genome length an not assembly length. For our experimental evaluation we will use the NGA50 metric as this is computed by QUAST [40].

3.6 Sequencing read coverage

Next-generation sequencing (NGS) coverage is defined as an average of the count of the total number of reads that align to a known reference base position. The sequencing coverage level indicates the level of confidence in case of variant genotyping at particular base positions. Also an important fact is to remember that the reads will be sampled in a random manner from multiple copies of a genome. So, the distribution of reads will be uneven over an entire genome sequence. Thus a fluctuation of coverage will be observed at different base positions from the average coverage.

By coverage, we mean depth of coverage, i.e., the total number of times a particular sequenced base covered the genome. The term coverage and depth is used interchangeably in literature. It is also known as mapping depth which indicates the confidence level of the coverage of a genome using sequenced fragments (short reads). The Lander/Waterman [86] equation for calculating the depth of coverage is given below:

$$C = L * N/G$$

Here, C is coverage, G is the length of genome, L is the length of reads and N represents the count of total reads. So, if we take the following example in count:

$$C = (100bp) * (189106)/(3109bp) = 6.3$$

3.7 Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) is a technique to estimate the parameters of a probability distribution. The parameter θ can be a vector such as $\theta = (\theta_1, \theta_2, \theta_3, \dots, \theta_k)$. The purpose of this process is to find a model that best fits the data under an assumed statistical model [87] so that the likelihood is maximized. An example of the use of MLE is in finding the mean and variance of a sample population where the population is assumed to follow a normal distribution. If $x_1, x_2, x_3, \dots, x_n$ is a set of observations that are generated from a probability distribution f which depends on some parameter θ , then the primary purpose of MLE is to maximize the likelihood function:

$$L = f(x_1, x_2, x_3, \dots, x_n | \theta) = \prod_{i=1}^n f(x_i | \theta)$$

The advantage of MLE is that it is consistent in the sense that a sequence of MLE will converge in probability to the actual parameter being estimated. To determine the value of θ in the space of values Θ that maximizes this likelihood, we use maximum likelihood estimation:

$$\hat{\theta} = \arg \max_{\theta \in \Theta} \hat{L}$$

During likelihood calculation, the product of the probabilities may get so small that they can underflow and thus become useless. Also, finding the maximum requires the calculation of differentiation of the function which sometimes gets too complicated. So, to make things simple as well as to preserve small probability estimates, the function is simplified by taking natural logarithm. Since natural log is a monotonically increasing function, the consistency of identifying correct parameter is maintained in both techniques.

$$\ln L = \sum_{i=1}^n \ln f(x_i | \theta)$$

3.8 Expectation Maximization algorithm

The Expectation-Maximization (EM) algorithm finds maximum-likelihood for cases where both model parameters as well as observations are unknown or incomplete. In practical scenarios, it is often analytically impossible to find the derivative of the log-likelihood function due to missing parameters or values. This is the scenario where instead of the exact approach, EM algorithm can give numerical solutions to predict model parameters. It is iterative in nature to identify the maximum likelihood. Expectation-maximization (EM) algorithm, originally published in [39] is presented in Figure 3.4 in most simplified manner possible. EM approach can be applied to solve those classes of problems which have some hidden observations as well as unknown model parameters.

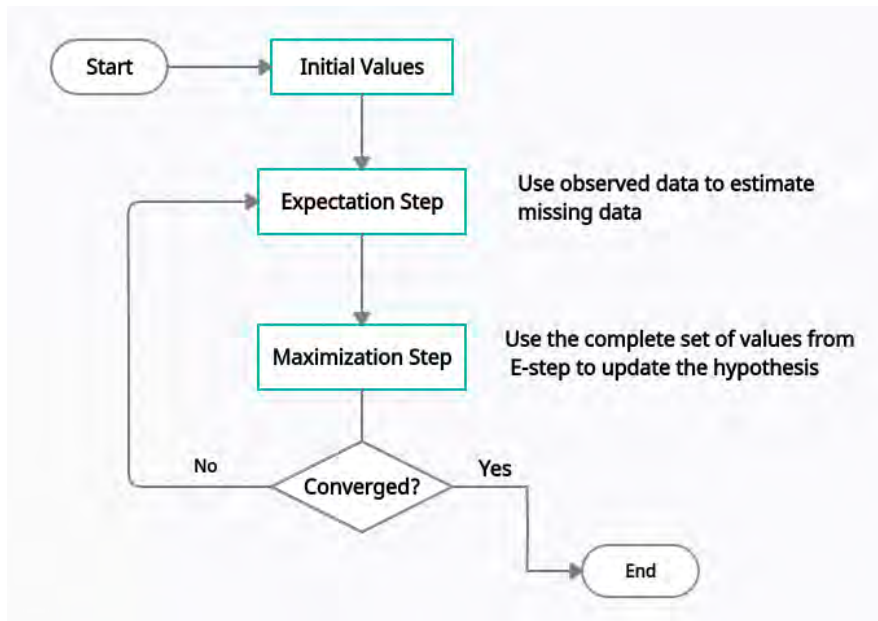


Figure 3.4: A flowchart of EM algorithm

The EM algorithm proceeds by picking an initial set of model parameters to estimate the hidden observations with the assumption that the data comes from a specific model. This is called the E-step, for the expected distribution. Then, using the newly estimated values of hidden observations, the parameters or initial hypothesis gets updated. This step is called the M-step and here, the probability distribution calculated from the E-step is updated after accumulating the calculated values in M-step to update the initial set of assumed parameters. These two steps keep iterating until the resulting values both converge to a fixed point or the allocated time ends. The only negative side of EM algorithm is that it has extremely slow convergence. It is feasible when the data dimensionality is low or missing data constitutes a small portion of the entire set. The

higher the dimensions, the slower the E-step will be. It is also possible that sometimes, in case of gap filling, the E-step may run extremely slowly if there is not enough overlap between suffix and prefix of assembled sequence data.

Chapter 4

Methods

This chapter in details describes the methodology and the formulation applied in Figbird to perform the gap filling task. A high level overview of our gap-filling method has been illustrated in the block diagram in Fig 4.1. For general understanding and overall organization, we will describe our methodology in the following two steps.

- Pre-processing step: Identifying relevant read pairs and learning distributions.
- Gap filling step: Probabilistic approach for filling gaps using the parsed reads.

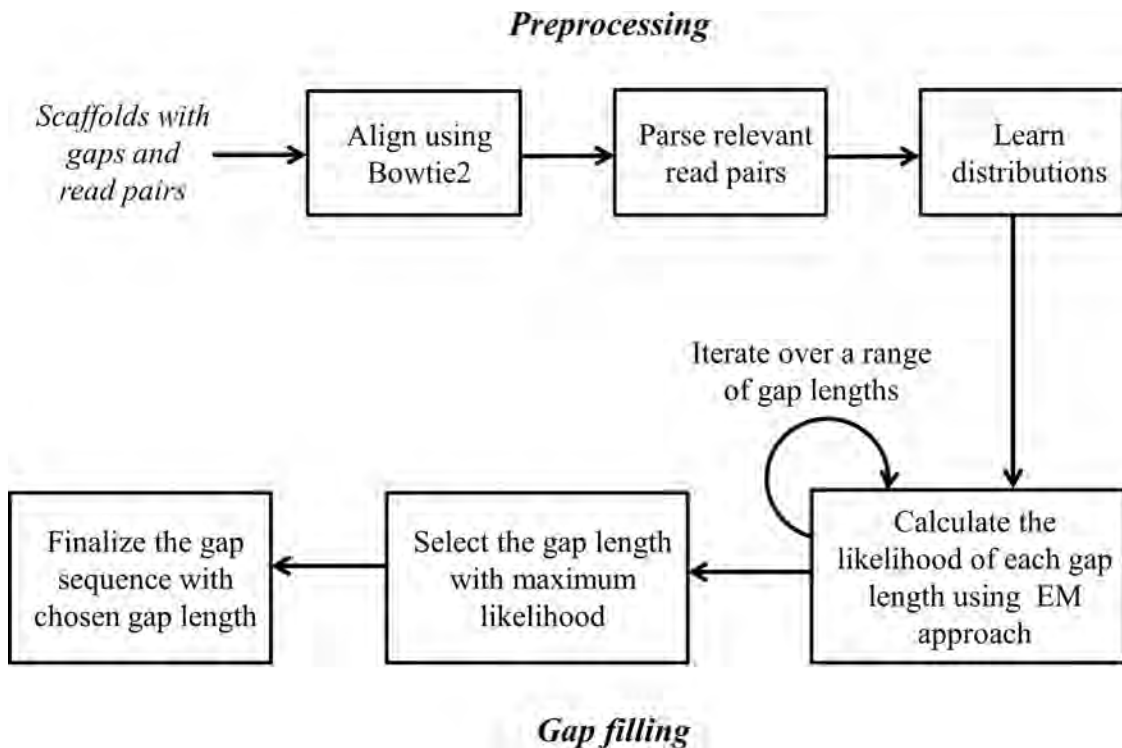


Figure 4.1: Overview of Figbird.

4.1 Algorithm Overview

At first paired end and mate pair reads (we will refer to both these type of reads as read pairs) is aligned to the scaffold set using Bowtie2 [88] aligner. Then the SAM [89] format output of bowtie2 is parsed to collect all possible relevant read pairs necessary for our method. The details of the read pairs as well as the read parsing criteria is described in Section 4.2. In the next step, the insert size distribution and parameters for our error model are learned using uniquely and fully mapped read pairs.

In the gap filling phase, read pairs with one end unmapped or partially mapped are locally assembled using a maximum likelihood approach calculated using a model described in Section 4.5. As we do not know exactly where the unmapped end of the read pair should be placed within the gap, we use the expectation-maximization (EM) algorithm [39] to iteratively find the placement of the read using the learned insert size distribution and the current estimate of the nucleotide in the gap sequence, and re-estimate the probabilities of the nucleotide using the current placements. The process is iterated over a range of gap estimates with different lengths and the one with the maximum likelihood value is chosen to fill the gap region. Once the gap length and the corresponding sequence is estimated, the distribution of probability of fully mapped reads learnt from the previous step are used to decide whether the reads should be considered to fill that particular gap based on a cut-off value and reads with probability below the cut-off are discarded. Finally, a consensus is calculated based on the probabilistic placements of the chosen reads in gaps which is regarded as the final predicted gap sequence.

4.2 Aligning and parsing read pairs

In this phase, read pairs are aligned to the draft scaffolds using Bowtie2 and the resulting output file in SAM format [89] is parsed to separate read pairs, and to assign relevant reads to specific gaps. The different types of reads considered are shown on Figure 4.2 and described below:

- (i) **Fully aligned:** Fully aligned read pairs are the ones whose both ends map to the draft genome concordantly, i.e., in proper orientation and within the specified insert size. These type of reads are used to learn insert size distributions and sequencing error rates which are used to calculate the likelihood of a gap estimate.
- (ii) **One end unmapped:** These are the read pairs which have one end mapped to the draft scaffold but the other end remains unmapped. To find which gap the unmapped

end is possibly associated with, the mapped end of the read pair is checked to determine whether it falls within $1.15 \times M$ base pairs on either side of the gap region and these pairs are then separately saved per gap region. Here, M denotes the insert size mean of the corresponding library given input to Figbird and the threshold is set based on the observation that most of the insert sizes fall within one hundred and fifteen percent of the input mean.



Figure 4.2: Different types of read pairs used for gap filling

If the condition is satisfied for more than one gap due to the congested and fragmented nature of the gaps, then it is ambiguous as to which gap the unmapped read actually belongs to. In that case, we chose the gap for which the distance is closest in terms of insert size mean of the library.

(iii) **One end partially mapped:** By running Bowtie2 in the mode that supports local alignment, we also collect the read pairs whose one end is partially mapped to one of the ends of a gap region. These type of reads should lead to an error free construction of gap sequences as the placement of the reads are known in this case.

Due to the potential presence of huge number of gaps as well as the large lengths of those gaps, a lot of read pairs often have none of their ends mapped to the draft genome. Since our method is going to run several times on different read sets, there is a chance that the reads that are unmapped in initial iterations may get mapped in subsequent iterations. So in addition to using the three types of reads stated above, we may also be able to utilize both end unmapped reads as well for gap filling.

4.3 A generative model for sequencing

Our method is dependent on a generative model for sequencing presented in [37]. The generative model gives the likelihood that a read pair is generated from a certain region of the genome based on learnt parameters. We have used this model and parameter estimation approach as a basis for computing the likelihood of the placement of other end of the read that partially or fully falls in a gap region. A brief description of the

model will be presented in this section. If, N paired end reads, $R = \{r_1, r_2, \dots, r_N\}$ are generated from a genome, then the read pair $r_i = (r_{i1}, r_{i2})$ constitute a segment symbolized by this read pair based on three distributions. The insert size l_i of the fragment is chosen from an insert distribution F . The starting point for the Watson end of the fragment s_i is chosen based on a distribution, S and an error model, E was learned to count for sequencing mismatches. The model is illustrated in following figure.

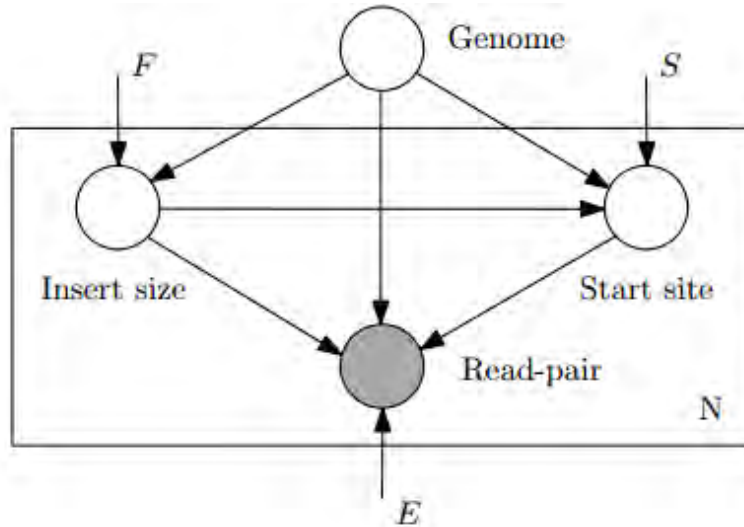


Figure 4.3: Overview of CGAL. (This figure was taken from CGAL [37] published by BioMed Central under Creative Commons Attribution License (<http://creativecommons.org/licenses/by/2.0>)).

4.4 Learning distributions

In order to fill gaps using Figbird, we need to learn insert size distribution and sequencing error characteristics which are required to calculate the probability that a set of read pairs is generated from a certain region of the genome using the generative model presented in CGAL [37]. Since sequencing characteristics differ among different libraries, we have chosen to learn distributions from individual libraries used in the experiments. To do that, we map the read pairs to the scaffolds using Bowtie2 and learn empirical distributions using fully mapped read pairs that map uniquely. As the number of uniquely mapped reads to learn the insert size distribution from may not be very high for some datasets, it is smoothed using a window size of 12 as well as truncated as proposed in [38]. We calculate the insert size distribution for each dataset as shown in examples in Figure 4.4.

We also compute the mean μ and left-sided and right-sided standard deviations, σ_l and σ_r respectively and truncate the distribution at $\mu - 2.5\sigma_l$ and $\mu - 2.5\sigma_r$, which we use

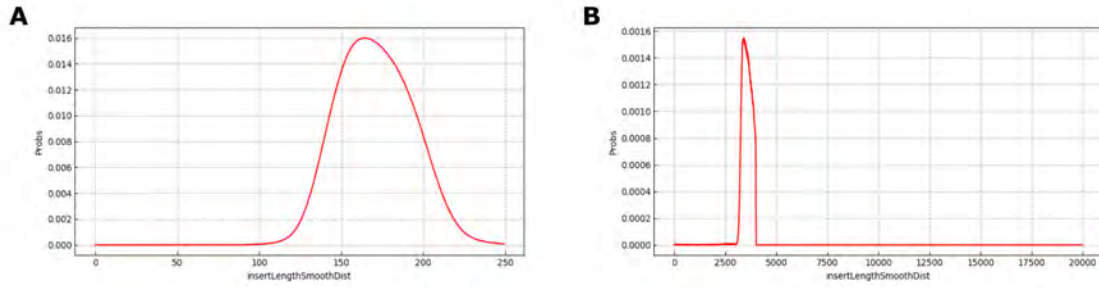


Figure 4.4: Insert size probabilities learned for (A) frag library and (B) jump library on ABySS2 assembly of *Staphylococcus aureus* bacterial genome.

as our minimum and maximum threshold for the insert size range of an unmapped read. The error model used in our computation, described in detail in [37], includes substitutions or mismatches, insertions-deletions and takes into account differences in error rates across positions in reads.

4.5 Gap Filling using the EM algorithm

In the next phase, we fill the gaps using read pairs with one end mapped and the other end unmapped or partially mapped with a likelihood based approach. Given a gap sequence \mathcal{G} of length g and a set of unmapped reads $\mathcal{R} = \{r_1, r_2, \dots, r_N\}$, the log likelihood of \mathcal{G} is given by

$$l(\mathcal{G}; \mathcal{R}) = \log \prod_{i=1}^N p(r_i | \mathcal{G})$$

where $p(r_i | \mathcal{G})$ is the probability that r_i is generated from \mathcal{G} . We are interested in the gap sequence that maximizes this likelihood and to calculate this, we use the generative model described in CGAL. However, since one end of the read pair is mapped to a fixed position, we modify the model and define probability of a read as follows:

$$p(r_i | \mathcal{G}) = p_F(L) p_E(r_i | \mathcal{G})$$

where L is the insert size of the read pair, and p_F and p_E are insert size and error distributions respectively.

In this thesis, we formulate gap filling as a parameter estimation problem. Given a gap of length g , we introduce the parameter to be estimated as

$$\theta_{j,c} \text{ for } 1 \leq j \leq g \text{ and } c \in \{A, C, G, T\}$$

where $\theta_{j,c}$ denotes the probability of nucleotide c at index j of gap sequence. The gap filling problem then converts into a parameter estimation problem where the goal is to find the estimates of $\theta_{j,c}$ s that maximize the likelihood $l(\mathcal{G}; \mathcal{R})$.

However, to estimate these parameters, we need to know the insert sizes exactly. Although the insert sizes are known for one end partially mapped reads, we do not know these for one end unmapped reads as sequencing experiments do not provide the exact distance between the two ends. It is worth noting that although we don't exactly know the insert size values, the read pairs follow an approximately normal distribution which can be learnt as discussed earlier and this observation can reduce the possible positions of the unmapped end within a range of minimum and maximum value of insert size for that read pair as indicated in Figure 4.5.

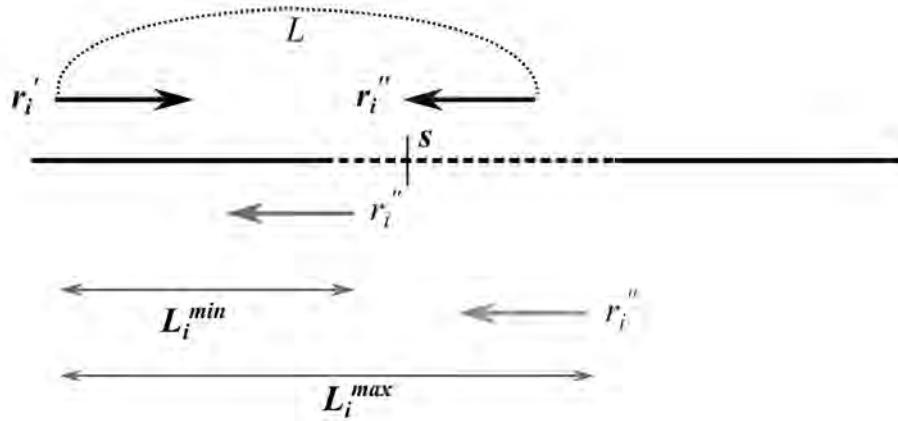


Figure 4.5: Possible placement of the unmapped end r_i'' of read r_i in different gap positions.

Now, if the insert size of a read pair was exactly known, we could have placed the read at the correct position within the gap and use the sequence to adjust the probabilities of the nucleotides occurring at those gap positions as shown in Figure 4.6A. On the other hand, if the gap sequence was known to us beforehand, then we could have aligned the read to that known sequence and obtain the likely placement of the unmapped end as shown in Figure 4.6B. But neither the exact insert sizes nor the sequence of the gap region are known beforehand and to solve one of these problems we need the solution of the other.

To solve this set of interlocking problems, we will use the expectation maximization (EM) algorithm. EM algorithm can be applied to solve those class of problems which have some hidden observations as well as unknown model parameters. It proceeds by picking an initial set of model parameters to estimate the hidden observations with

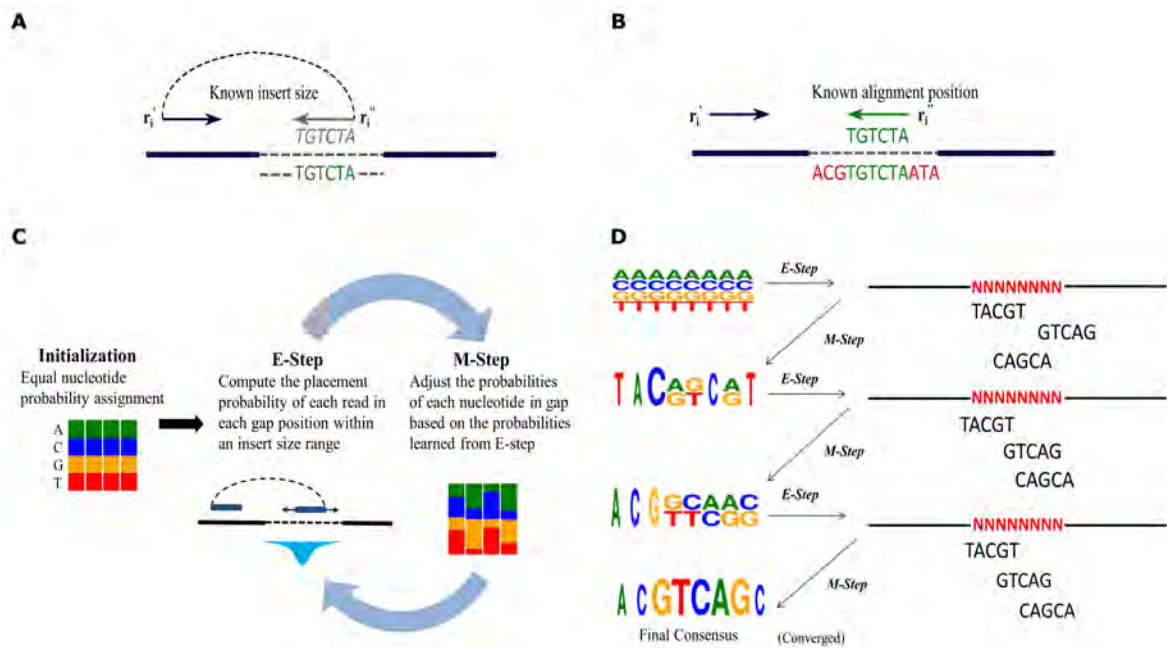


Figure 4.6: Formulation of gap filling using the EM algorithm. (A) If the insert sizes are known exactly, the reads can be placed within the gap in the correct positions and the gap sequence can be inferred. (B) If the sequence of nucleotides in the gap is known, reads can be aligned to the sequence, and their placement and insert sizes can be estimated. (C) Figbird solves gap filling using the EM algorithm. It starts by initializing each nucleotide with equal probability. In the E-step, current probabilities of nucleotides and insert size distribution is used to calculate placement probabilities of each read within the gap. Then in the M-step, the placement probabilities and read sequences are used to update the probabilities of nucleotides. These two steps are iterated until convergence. (D) A simplified simulation of the EM algorithm for gap filling.

the assumption that the data comes from a specific model. This is called the E-step. Then, using the newly estimated values of hidden observations, the parameters or initial hypothesis gets updated. This step is called the M-step. These two steps are iterated until the resulting values converge to a fixed point or the allocated time ends. In our method, the hidden observations are insert sizes of the read pairs and the parameters are the probabilities of each nucleotide occurring at each gap position.

EM formulation

A schematic diagram of our expectation maximization approach for gap filling is shown in Figure 4.6C. We start with an initial hypothesis that each of the four bases $\{A, C, G, T\}$ are equally probable at each gap position j and initialize the estimation parameter $\theta_{j,c}$ with 0.25 for all gap positions and for all four possibilities of nucleotide c . Then the E step and M step is applied iteratively as follows:

E-step: In the E-step, we place the unmapped end of the read $r_i = c_1c_2 \dots c_l$ in all possible gap positions based on the set of allowable insert sizes $\mathcal{L}_i = [L_i^{min}, L_i^{max}]$, where L_i^{min} and L_i^{max} are the minimum and maximum threshold value of insert size for read r_i respectively. Then the probability that the read is generated from that position is calculated using the current estimated probabilities θ . This posterior probability of read r_i having a particular insert size $L \in \mathcal{L}_i$, i.e., that it starts at s , is given by:

$$f_i(L) \propto p_F(L) \left[\prod_{j=s, k=1}^{j=s+l-1, k=l} \left(\theta_{j,c_k} \times (1 - p_{er}(k)) + \tau_{j,c_k} \times p_{er}(k) \right) \right]$$

where l is the length of the read and s is the start position of the read within the gap corresponding to the insert size L , $p_{er}(k)$ is the error probability at read position k , and τ_{j,c_k} is the probability of getting nucleotide c at gap index j and read position k due to a sequencing error which can be calculated using the following equation:

$$\tau_{j,c_k} = \sum_{k=1}^5 \sum_{i=1}^5 \theta_{j,i} \times \gamma_{i,c_k}$$

where, γ is a 5×5 square matrix containing the probability of substitution of each of the four nucleotide characters and character 'N' into others.

M-step: In this step, we accumulate the probabilities calculated for all the reads in E-step in an intermediate matrix α with the same dimensions as θ . If s is the starting position of read $r_i \in \mathcal{R}$ in the gap with respect to an insert size $L \in \mathcal{L}_i$, α is updated according to the following equation:

$$\alpha_{j,c} = \epsilon + \sum_{i=1}^N \sum_{L \in \mathcal{L}_i} f_i(L) \mathbb{1}_i(j - s + 1, c)$$

where ϵ is a small value added to ensure the probability of characters do not become zero and $\mathbb{1}_i(k, c)$ is a variable indicating whether the k -th character of r_i equals c , i.e.,

$$\mathbb{1}_i(k, c) = \begin{cases} 1 & \text{if } r_{i,k} = c \\ 0 & \text{otherwise} \end{cases}$$

Finally, the update of our estimation parameter θ using the intermediate values in α is done as following:

$$\theta_{j,c} = \frac{\alpha_{j,c}}{\sum_{i=1}^4 \alpha_{j,c_i}} \quad \forall j \quad 1 \leq j \leq g$$

Here, c_i denotes each of the four possible nucleotides in position j . Based on this updated hypothesis, we will continue our E and M steps until there is a convergence, i.e., the placement positions of reads don't change anymore and thus the hypothesis reaches a fixed set of values.

A simplified simulation of our EM algorithm is presented in Figure 4.6D. Initially, the probabilities of each nucleotide is equal across all positions as indicated in the top left of the figure. Based on this set of parameters, we calculate the probabilities of each read aligning at each gap position in the E-step and accumulate those probabilities at M-step to determine an intermediate consensus with updated nucleotide probabilities. The relative height and frequency of nucleotides at different positions in the figure denote the corresponding information content and relative probabilities at those positions. The information content I for a particular position i at y -axis is denoted by:

$$I = \log_2(s) - (H_i + e_n) \quad (4.1)$$

Here, $s = 4$ is used for 4 different types of nucleotide. H_i is the Shannon Entropy [90] at position i defined as:

$$H_i = \sum_{c=1}^4 f_{c,i} * \log_2 f_{c,i} \quad (4.2)$$

where, $f_{c,i}$ is the relative frequency of base c at position i . The e_n is small-sample correction defined as:

$$e_n = \frac{1}{\ln 2} * \frac{s-1}{2n} \quad (4.3)$$

Here, value of s is 4 and n is the total number of sequences present in the alignment. These two steps then iterate two more times and finally at the end of third iteration we manage to obtain the true placements of the unmapped reads in gap based on the continuously updated set of parameters and the steps converge to a fixed value. The final consensus is constructed based on the final placement of reads using a majority voting approach and the gap is filled with the predicted final consensus sequence.

4.6 Selecting the gap length

Once the EM converges for a particular gap length g , we compute the likelihood of the estimated parameters as follows:

$$l(\boldsymbol{\theta}_{g,c}; \mathcal{R}) = \log \prod_{i=1}^N p(r_i | \boldsymbol{\theta}_{g,c})$$

where $p(r_i | \boldsymbol{\theta}_{g,c})$ is the maximum over all placement probabilities of r_i , i.e.,

$$p(r_i | \boldsymbol{\theta}_{g,c}) = \max_{L \in \mathcal{L}_i} f_i(L) \quad (4.4)$$

However, since the gap length is often not known exactly, we iterate over a range of gap lengths and select the gap length that maximizes the above likelihood, i.e., $\max_g l(\boldsymbol{\theta}_{g,c}; \mathcal{R})$. For each gap with length g in the scaffold file, we compute the likelihood for the range $0.5 \times g$ to $2.5 \times g$ and the gap estimate with the maximum likelihood is selected as the final gap estimate. We observe that for most of the gaps, the actual gap length is within the specified range. However, for long gaps this becomes computationally expensive. So, we use a heuristic for deciding the range which is described in details in Section 4.10. It is to be noted that gap length can be negative if the sequences preceding and succeeding the gap region overlaps.

4.7 Finalizing the gap sequence

At this stage of our pipeline, we will finalize our gap sequence based on an error model and learnt cut-off value. The significance of this step is that every unmapped read parsed during the preprocessing phase does not truly belong to that gap region due to the fact that one or both end of a read pair might be very error prone and the aligner sometimes

fail to map these reads to the scaffolds and thus they remain unmapped. So it is essential not to consider such reads in gap filling process. To prune these unnecessary reads out, we perform the following steps. Firstly, we generate an intermediate consensus sequence \mathcal{C} based on the output of final M-step of our method for the gap length corresponding the highest likelihood. Then each unmapped read r_i is slid across the allowable consensus positions based on the insert size, and the error probability of r_i placed on consensus position s is calculated using the following equation:

$$p(r_i|\mathcal{C}) = \max_{L \in \mathcal{L}} e(L)$$

If the consensus character at position j , i.e., \mathcal{C}_k matches with r_{ij} which is the j^{th} character of read r_i , then

$$e(L) = \prod_{j=s, k=1}^{j=s+l-1, k=l} \mathbb{1}_i(k, \mathcal{C}_j) (1 - p_{er}(k) - p_{in}(k) - p_{del}(k)) \\ + (1 - \mathbb{1}_i(k, \mathcal{C}_j)) (p_{er}(k) \times \gamma(\mathcal{C}_j, r_{i,k}))$$

where p_{er} , p_{in} , p_{del} are error position, insertion and deletion distributions respectively across all read position calculated using the parsed CIGAR information from SAM alignment output. If the error probability $p(r_i|\mathcal{C})$ of r_i is less than a cut off error probability, only then we will consider the read for gap filling, otherwise it is discarded from consensus consideration. The cut off probability value is pre-calculated using the distribution of probabilities of uniquely and fully mapped reads. It is the value above which the probabilities of 80% of such reads lie.

Finally, we place each read that are above the cut-off threshold value at their most likely position according to Equation 4.4 and construct a final consensus sequence based on a maximum voting approach. This is our final predicted sequence \mathcal{G} for that particular gap.

4.8 Implementation

The method is implemented using C++ and is available for download freely at: <https://github.com/SumitTaraferder/Figbird>. In order to reduce the runtime and ensure accuracy, we apply the following heuristics in the implementation:

- Removal of duplicate reads: We have removed duplicate reads while parsing reads with one end unmapped which helps significantly in the calculation of likelihood

due to the presence of erroneously parsed reads as discussed in before. We have observed that such reads tend to pile up in certain parts of gaps due to the sequence similarity and including them negatively affects both the accuracy and running time of the method.

- Iterative implementation: To make the most out of the various types of reads available, we run Figbird with one end partially mapped and one end unmapped reads of different libraries in an iterative manner. A detailed description of the iterations are given in Section 4.9
- Reduction of read library: To make the method scalable, we have removed read pairs with both ends mapped from our dataset after the first iteration as learning the distributions and error parameters once suffices for the rest of the iterations.
- Other heuristics: To improve accuracy in the predicted consensus sequence, a number of heuristics are applied at different stages of gap filling which are described in details in Section 4.10.

4.9 Script iterations and read usage

To run our method as a software Figbird, we have prepared a driver script to manage the entire program. In this section, we will discuss about some parameters, details of iterations and types of reads and some house checking used in iteration for the entire process.

Firstly the read pairs and their usage is described in table below:

Read type	Read library
Partial	Frag
Unmapped	Shortjump

Table 4.1: Reads used in gap filling

As, frag reads have higher length compared to jump reads, we used only this read as partial reads as there is a higher chance of partially falling in gaps for these reads. Also as longer reads contain more N, only the reads with N count less than or equal to 3 is considered. All the reads are checked so that don't contain any character other than A,C,G,T and N. We have also considered gaps with length 1 unlike other methods. Sometimes filling a 1 length gap can increase or reduce erroneous length and NGA50 by a huge margin because of the nature of scaffold assembly and a false filling of that gap can give totally opposite result which makes every change very tricky and time

consuming. That's why these type of reads were ignored in other methods, but we have tried to fill every possible gap.

Our method is implemented in an iterative fashion where multiple iterations are performed using different sets of reads with different types. The reason for an iterative approach is that a lot of gaps that have been filled in previous iterations can act as mapping regions for the reads that are currently unmapped. So a lot of reads that were unmapped in first few iterations will now have a higher chance of getting mapped and thus we can extract more out of the higher insert size jump libraries. For each iteration, the alignment using bowtie2 is done twice. This is for the fact, that once it is run to find the partially aligned reads and second time it is run to find the one end unmapped reads. As frag type library has much higher number of read pairs (36 million read pairs as opposed to 14 million in case of jump library for *Staphylococcus aureus* genome, we reduced frag read set after first iteration by removing those pairs whose both end has been mapped (Cigar string in SAM file will be 101M), so that subsequent iterations have less read to map.

In iteration 1, we use frag reads as partial reads (soft clipped) and initialize probabilities with soft clipped frag type reads and use these partial reads to fill the gaps. In iteration 2–3, we use jump reads as unmapped reads and initialize probabilities with soft clipped frag type reads. The reason for using jump reads at the beginning part of the iterations is because of their comparative higher insert size. So, they were able to close more gaps and larger gaps at the beginning, that means the burden is going to be less for later iterations with less and small length gaps. In iteration 4 and 5, we use frag reads as partial reads and jump type reads as unmapped reads to fill the gaps respectively. Then finally, in iteration 6 – 8, we use frag reads as soft clipped reads and initialize probabilities with soft clipped frag type reads to complete the gap filling process. The reason for using these three iterations at ending part is that there are a lot of gaps which have small flanking regions on either side. So, using a library with higher insert will be useless in this scenario and thus the choice of frag type reads is made which has lower insert size.

At any point during the run of Figbird, if there is no change in gap length for the current iteration compared to previous one, we will break the loop and end the process there. To make these iterations faster, we will reduce the read set as well as the scaffold set ,i.e., only those scaffolds having gaps is kept and rest are removed. While placing the reads in gaps, we have chosen insert size threshold to be 3 times the left and right

standard deviations from the mean calculated from the fully mapped read pairs for that library of reads. Finally the number of EM iterations during gap filling were kept 3 for partially aligned reads and 200 for the case of unmapped reads, with a break condition from the loop if the consensus is complete before the limit or it is stuck for 5 iterations straight.

4.10 Adjustment of gap filling based on heuristics

4.10.1 Exploration of range for gap estimates likelihood calculation:

As discussed in Section 4.5, for each gap with length l_g , we will compute $l(\mathcal{G}; \mathcal{R})$ for all \mathcal{G} with length between $0.5 \times l_g$ to $2.5 \times l_g$. This is done for gaps with length ≤ 400 . Otherwise we will fill the gaps considering their original length. But gap lengths given are not always exactly accurate. So to avoid the problem of filling gaps exactly with the same length, we do the following:

```

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
ACGTACGTNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNACGTACGT
ACGTACGTACGTACGT.....ACGTACGTACGTACGTACGT

```

Figure 4.7: How consensus string is formulated at each EM iteration

Gaps are iteratively filled at each iteration as shown in figure 4.7. If the number of N in consensus sequence becomes less than two times the read length, then we stop EM iteration and break from the loop. As the gap length became smaller, it is evaluated as the range specified above in next iterations. Also, in the finalize step, for this type of gaps, we further cut read length characters from either side to make sure it gets evaluated using the range.

4.10.2 Detection of repetitive region:

Sometimes, in our gap filling process, we will skip some gaps from filling process based on some repeat detection condition. This is only applicable in case of read pairs with low insert size (< 250 bp) ,i.e., frag type reads (partial or unmapped) in our case. To detect such gap, what we do is following: For each read, we evaluate the following 2 conditions:

- If a chunk of characters (≥ 20) characters from left side of the gap start position is present in the read more than 1 time

- If a chunk of characters (≥ 20) characters from right side of the gap start position is present in the read more than 1 time

If both of this conditions are met for the same read, then the region is considered as highly repetitive and we don't fill that gap. Otherwise, if one of the above conditions is met and original Gap is greater than 6 times (A high value is given to make sure the gap length surpasses the insert size) the read length, the gap is not filled. The reason for doing so is that as it is a large gap and insert size for this condition is lower, then filling it with partial or unmapped frag does not help much. As we have to depend on jump library for these cases, so we avoided them from filling up incorrectly as jump reads were performing among the two libraries.

4.10.3 Negative overlap detection:

Sometimes there are some gaps present in scaffold where, there shouldn't have been a gap. That means the gap should be closed down and there is an indirect overlap between the left and right flank sequences of these gaps, which we are calling the negative overlap. We have considered this for gaps with length ≤ 30 (default value) which can be set by user. If the two flank sequences have an overlap greater than 5 bp and if there is any read with sufficiently long length that supports this merging of sequences, then we remove the gap and merge both sides into one and shrink the gap. The number of such gaps is very few (5 out of 1000) and only present in some assembly such as MSR-CA and SGA [91].

4.10.4 Maximum likelihood modification:

Likelihood modification for invalid reads: While placing a read in the gap, if it falls within the insert size range allowed, we place the read in certain range in all possible positions and calculate the best position of that read using the likelihood computed using the model parameters and probability distributions. Finally we add this probability to calculate the sum of likelihood for that gap estimate. Now, to solve the unnecessary read problem and incorrect gap length prediction specially in case of jump reads, we do the following: We compute the consensus sequence based on above placement of reads and for each read, find the probability of that read generating from that position. If the probability is less than \log (cutoff value), we accept this likelihood value in sum calculation, otherwise we add a penalty for that read (-50). The reason to put a fixed negative penalty is to discard a gap estimate of smaller length and stop shrinking the gap. If there are more valid reads falling in the gap, there is more used reads, thus gap estimate is close to actual one.

Likelihood modification based on coverage: We also modify our sum likelihood calculation based on gap coverage as shown in figure below:

ACGTACGTACGTNNNNNACGTACGNNNNNNNNNACGTTTTTA
x
y

Figure 4.8: Consensus string at each EM iteration

In case of such scenarios, we try to find the starting and ending position of such fragmented regions ,i.e., x and y in above figure and discard all the reads that fall in this region by adding a similar penalty (-50) sum of max likelihood calculation. Note that they are mostly the unnecessary reads as the correct reads tend to have overlap. Even if they are correct reads, we don't want these fragmented regions in our method and increase the final gap count.

Likelihood modification based on the overlapping characteristics: Based on the overlapping characteristics of all the reads either unmapped or partially assembled, a penalty scheme is added. The penalty scheme for unmapped reads contains the followings:

- Gap penalty: Gap found between reads aligned to the left flank and right flank and going inwards.
- Overlap threshold penalty: If overlap between the reads is less than 4 bp for that particular read placement.
- Left/ Right alignment advantage: A likelihood boost based on the count of the reads that has more than 4 character aligned to the left or right side of the gap.

In case of partially mapped reads, our overlapping characteristics are determined as follows. For each gap estimate, we call a function that detects whether there is a overlap between reads and does the following. If there is a read that covers the entire gap and matches with both left and right regions of the gap sequence with fixed low threshold of error, we add a positive value to the maximum likelihood. Otherwise, if there is a correct overlap with reads from both sides, then we find the overlap length and add positive value to maximum likelihood based on that count of overlaps. Else if there is a false overlap detected, then we give a penalty for that gap estimate.

4.10.5 Consensus probability update based on intermediate alignment:

In an ideal scenario, gaps are filled from one/both flanks and eventually merges at some EM iteration. But sometimes the consensus sequence gets updated circularly ,i.e., it changes back and forth into the same sequence every other iteration or it gets stuck as the EM iteration carries on at one/both sides as shown in figure below:

```

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
ACGTNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNACGTACGTACGTAC
ACGTNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNACGTACGTACGTAC
ACGTNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNACGTACGTACGTAC

```

Figure 4.9: Consensus string gets stuck after certain EM iterations

The reason for this is that the read to be placed next has very little overlap with the filled consensus sequence and thus our algorithm can't find the correct position to place it. The advantage of this process is not only to solve these issues but also to make the convergence process faster. EM is by default a slow algorithm and the probabilities take a long time to converge. So, to solve these problems, we will perform the update if a combination of conditions are met. They are:

- If the consensus is stuck for more than two iterations.
- If the length of the gap is ≥ 400 and thus for this gap estimate we didn't run a range of gap estimate calculation.

If both of these conditions are met, we do the following for each flank. We find those reads that has a certain threshold of base pair match with current filled consensus sequence of that flank. Conditions for accepting such read are that they are not yet accepted or placed, the read is in proper insert size range, segment taken from left or right flank must be greater than 20 and match count between prefix and suffix depends on the length of the read. Then finally, we update the probability with the consensus of all such reads considered.

4.10.6 Clearing a filled sequence based on threshold values:

Sometimes, during the iterations where unmapped reads are used, some gaps can be wrongly filled by our method. The reasons could be either because of wrong read placement due to erroneous reads or false alignment by bowtie2. In those cases, we have identified such gap sequences by performing a number of checks and for all such

cases, we have considered an overlap threshold that is arbitrarily 10% of the length of reads. Since the minimum read length in case of unmapped reads is 37 in our experiment, we have used the value 4 for all three datasets. The conditions for clearing such gaps are:

- If there is no aligned read with left and/or right flank such as shown below, then clear the gap.

```

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  ACGATACGATACGATACGAT          ACGATACGATACGATACGAT
    ACGATACGATACGATACGAT          ACGATACGATACGATACGAT
  
```

Figure 4.10: Zero alignment with both flanks

- There is a check for discontinuity among the placed reads where the overlap between the two reads is ≤ 2 , it is considered as a fragmented or discontinuous sequence and a read length amount character is going to be chopped off from the point of discontinuity to outwards direction in both sides.

```

          NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
ACGATACGATACGATACGAC
              ACGATACGATACGATACGAT
                  ATGATACGATACGATACGAT
  
```

Figure 4.11: An illustration of discontinuous placement of reads

- If the amount of left flank aligned character or right flank aligned character in consensus is less than overlap threshold then clear that particular sided aligned reads and build a new consensus.

Chapter 5

Experiments and results

In this chapter, we compare Figbird with four other state-of-the-art gap filling tools in literature that use short reads from SGS technology. The results in this chapter show that, Figbird is able to close more gaps reducing overall erroneous length and misassembly compared to the other tools.

5.1 The genome assemblies

To assess the performance of Figbird and to compare it with existing gap filling tools, we use the GAGE dataset [41]. It is a standard dataset that was generated to critically evaluate the genome assemblers and is also used to assess gap filling tools [67]. In this experiment, we use the data for two bacterial species *Staphylococcus aureus*, *Rhodobacter sphaeroides* as well as Homo sapiens Chromosome 14, as shown in Table 5.1, for which reference genomes are available. We collect a wide array genome assemblies for the three datasets generated using various tools as part of the GAGE project and performed gap filling using Figbird on each of them, and evaluate the results by comparing the filled sequences with the reference sequences.

Organism	Genome size	No. of assemblies	No. of gaps	Gap length range
<i>Staphylococcus aureus</i>	2 - 3 Mbp	8	35 - 654	1 - 3462
<i>Rhodobacter sphaeroides</i>	3 - 5 Mbp	9	38 - 938	1 - 4808
Human Chromosome 14	80 - 150 Mbp	9	1061 - 51567	1 - 34745

Table 5.1: Genomes used in evaluation

5.2 The sequenced reads

As part of the GAGE datasets, second generation sequencing reads were collected for the three different genomes specified before. The sequenced reads available in GAGE define a crucial role in the evaluation and design process of sequencing experiments and thus it is standard to use these reads for our research purpose. In GAGE, there are two libraries available for each of these three datasets. For our experiment, we use both the fragment (paired-end) as well as the short jump (mate pair) libraries. The details about the short reads libraries used in this experiment are listed in Table 5.2.

Organism	Library	Mean Insert Size	Read length	No. Read Pairs
<i>Staphylococcus aureus</i>	Frag	180	101	1,294,104
	Shortjump	3500	30 - 37	1,614,660
<i>Rhodobacter sphaeroides</i>	Frag	180	101	2,050,868
	Shortjump	3500	50 - 101	1,526,850
Human Chromosome 14	Frag	155	101	36,504,800
	Shortjump	2283 - 2803	50 - 101	14,054,994

Table 5.2: Read sets used in evaluation

The coverage is 45X for all cases, which indicates that the library has coverage high enough for gap filling purpose. For more details about the different assemblies and read sets, readers are encouraged to check the official [GAGE](#) website. For sequences from the short jump library, we use Quake [92] corrected versions of the reads for better accuracy due to its conservative nature of error correction mechanism [93]. We run Bowtie2 version 2.2.3 to align these read pairs to the gapped scaffolds for our experiment and then fill the gaps using Figbird.

5.3 Configurations of the tools used for comparison

We compare the performance of our tool Figbird with the four state-of-the-art tools available for filling gaps using short reads which are, SOAPdenovo’s stand-alone tool GapCloser v1.12-r6 [64], GapFiller v1.10 [60], Gap2Seq v1.0 [67] and Sealer [66]. For GapFiller, both BWA [94] and Bowtie [76] aligners are used. All experiments are run using 24 cores on a machine with Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz processors. We used the time.h library in C++ to measure the time taken for each of these tools and used the Python script [Memusg](#) to benchmark peak memory usage.

5.4 Quality Evaluation of Filled Sequence

As gap-filling is the last stage of the genome assembly pipeline, the thorough evaluation of the quality of the filled sequences is extremely crucial. A wrongly introduced sequence will affect the subsequent metagenomic analysis if the gap region falls into the human protein-coding genes. Also, considering the fact that contig construction stops at regions with low coverage and highly repetitive regions, we have chosen to assess the quality using QCAST [40]. QCAST uses nucmer [45] to find alignments between the gap-filled assembly and the reference sequence. For this experiment, we used QCAST v2.3 and the outputs were modified using a python script used in [67] to make corrections in the classification of “misassemblies” as depicted in [67]. The value of N is chosen to be 4000, since it is an upper bound of the insert distance of the mate pair libraries.

To assess the quality of the filled sequence and the robustness of our method, we list six different metrics reported by the modified version of QCAST on every assembly. A short note on each metric follows:

- Misassemblies: The number of misassembled sequences in a scaffold that are larger than M bp. Here, M is chosen to be 4000, which is the upper bound on the insert size of the mate pair libraries used in this experiment.
- Erroneous length: It is the sum of the lengths of all mismatches, indels and local misassemblies, i.e., the length of the misplaced sequence is $\leq M$
- Unaligned length: The total length of the unaligned sequence in an assembly.
- NGA50: NG50 is the size of the longest scaffold such that at least half of the reference genome is contained by scaffolds longer than it. NGA50 is the NG50 after scaffolds have been broken at every position where a local misassembly or misassembly has been found.
- Number of gaps: The total number of gaps, i.e., a contiguous sequence of ‘N’ remaining where gap length can be ≥ 1
- Total gap length: The sum of remaining amount of unknown nucleotide positions denoted by ‘N’ in the filled assembly.

5.5 Findings on the GAGE datasets

Table 5.3: Quality of the original and the gap-filled assemblies of the *Staphylococcus aureus* genome using various tools.

Tool	Original	Gap-Closer	GapFiller bowtie	GapFiller bwa	Gap2-Seq	Sealer	Figbird
ABySS							
Misassmblies	5	40	0	-20	60	20	20
Erroneous-length	10587	26.3	-3.5	31	70.5	32.5	-21.8
Unaligned-length	7935	-21.9	-10.2	-10.2	-43	0	-7.9
NGA50	31079	0	0	0	0.3	0.3	-1.2
Number of gaps	69	-18.8	-13	-29	-87	-71	-29
Total gap length	55885	-24.9	-9.5	-26	-94.5	-59.4	-17.6
ABySS2							
Misassmblies	5	20	0	40	40	40	20
Erroneous-length	10312	-3.7	0.5	0.1	-27.4	-13.7	-30.9
Unaligned-length	0	0	0	0	0	0	0
NGA50	106796	15.1	0	0	29	0	3.6
Number of gaps	35	-34.3	-11.4	-31.4	-80	-80	-57.1
Total gap length	9393	-60.8	-29.9	-53.2	-94.5	-93	-79.9
Allpaths-LG							
Misassmblies	0	0	1	1	0	0	0
Erroneous-length	5991	-22.7	-5.9	-8.4	9.7	55.5	-26.9
Unaligned-length	0	0	0	0	0	0	0
NGA50	110168	2.7	35.9	69.6	48.5	35.9	31.5
Number of gaps	48	-47.9	31.2	-41.7	-70.8	-68.8	-52.1
Total gap length	9900	-74.4	-23.1	-40.8	-94.7	86.6	-72.9
Bambus2							
Misassmblies	0	1	0	0	0	0	0
Erroneous-length	24570	-30.6	-4.4	16.9	-1.4	-19	-15.6
Unaligned-length	0	0	0	0	0	0	0
NGA50	40233	34.8	1.6	7.3	17.2	24	17.2
Number of gaps	99	-67.7	-14.1	-18.2	-69.7	-53.5	-62.6
Total gap length	29205	-78	-23.9	-37.2	-84.1	37.8	-76.7
MSR-CA							
Misassmblies	10	-30	-30	-30	-20	20	-30
Erroneous-length	17276	-2.6	0.3	1.8	-4	-12.3	-9.1
Unaligned-length	0	0	0	0	0	0	0
NGA50	64114	50.3	20.4	20.4	50.3	45.3	58.3
Number of gaps	81	-51.9	-19.8	-29.6	-56.8	-50.6	-61.7
Total gap length	10353	-76.4	-24.5	-39.4	-70.5	-47.3	-73.3

(continued)

Table 5.3 – Continued

Tool	Original	Gap-Closer	GapFiller bowtie	GapFiller bwa	Gap2-Seq	Sealer	Figbird
SGA							
Misassmblies	2	0	0	0	-50	-50	-50
Erroneous-length	13811	-49.3	-19.9	-31.4	-10.6	-16.8	-53.8
Unaligned-length	0	0	0	0	0	0	0
NGA50	9541	123.7	8.9	9.7	221.4	173.3	169.1
Number of gaps	654	-75.1	-20.8	-37.3	-80.1	-78.7	-82.3
Total gap length	300607	-54.3	-5.7	-10.1	-72.1	-61.6	-83.2
SOAPdenovo							
Misassmblies	2	0	0	0	0	0	0
Erroneous-length	35433	-1.3	1.2	.7	-1.5	-0.2	-2.4
Unaligned-length	4055	-100	-100	-100	3.9	0	-7.5
NGA50	69834	0	0	0	0	0	0
Number of gaps	9	-22.2	-22.2	-33.3	-55.6	-22.2	-33.3
Total gap length	4857	-60.4	-24	-30.8	-94.2	-1.2	-27.7
Velvet							
Misassmblies	25	8	0	4	8	12	4
Erroneous-length	24160	-32.1	-2.2	-18.8	-36.3	13.8	-15.6
Unaligned-length	1270	-49.4	-20.5	-21.3	-49.4	0	-49.4
NGA50	46087	19.1	26	49	73.3	20.9	51.5
Number of gaps	128	-46.9	-30.5	-41.4	-68.8	-52.3	-54.7
Total gap length	17688	-59.6	-37.9	-49.2	-81.2	-29.2	-68.8
Total (average %)							
Misassmblies	49	5	-3	0	-3	6	-4
Erroneous-length	142140	-14	-4	-1	-2	5	-22
Unaligned-length	13260	-21	-16	-16	-12	0	-8
NGA50	477852	31	12	20	51	37	65
Number of gaps	1123	-45	-20	-32	-71	-59	-54
Total gap length	437888	-61	-22	-36	-85	-52	-62

Table 5.4: Quality of the original and the gap-filled assemblies of the *Rhodobacter sphaeroides* genome using various tools.

Tool	Original	Gap-Closer	GapFiller bowtie	GapFiller bwa	Gap2-Seq	Sealer	Figbird
ABySS							
Misassmbles	20	0	0	0	5	5	0
Erroneous-length	140634	1	-2.5	0.1	-0.4	-0.8	-1.2
Unaligned-length	23522	-7.2	100.9	70.8	-10	-10	106.6
NGA50	6538	0.2	0.6	0.2	4.4	2.7	0.2
Number of gaps	323	-20.1	-8.7	-19.8	-45.8	-39.6	-36.5
Total gap length	114587	-3.2	-0.3	-3.7	-19.6	-8.6	-5.6
ABySS2							
Misassmbles	12	8.3	0	0	133.3	0	0
Erroneous-length	15750	13.5	-0.4	0	38.5	-5	0.3
Unaligned-length	8230	-0.4	-1.9	-36.1	0	0	-39.6
NGA50	31197	11.1	0	3.8	11.1	5.7	-12.1
Number of gaps	292	-21.2	-1.4	-5.8	-19.9	-10.3	-11.3
Total gap length	62627	-9.9	2.6	-9	-38.7	-8.7	-6.3
Allpaths-LG							
Misassmbles	5	20	0	0	0	0	0
Erroneous-length	11738	97.1	-2.6	-2	3.4	-2.7	0.6
Unaligned-length	0	0	0	0	0	0	0
NGA50	79634	11.5	2	0	12.8	0	0
Number of gaps	170	-3.5	-3.5	-3.5	-9.4	-7.6	-10.6
Total gap length	21409	-13.4	8	1.2	-25	-10.2	-10.7
Bambus2							
Misassmbles	5	0	0	0	0	0	0
Erroneous-length	106359	-0.3	-0.6	-0.8	0.5	-4.2	-0.5
Unaligned-length	4716	-0.7	-2.7	-5.5	-100	0	-11.5
NGA50	15043	0	0	0	1.3	0.6	0.5
Number of gaps	85	-15.3	-5.9	-7.1	-35.3	-21.2	-11.8
Total gap length	57041	-14.2	-9.6	-13.6	-31.9	-14.9	-23.8
CABOG							
Misassmbles	15	0	0	-13.3	-13.3	-13.3	0
Erroneous-length	16750	44.1	0.3	0.4	-1.9	-1.4	-2.1
Unaligned-length	0	0	0	0	0	0	0
NGA50	26819	0.8	11.4	11.4	3.9	3.9	24.8
Number of gaps	193	-2.6	-2.1	-4.7	-9.3	-5.7	-18.1
Total gap length	21547	-13	5.7	-3.9	-22.5	-7.3	-10.5

(continued)

Table 5.4 – Continued

Tool	Original	Gap-Closer	GapFiller bowtie	GapFiller bwa	Gap2-Seq	Sealer	Figbird
MSR-CA							
Misassmbles	-20	-20	0	10	270	-20	-10
Erroneous-length	22522	-0.8	2.6	8.4	19.7	-0.9	1.5
Unaligned-length	1377	0	0	0	0	0	0
NGA50	75776	-5.3	19	20	13.9	-1.1	44
Number of gaps	356	-12.1	-7.3	-10.4	-26.4	-5.3	-29.5
Total gap length	32628	-19.9	3.4	-7	-67.1	-9.8	-27.3
SGA							
Misassmbles	2	0	0	0	1600	0	50
Erroneous-length	58135	4.1	-2.9	-5.1	33.9	-13.3	-1.3
Unaligned-length	69226	-0.7	-12.5	-13.2	-41	-24.4	-24.3
NGA50	2601	5.7	1.2	5.2	98	31.8	8.6
Number of gaps	938	-8.6	-3.9	-7.7	-37.1	-18.8	-13.2
Total gap length	1145600	-2.2	-0.3	-2.7	-23.3	-9.4	-10.4
SOAPdenovo							
Misassmbles	3	0	0	0	33.3	0	0
Erroneous-length	56228	8.7	-0.1	0.1	-7.1	-0.5	0.9
Unaligned-length	0	0	0	0	0	0	0
NGA50	27434	0	-1.2	-1.2	0	0	-1.2
Number of gaps	38	0	0	0	-10.5	-2.6	-10.5
Total gap length	10461	-10	2.4	0.3	-20.2	-3.5	-17.3
Velvet							
Misassmbles	19	15.8	0	-15.8	10.5	0	-26.3
Erroneous-length	40419	-14.5	-4.5	3.2	-5.4	-5.5	10.9
Unaligned-length	28344	-3	-5.9	-7.6	-17.4	-1.5	-22.7
NGA50	54238	0.3	-9.8	-0.9	0	0	-16.4
Number of gaps	427	-11.2	-5.4	-9.4	-21.5	-13.3	-29.3
Total gap length	86815	-7.6	0	-6.3	-26.3	-3.8	-13.3
Total (average %)							
Misassmbles	91	3	0	-2	227	-3	2
Erroneous-length	468535	18	-1	1	10	-3	2
Unaligned-length	135415	-1	9	1	-18	-3	-5
NGA50	319280	2	3	5	17	5	6
Number of gaps	2822	-10	-4	-7	-23	-13	-18
Total gap length	1552715	-10	2	-4	-30	-8	-13

Table 5.5: Quality of the original and the gap-filled assemblies of Human Chromosome 14 using various tools.

Tool	Original	Gap-Closer	GapFiller bowtie	GapFiller bwa	Gap2-Seq	Sealer	Figbird
ABySS							
Misassmblies	3	133	33.3	0	100	66.7	0
Erroneous-length	190458	18.2	7.6	7.4	-9.4	-17.6	8.5
Unaligned-length	262068	-16.6	-28.9	-34.1	-8.4	-3.6	-19.5
NGA50	1320	1	0.5	0.7	1.3	1.2	0.8
Number of gaps	1061	-5.9	-28.9	-32.5	-33.4	-46.6	-27.2
Total gap length	585628	-24.5	-23.4	-27.5	-25.5	-25.6	-22.9
ABySS2							
Misassmblies	99	18.2	2	4	5.1	0	8.1
Erroneous-length	555099	15.9	1.5	2.8	3.5	-6.5	3.1
Unaligned-length	157759	-21.4	-28.8	-36.3	-15.4	-2.2	-26.4
NGA50	11869	4.1	1.5	3	2.4	2.9	3.2
Number of gaps	2820	-14.5	-29.5	-38.5	-23.9	-22.69	-43.8
Total gap length	949137	-34.9	-10.6	-25.3	-36.8	-15.5	-28.1
Allpaths-LG							
Misassmblies	95	-6.3	5.3	8.4	14.7	2.1	6.3
Erroneous-length	667229	34.5	-0.6	7.8	-3	-3.4	8.6
Unaligned-length	36941	-14.3	-11.1	-11.9	26.8	12.3	3.8
NGA50	34534	48.3	20.7	23	23.3	6	32.5
Number of gaps	4307	-35.1	-19.3	-20.6	-29.8	-17.7	-33.6
Total gap length	3227193	-37.9	-16	-17.2	-16	-7.2	-39.4
Bambus2							
Misassmblies	1584	3.1	2.3	5.4	2.1	-0.2	3.4
Erroneous-length	11114542	-9.6	-0.2	14.8	-0.6	-18.2	-3.9
Unaligned-length	161358	-42.7	-35.8	-31.3	2.5	-33.8	-37.3
NGA50	3045	34.8	12	15.2	1.8	23.7	25.1
Number of gaps	11809	-16.4	-2.3	-2.3	-6.6	-22.3	-1.6
Total gap length	10370362	-45.6	-18.9	-29.3	-4.6	-21.4	-46.6
CABOG							
Misassmblies	91	16.5	7.7	5.5	7.7	2.2	4.4
Erroneous-length	615239	19	-2.2	0.2	-3.5	-4.4	-0.3
Unaligned-length	2506	0	0	0	0	0	0
NGA50	46665	16.2	58.8	62.3	8.9	11.9	39.8
Number of gaps	3043	-18.9	-46.5	-51.2	-13.8	-15.1	-38.6
Total gap length	231078	-50.3	-34.9	-41.2	-22.6	-19.9	-32.5

(continued)

Table 5.5 – Continued

Tool	Original	Gap-Closer	GapFiller bowtie	GapFiller bwa	Gap2-Seq	Sealer	Figbird
MSR-CA							
Misassmblies	1110	14.5	9.9	25.3	6.8	1.3	13.3
Erroneous-length	5412965	2.8	3.9	21.7	-6	-6.5	-4.9
Unaligned-length	318421	-30.5	-28.8	-33	-13.1	-9	-31.2
NGA50	5704	73.3	65.9	77.7	32.9	18.9	52.5
Number of gaps	30622	-34.9	-42.4	-49.5	-27.8	-26	-46.5
Total gap length	6097928	-49.3	-37.8	-49.2	-18.1	-11.3	-48.3
SGA							
Misassmblies	8	375	37.5	87.5	287.5	12.5	150
Erroneous-length	1580489	21.1	-18.4	-13.9	-24.6	-26.6	-0.6
Unaligned-length	1160159	-83.9	-82.8	-87	-38.6	-22.1	-90.5
NGA50	2644	244.2	206.6	239.3	149.1	80.4	164.6
Number of gaps	21459	-56.7	-46.3	-49.9	-51.5	-39.4	-49
Total gap length	12840408	-53.5	-50	-55.4	-30.2	-17.6	-62.9
SOAPdenovo							
Misassmblies	1250	17.1	3.5	9.4	11.7	1.6	13.1
Erroneous-length	8449941	-1.3	1.1	3.2	-0.5	-4.7	0.2
Unaligned-length	1306173	-28.8	-24.4	-28.1	-14.1	-10.1	-27.2
NGA50	6592	17.4	3.6	4.1	4	7.5	4.9
Number of gaps	8544	-25.2	-4.5	-5.1	-18.8	-11.7	-5.7
Total gap length	10255930	-21.3	-11.3	-15.9	-6.3	6	-18
Velvet							
Misassmblies	9308	26.3	24.8	51.5	13.3	10.5	24
Erroneous-length	12531431	-10.4	32.3	58.6	-0.5	-23.1	23.5
Unaligned-length	23484076	-58.4	-54.9	-70	-20.7	-31	-52.4
NGA50	1793	104.4	49.6	67.5	27	65.1	43.1
Number of gaps	51567	-43.4	-24.1	-26.6	-26.6	-37	-21.5
Total gap length	63559964	-22.8	-14.7	-22.8	-4.2	-11.8	-17.1
Total (average %)							
Misassmblies	13449	67	15	22	50	11	20
Erroneous-length	40562294	11	3	12	-4	-12	4
Unaligned-length	26731702	-32	-32	-36	-9	-11	-34
NGA50	102297	61	47	55	28	25	40
Number of gaps	132412	-27	-27	-30	-25	-26	-29
Total gap length	107168491	-37	-24	-31	-18	-15	-34

5.6 Discussions

In this section, we present the performance comparison of Figbird with four other state-of-the-art gap filling tools that use short reads as discussed in Section 5. The detailed evaluation results from QUASt are provided in Tables tables 5.3 to 5.5. Each table shows the percentage increase or decrease in the six evaluation metrics achieved by the five gap filling tools along with the original values before gap filling. For each assembly, these relative percentages are determined using the differences in evaluated values between original assembly and gap filled assembly using QUASt. The results over all the assemblies are presented in the bottom of the tables named ‘Total’, which is obtained by determining the average for a particular metric over all the assemblies for each gap filling tool. The overall percent reduction in gap length as well as percent changes in misassembly and errors for all three datasets are also shown in Figure 5.1. We focus on these three metrics as unaligned length and NGA50 are related to these while filling a gap partially lead to an increase in number of gaps which we believe is misleading. This cumulative way of representation of the total result instead of value based method helps us to remove bias from the result in case of the presence of some extremely large valued metrics among smaller ones. From the overall results, we observe that, Figbird is able to close considerably high portion of gap regions yet managing to keep the erroneous length and misassembly low compared to the other state of the art tools. It is noteworthy to mention here that the results obtained in this work are deterministic by nature and contain no element of randomness in the model despite being called a probabilistic model. The sole reason for calling the model probabilistic is that it is based on a number probability distributions learned from fully aligned read pairs which indicate the probable position of the unmapped or partially mapped end of a read pair in a range of insert size possibilities.

For the *Staphylococcus aureus* dataset, we observe from the scatter plots in Figure 5.1A that none of the other tools have managed to fill more nucleotides than Figbird while achieving better accuracy in terms of error or misassembly, i.e., there are no points in the scatter plots that are to the right and below the points representing Figbird. We can also see from Table 5.3 that Figbird has reduced the total misassembly rate by 4%, which is the best among all other tools, and reduced erroneous length by 22% thus outperforming the next best tool GapCloser in this respect by 8%. The only tool which is able fill a higher portion of gaps than Figbird is Gap2Seq which is at the expense of higher rate of misassembly and substantially higher amount of erroneous sequences.

In case of our second bacterial dataset *Rhodobacter sphaeroides*, a similar trend can

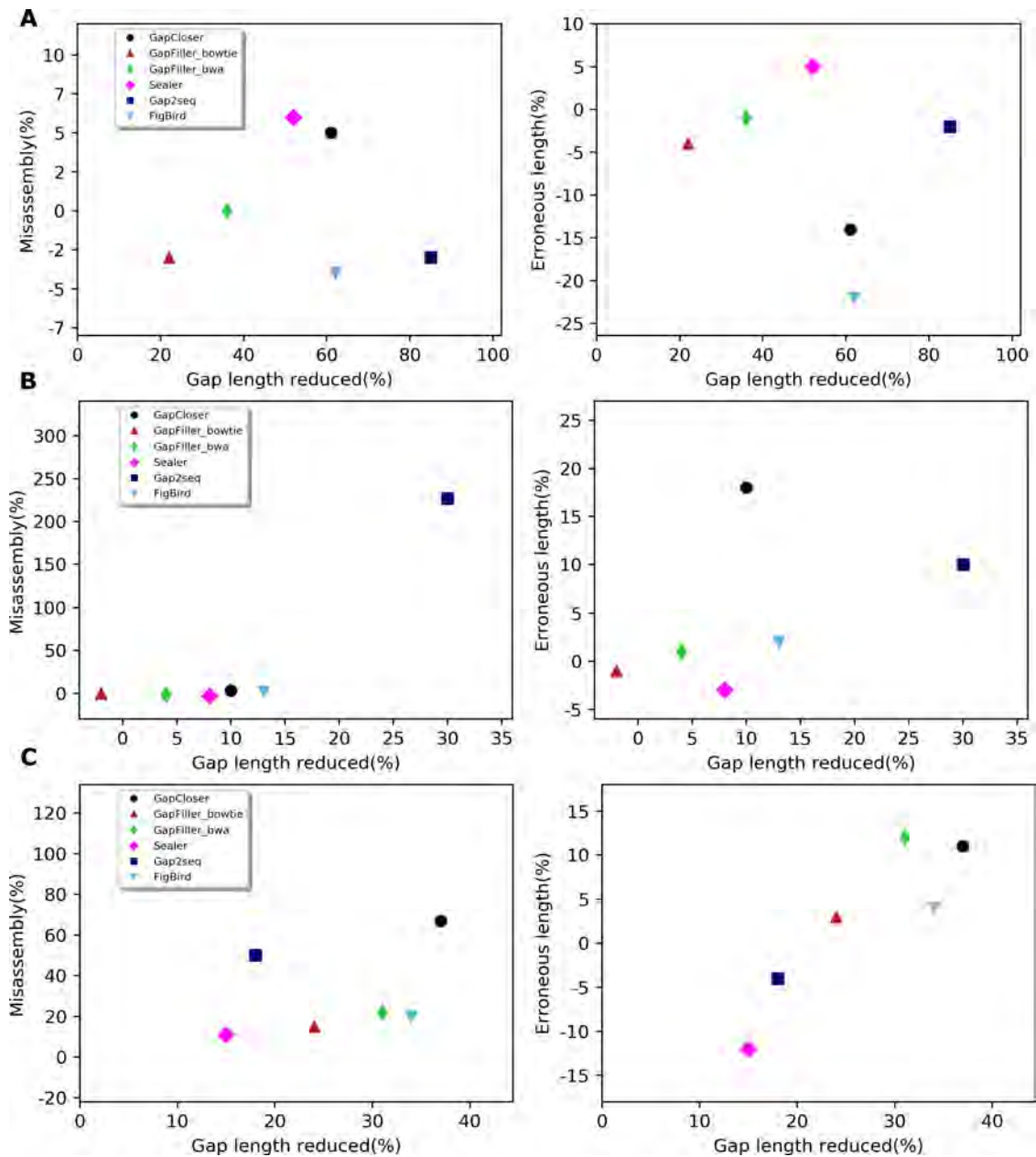


Figure 5.1: Scatter plots showing the performance of Figbird compared to other gap filling tools on (A) *S. aureus*, (B) *R. sphaeroides* and (C) Human Chromosome 14 datasets in terms of average percentage of misassemblies and erroneous length against the average percentage of nucleotides filled by each tool. The values are computed by determining the percent change in misassembly and erroneous length, and percent reduction in gap length obtained by each tool for all the de novo draft genome assemblies from GAGE, and then averaging over the values for all those assemblies.

be observed from Figure 5.1B like the previous dataset. None of the other tools have outperformed Figbird which has managed to close second highest amount of ‘N’s, only behind Gap2Seq. However, Gap2Seq increases misassembly and erroneous length by 227% and 10% respectively which are substantially higher than the 2% misassembly and 2% erroneous length increase by Figbird, as shown in Table 5.4. It is worth highlighting that the sequencing error rate for *R.sphaeroides* dataset is higher compared to that of *S.aureus* dataset and the sequencing reads being more error-prone, mapping tools finds it difficult to align reads properly to the scaffolds [95], and also leads to errors during gap filling. This shows that Gap2Seq lacks robustness to sequencing errors and may introduce large amount of errors whereas Figbird is still performs in a balanced way without introducing massive amounts of misassemblies and errors.

Finally, detailed results of QAST evaluation on the Human Chromosome 14 (HC14) dataset is summarized in Table 5.5. Despite the highly repetitive nature of HC14 dataset as suggested in [64], three of the tools, Figbird, GapCloser and GapFiiler filled more than 30% nucleotides. From the overall comparison presented in Figure 5.1C, it can be observed that Figbird is second to GapCloser in terms of the amount of gap filled. But the additional 3% gap filled by GapCloser comes at the expense of 47% and 7% more misassemblies and erroneous length respectively compared to Figbird. Moreover, GapCloser fills smaller amount of gaps while making more misassemblies and errors than Figbird in the other datasets. On the other hand, Gap2Seq, which is able to reduce the gap length by the highest amount in the other two datasets, fills 16% less gaps than Figbird despite 30% more misassemblies. Overall, our EM based method shows pareto optimal performance with respect to amount of gap filled, misassemblies and erroneous sequence introduced in all three datasets, i.e., no other tool is able to fill more nucleotides while making less misassemblies or errors than Figbird. Figbird has achieved best or near best performance score for all the evaluation metrics in every dataset used in evaluation suggesting the usefulness of our approach of gap filling procedure.

5.7 Time and memory usage comparison

Table 5.6: Gap-closing performance of the tools mentioned in the paper on 8 draft genome assemblies of *Staphylococcus aureus*.

Assembly	Software	Time (min)	Memory (GB)
ABySS	GapCloser	0.2	0.68
	GapFiller-bwa	5.8	0.12

	Sealer	3.7	20
	Gap2seq	0.2	1.95
	Figbird	10	0.7
	GapCloser	0.23	0.67
ABySS2	GapFiller-bwa	5.6	0.15
	Sealer	3.7	20
	Gap2seq	0.2	1.95
	Figbird	16	0.72
	GapCloser	2.2	0.71
Allpaths-LG	GapFiller-bwa	5.6	0.12
	Sealer	3.7	20
	Gap2seq	0.3	1.95
	Figbird	39	0.71
	GapCloser	0.23	0.72
Bambus2	GapFiller-bwa	7.5	0.12
	Sealer	3.6	20
	Gap2seq	0.45	1.9
	Figbird	63	0.75
	GapCloser	0.26	0.73
MSR-CA	GapFiller-bwa	6.6	0.11
	Sealer	3.8	20
	Gap2seq	0.65	1.92
	Figbird	34	0.79
	GapCloser	0.15	0.73
SGA	GapFiller-bwa	15.2	0.15
	Sealer	3.6	20
	Gap2seq	2	1.9
	Figbird	77	0.9
	GapCloser	0.16	0.69
SOAPdenovo	GapFiller-bwa	4.7	0.11
	Sealer	3.6	20
	Gap2seq	0.3	1.9
	Figbird	10	0.7
	GapCloser	0.17	0.71
Velvet	GapFiller-bwa	7.5	0.12
	Sealer	3.8	20
	Gap2seq	0.9	1.9
	Figbird	35	0.8

Table 5.7: Gap-closing performance of the tools mentioned in the paper on 9 draft genome assemblies of *Rhodobacter sphaeroides*.

*The performance metric values for Gap2seq tool has been taken from their paper [67] as the tool could not be evaluated due to the high run time and memory constraints.

Assembly	Software	Time (min)	Memory (GB)
ABySS	GapCloser	0.1	0.41
	GapFiller-bwa	17	0.11
	Sealer	4.2	20
	Gap2seq*	800	2.1
	Figbird	22	0.78
ABySS2	GapCloser	0.11	0.42
	GapFiller-bwa	21.3	0.12
	Sealer	4.1	20
	Gap2seq*	998	2.1
	Figbird	25	0.8
Allpaths-LG	GapCloser	10.9	0.82
	GapFiller-bwa	7.3	0.12
	Sealer	4.3	20
	Gap2seq*	578	2.2
	Figbird	19	0.79
Bambus2	GapCloser	0.35	0.61
	GapFiller-bwa	10.1	0.14
	Sealer	4.5	20
	Gap2seq*	991	2.1
	Figbird	176	0.83
CABOG	GapCloser	0.2	0.55
	GapFiller-bwa	13.8	0.11
	Sealer	4.2	20
	Gap2seq*	121	2.2
	Figbird	23	0.8
MSR-CA	GapCloser	0.25	0.64
	GapFiller-bwa	20.8	0.11
	Sealer	4.3	20
	Gap2seq*	858	2.5
	Figbird	68	0.8
SGA	GapCloser	0.2	0.67
	GapFiller-bwa	30.1	0.15
	Sealer	4.2	20
	Gap2seq*	10e+3	2.1
	Figbird	91	1.1

SOAPdenovo	GapCloser	0.3	0.59
	GapFiller-bwa	2.4	0.12
	Sealer	4.2	20
	Gap2seq*	105	2.1
	Figbird	10	0.8
Velvet	GapCloser	0.1	0.58
	GapFiller-bwa	13.1	0.12
	Sealer	4.1	20
	Gap2seq*	1221	2.1
	Figbird	54	1.1

Table 5.8: Gap-closing performance of all the tools on 9 draft genome assemblies of Human Chromosome14 genome.*The performance metric values for Gap2seq tool has been taken from the corresponding paper [67] as the tool could not be evaluated due to the high run time and memory constraints.

Assembly	Software	Time (hr)	Memory (GB)
ABySS	GapCloser	0.14	6.5
	GapFiller-bwa	1.7	0.11
	Sealer	0.31	20
	Gap2seq*	166	22
	Figbird	2.2	7.5
ABySS2	GapCloser	0.16	6.4
	GapFiller-bwa	2.9	0.14
	Sealer	0.32	20
	Gap2seq*	333	17
	Figbird	3.5	7.5
Allpaths-LG	GapCloser	24	8.2
	GapFiller-bwa	6.3	0.17
	Sealer	0.52	20
	Gap2seq*	8e+3	21
	Figbird	8	5.5
Bambus2	GapCloser	0.19	6.8
	GapFiller-bwa	13.1	0.12
	Sealer	0.5	20
	Gap2seq*	16e+3	25
	Figbird	17	5.4
	GapCloser	0.08	6.5

	GapFiller-bwa	4.4	0.11
	Sealer	0.35	20
	Gap2seq*	167	14
	Figbird	5.4	5.3
MSR-CA	GapCloser	0.4	7.6
	GapFiller-bwa	13.8	0.14
	Sealer	0.8	20
	Gap2seq*	13e+3	23
	Figbird	29	6.1
SGA	GapCloser	0.3	6.5
	GapFiller-bwa	13.1	0.11
	Sealer	0.6	20
	Gap2seq*	15e+3	23
	Figbird	32	8.9
SOAPdenovo	GapCloser	0.25	6.8
	GapFiller-bwa	5.2	0.3
	Sealer	0.5	20
	Gap2seq*	14e+3	25
	Figbird	9	7.1
Velvet	GapCloser	1.2	7.7
	GapFiller-bwa	41.7	0.2
	Sealer	1.1	20
	Gap2seq*	16e+3	27
	Figbird	38	11.5

We have compared the performance of Figbird in terms of run time and peak memory usage with four other tools in literature as mentioned in Section 5. The detailed commands and parameters used to run each of these tools can be found in Section 5.8. Table 5.8 shows the time and memory usage of the tools on the Human Chromosome 14 dataset. The comparisons on other two datasets are summarized on Table 5.6 and 5.7. From Table 5.8, we can see that, GapCloser and Sealer are the two fastest tools in terms of run time across all the assemblies while Gap2Seq is the slowest among all. Figbird and GapFiller-bwa have moderate time requirements. In case of memory usage, Gap2seq and Sealer requires the highest amount of memory due to their problem formulation whereas GapFiller requires the lowest. The memeory requirement of Figbird is almost similar to that of GapCloser, taking less memory in case of Allpaths-LG, CABOG, MSR-CA etc. while taking slightly more memory in case of Velvet, ABySS etc. Overall,

we find that although there are gap filling tools with lower time and memory usage than Figbird, it falls within the time and memory requirement range of the state-of-the-art tools. The increased run time can be regarded as a trade-off with the improved performance. Moreover, the time and memory requirements of Figbird scale linearly with the number and lengths of gaps, and the number of reads making it applicable to large datasets unlike some of the other gap filling tools.

5.8 Commands and configurations used to run the tools

To evaluate using GapCloser and GapFiller, we have provided a separate configuration file for each of them during each ru assembly run. Following are the commands with specific parameters used to run for each tool. All other parameters that are not mentioned in commands have been kept default.

- GapCloser:
`./GapCloser -t 12 -l 101 -a scaffold.fa -b dataset.config -o output.fa`
- GapFiller:
`perl GapFiller.pl -l dataset.config -s scaffold.fa -i 10 -T 24 -b output.fa`
- Gap2Seq:
`./Gap2Seq -scaffolds genome.scf.fasta -filled genome.scf.fill.fasta -reads frag1.fastq frag2.fastq,shortjump1.fastq,shortjump2.fastq`
- Sealer:
`./abyss-sealer -b40G -k90 -k80 -k70 -k60 -k50 -k40 -k30 -o output -S scaffold.fa -j 24 -P 10 -B 3000 -F 4000 frag1.fasta frag2.fasta shortjump1.fasta shortjump2.fasta`
- Figbird:
`./RunFigbird.sh Config.json > output.txt`

In the configuration files for GapCloser, GapFiller and Figbird, we have given average insert size as 180 for frag library in case of *S.aureus* and *R.sphaeroides* dataset as input. In case of Human Chromosome 14 we have used 160. For jump library, the insert size is 3500 for *S.aureus* and *R.sphaeroides* dataset and 2500 for Human Chromosome14 dataset for all three above mentioned tools. The standard deviation in GapFiller tool has been given 0.25 for all three datasets.

Chapter 6

Conclusions

In this thesis, we have presented Figbird, a bioinformatics tool that can fill withstanding gaps in scaffolds in the genome assemblies using second generation sequencing reads. Genome assemblies still contain thousands of gaps and it is extremely important to fill those gaps as much as possible, as a lot of those can contain exons or even a full genome providing that the gap length is huge. Such a gapless genome can significantly help in downstream analysis and with Figbird, we can fill those gaps with minimal introduction of erroneous sequence and achieve better outputs than contemporary tools in literature.

As gap filling is the last stage of genome assembly pipeline and the easy regions of the genome have already been reconstructed by the contig assembler in previous stages, thus only the complex regions are left at this stage to fill up. So, the main objectives of our study was to incorporate crucial information such as insert size distribution, sequencing errors, aligned read statistics and sequence quality during gap filling to correctly estimate the true lengths of gaps. We developed a likelihood based method similar to the Expectation-Maximization algorithm for gap filling based on a generative model for sequencing namely CGAL. Our proposed methodology can be summarized into two steps namely preprocessing step and gap filling step. In the preprocessing step, two different types of read pairs, i.e., pair-end and mate-pair reads generated by second generation Illumina sequencing technology were corrected using a read correction tool Quake for better accuracy. Then those reads were aligned to the draft genome assembly containing gaps using Bowtie2 software. Then partially mapped as well as fully unmapped read sequences will be assigned to appropriate gaps based on the parsed alignment information recorded in SAM format which will later be used in gap filling step. During the gap filling step, each of the four bases were initially be assumed to be equally probable in each gap position and the following two steps similar to the expectation-maximization (EM) algorithm was applied iteratively. In E-step, each read

will be placed in each gap position and the probability that the read originated from that position will be computed. In the M-step, the probabilities of the four bases at each gap position will be adjusted using the probabilities of reads computed in the previous E-step. With the help of this novel methodology along with the Gaussian distributions accounting for sequencing errors and insert size, we were able to fill gaps with more accuracy and achieved better results across various assemblies from standard GAGE datasets.

6.1 Discussions

In this thesis work, we have used read pairs generated by high throughput second generation sequencing machines: Illumina Genome Analyzer II and Illumina HiSeq 2000, which are comparatively accurate than the third generation technologies such as PacBio or MinION. The reason for doing so is firstly, that the error rates in sequencing for PacBio and MinION are much higher compared to that of Illumina. So a more conservative approach should be taken in these cases since errors in this step will accumulate in later steps of assembly. Secondly, there is also an issue of coverage in case of Illumina since the coverage is higher for this than the other two despite being cost efficient and higher chance of availability. As gap filling is the last step of genome assembly pipeline, we have chosen to use accurate short reads which will provide minimal errors in assembled gap sequences.

Unlike other state-of-the-art methods which focus on a graph based solution, our method uses an expectation maximization approach which bypasses the inherent complexities associated with the local assembly of reads or k-mers from the reads and thus it can circumvent the obstacles of the overlap or De-Bruijn graph based methods. The method is largely parameter free and shows promising results on variety of draft bacterial and human chromosome assemblies outperforming other gap filling tools on maximum occasions which we have demonstrated in the Section 5. The quality and robustness of our proposed method was assessed on real data from the GAGE project and the analyses included a comprehensive comparison among our method and other state-of-the-art gap filling tools based on six different metrics such as misassembly, erroneous length, number of gaps remaining etc. evaluated using QUAST. Results from the experiment clearly shows that Figbird has managed to achieve a balanced result across different assembly pipelines, managing to close a larger number of gaps improving the overall NGA50 value while still maintaining to keep the amount of misassembly and length of erroneously introduced sequence lower in most cases.

Among many practical applications of gap filling, variant analysis [96] is one worthy of mentioning. Such an application is presented here [26], where they have presented a one-to-one correspondence between the hardest structural variant search problem of genotyping insertion variants and gap filling in genome assembly. Such an example shows how gap filling problem can not only reach towards a complete genome but also achieve better genomic analysis via more information captured through this newly filled sequences.

6.2 Future works

One of the two possible extensions of the current methodology can be the modification of the method to incorporate long read sequences from the TGS technologies such as PacBio or ONT. Long reads will definitely span those large gaps where SGS reads won't map due to significantly low insert sizes. Our strategy of using learnt distributions will not only help to solve the issues of repeat related gaps but also the error model can solve the issues of misassembly and high error rates associated with long reads. But to do so, we will have to change our sequencing error distributions accordingly. The second possible change that can yield a better gap filling output will be to split the reads into k-mers where parameter k has to be identified through an evaluation of the read library. When reads are longer in length (approximately > 70 bp), there are possibilities of a burst error occurring through the read at any position. Thus, they read may not get aligned properly by bowtie2 currently and eventually getting ignored at current state. But as our entire methodology is based on inset size of a read pair, we will have to come to a solution to the definition and application of insert size in case the reads get split into multiple k-mers and change the corresponding insert size distribution parameters implemented in CGAL [37].

6.3 Availability

Figbird is publicly available at <https://github.com/SumitTarafter/Figbird>, with a user manual and all necessary dependencies.

References

- [1] M. Brbić, M. Piškorec, V. Vidulin, A. Kriško, T. Šmuc, and F. Supek, “The landscape of microbial phenotypic traits and associated genes,” *Nucleic Acids Research*, p. gkw964, 2016.
- [2] J. Lamb, E. D. Crawford, D. Peck, J. W. Modell, I. C. Blat, M. J. Wrobel, J. Lerner, J.-P. Brunet, A. Subramanian, K. N. Ross, et al., “The Connectivity Map: using gene-expression signatures to connect small molecules, genes, and disease,” *Science*, vol. 313, no. 5795, pp. 1929–1935, 2006.
- [3] J. Shendure and H. Ji, “Next-generation DNA sequencing,” *Nature Biotechnology*, vol. 26, no. 10, pp. 1135–1145, 2008.
- [4] F. Sanger, S. Nicklen, and A. R. Coulson, “DNA sequencing with chain-terminating inhibitors,” *Proceedings of the national academy of sciences*, vol. 74, no. 12, pp. 5463–5467, 1977.
- [5] A. M. Maxam and W. Gilbert, “A new method for sequencing DNA,” *Proceedings of the National Academy of Sciences*, vol. 74, no. 2, pp. 560–564, 1977.
- [6] C. S. Pareek, R. Smoczynski, and A. Tretyn, “Sequencing technologies and genome sequencing,” *Journal of Applied Genetics*, vol. 52, no. 4, pp. 413–435, 2011.
- [7] Z. Qiang-long, L. Shi, G. Peng, and L. Fei-shi, “High-throughput sequencing technology and its application,” *Journal of Northeast Agricultural University (English Edition)*, vol. 21, no. 3, pp. 84–96, 2014.
- [8] J. K. Kulski, “Next-generation sequencing—an overview of the history, tools, and “omic” applications,” *Next generation sequencing-advances, applications and challenges*, pp. 3–60, 2016.
- [9] D. A. Wheeler, M. Srinivasan, M. Egholm, Y. Shen, L. Chen, A. McGuire, W. He, Y.-J. Chen, V. Makhijani, G. T. Roth, et al., “The complete genome of an individual by massively parallel DNA sequencing,” *Nature*, vol. 452, no. 7189, pp. 872–876, 2008.

- [10] M. Kchouk, J.-F. Gibrat, and M. Elloumi, “Generations of sequencing technologies: from first to next generation,” *Biology and Medicine*, vol. 9, no. 3, 2017.
- [11] A. J. de Koning, W. Gu, T. A. Castoe, M. A. Batzer, and D. D. Pollock, “Repetitive elements may comprise over two-thirds of the human genome,” *PLOS Genetics*, vol. 7, no. 12, p. e1002384, 2011.
- [12] D. R. Bentley, S. Balasubramanian, H. P. Swerdlow, G. P. Smith, J. Milton, C. G. Brown, K. P. Hall, D. J. Evers, C. L. Barnes, H. R. Bignell, et al., “Accurate whole human genome sequencing using reversible terminator chemistry,” *Nature*, vol. 456, no. 7218, pp. 53–59, 2008.
- [13] S. Koren, M. C. Schatz, B. P. Walenz, J. Martin, J. T. Howard, G. Ganapathy, Z. Wang, D. A. Rasko, W. R. McCombie, E. D. Jarvis, et al., “Hybrid error correction and de novo assembly of single-molecule sequencing reads,” *Nature Biotechnology*, vol. 30, no. 7, pp. 693–700, 2012.
- [14] S. Ardui, A. Ameer, J. R. Vermeesch, and M. S. Hestand, “Single molecule real-time (SMRT) sequencing comes of age: applications and utilities for medical diagnostics,” *Nucleic Acids Research*, vol. 46, no. 5, pp. 2159–2168, 2018.
- [15] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, “ABYSS: a parallel assembler for short read sequence data,” *Genome Research*, vol. 19, no. 6, pp. 1117–1123, 2009.
- [16] D. R. Zerbino and E. Birney, “Velvet: algorithms for de novo short read assembly using de Bruijn graphs,” *Genome Research*, vol. 18, no. 5, pp. 821–829, 2008.
- [17] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe, “ALLPATHS: de novo assembly of whole-genome shotgun microreads,” *Genome Research*, vol. 18, no. 5, pp. 810–820, 2008.
- [18] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, et al., “SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing,” *Journal of Computational Biology*, vol. 19, no. 5, pp. 455–477, 2012.
- [19] M. Li, Z. Liao, Y. He, J. Wang, J. Luo, and Y. Pan, “ISEA: Iterative seed-extension algorithm for de novo assembly using paired-end information and insert size distribution,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 14, no. 4, pp. 916–925, 2016.

- [20] J. Luo, J. Wang, Z. Zhang, F.-X. Wu, M. Li, and Y. Pan, “EPGA: de novo assembly using the distributions of reads and insert size,” *Bioinformatics*, vol. 31, no. 6, pp. 825–833, 2015.
- [21] J. Luo, J. Wang, W. Li, Z. Zhang, F.-X. Wu, M. Li, and Y. Pan, “EPGA2: memory-efficient de novo assembler,” *Bioinformatics*, vol. 31, no. 24, pp. 3988–3990, 2015.
- [22] A. Dayarian, T. P. Michael, and A. M. Sengupta, “SOPRA: Scaffolding algorithm for paired reads via statistical optimization,” *BMC Bioinformatics*, vol. 11, no. 1, pp. 1–21, 2010.
- [23] M. Boetzer, C. V. Henkel, H. J. Jansen, D. Butler, and W. Pirovano, “Scaffolding pre-assembled contigs using SSPACE,” *Bioinformatics*, vol. 27, no. 4, pp. 578–579, 2011.
- [24] T. J. Treangen and S. L. Salzberg, “Repetitive DNA and next-generation sequencing: computational challenges and solutions,” *Nature Reviews Genetics*, vol. 13, no. 1, pp. 36–46, 2012.
- [25] J. R. Miller, S. Koren, and G. Sutton, “Assembly algorithms for next-generation sequencing data,” *Genomics*, vol. 95, no. 6, pp. 315–327, 2010.
- [26] R. Walve, L. Salmela, and V. Mäkinen, “Variant genotyping with gap filling,” *PLOS ONE*, vol. 12, no. 9, p. e0184608, 2017.
- [27] M. J. Chaisson, R. K. Wilson, and E. E. Eichler, “Genetic variation and the de novo assembly of human genomes,” *Nature Reviews Genetics*, vol. 16, no. 11, pp. 627–640, 2015.
- [28] M. Prudencio, P. K. Gonzales, C. N. Cook, T. F. Gendron, L. M. Daugherty, Y. Song, M. T. Ebbert, M. Van Blitterswijk, Y.-J. Zhang, K. Jansen-West, et al., “Repetitive element transcripts are elevated in the brain of C9orf72 ALS/FTLD patients,” *Human Molecular Genetics*, vol. 26, no. 17, pp. 3421–3431, 2017.
- [29] P. Johnsson, L. Lipovich, D. Grandér, and K. V. Morris, “Evolutionary conservation of long non-coding RNAs; sequence, structure, function,” *Biochimica et Biophysica Acta (BBA)-General Subjects*, vol. 1840, no. 3, pp. 1063–1071, 2014.
- [30] J. L. Argueso, J. Westmoreland, P. A. Mieczkowski, M. Gawel, T. D. Petes, and M. A. Resnick, “Double-strand breaks associated with repetitive DNA can reshape the genome,” *Proceedings of the National Academy of Sciences*, vol. 105, no. 33, pp. 11845–11850, 2008.

- [31] S. Raffaele and S. Kamoun, “Genome evolution in filamentous plant pathogens: why bigger can be better,” *Nature Reviews Microbiology*, vol. 10, no. 6, pp. 417–430, 2012.
- [32] R. de Jonge, M. D. Bolton, A. Kombrink, G. C. van den Berg, K. A. Yadeta, and B. P. Thomma, “Extensive chromosomal reshuffling drives evolution of virulence in an asexual pathogen,” *Genome Research*, vol. 23, no. 8, pp. 1271–1282, 2013.
- [33] B. P. Thomma, M. F. Seidl, X. Shi-Kunne, D. E. Cook, M. D. Bolton, J. A. van Kan, and L. Faino, “Mind the gap; seven reasons to close fragmented genome assemblies,” *Fungal Genetics and Biology*, vol. 90, pp. 24–30, 2016.
- [34] D. Domanska, C. Kanduri, B. Simovski, and G. K. Sandve, “Mind the gaps: overlooking inaccessible regions confounds statistical testing in genome analysis,” *BMC bioinformatics*, vol. 19, no. 1, pp. 1–9, 2018.
- [35] M. Meyer and M. Kircher, “Illumina sequencing library preparation for highly multiplexed target capture and sequencing,” *Cold Spring Harbor Protocols*, vol. 2010, no. 6, pp. pdb-prot5448, 2010.
- [36] J. A. Frank, Y. Pan, A. Tooming-Klunderud, V. G. Eijsink, A. C. McHardy, A. J. Nederbragt, and P. B. Pope, “Improved metagenome assemblies and taxonomic binning using long-read circular consensus sequence data,” *Scientific Reports*, vol. 6, no. 1, pp. 1–10, 2016.
- [37] A. Rahman and L. Pachter, “CGAL: computing genome assembly likelihoods,” *Genome Biology*, vol. 14, no. 1, p. R8, 2013.
- [38] A. Rahman and L. Pachter, “SWALO: scaffolding with assembly likelihood optimization,” *Nucleic Acids Research*, vol. 49, pp. e117–e117, 08 2021.
- [39] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *Journal of the Royal Statistical Society. Series B (methodological)*, pp. 1–38, 1977.
- [40] A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler, “QUAST: quality assessment tool for genome assemblies,” *Bioinformatics*, vol. 29, no. 8, pp. 1072–1075, 2013.
- [41] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts, et al., “GAGE: A critical evaluation of genome assemblies and assembly algorithms,” *Genome Research*, vol. 22, no. 3, pp. 557–567, 2012.

- [42] V. C. Piro, H. Faoro, V. A. Weiss, M. B. Steffens, F. O. Pedrosa, E. M. Souza, and R. T. Raittz, “FGAP: an automated gap closing tool,” *BMC Research Notes*, vol. 7, no. 1, pp. 1–5, 2014.
- [43] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs,” *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.
- [44] S. Kosugi, H. Hirakawa, and S. Tabata, “GMcloser: closing gaps in assemblies accurately with a likelihood-based selection of contig or long-read alignments,” *Bioinformatics*, vol. 31, no. 23, pp. 3733–3741, 2015.
- [45] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. L. Salzberg, “Versatile and open software for comparing large genomes,” *Genome Biology*, vol. 5, no. 2, pp. 1–9, 2004.
- [46] S. Kosugi, S. Natsume, K. Yoshida, D. MacLean, L. Cano, S. Kamoun, and R. Terauchi, “Coval: improving alignment quality and variant calling accuracy for next-generation sequencing data,” *PLOS ONE*, vol. 8, no. 10, p. e75402, 2013.
- [47] L. Noé and G. Kucherov, “YASS: enhancing the sensitivity of DNA similarity search,” *Nucleic Acids Research*, vol. 33, no. suppl_2, pp. W540–W543, 2005.
- [48] A. C. English, S. Richards, Y. Han, M. Wang, V. Vee, J. Qu, X. Qin, D. M. Muzny, J. G. Reid, K. C. Worley, et al., “Mind the gap: upgrading genomes with Pacific Biosciences RS long-read sequencing technology,” *PLOS ONE*, vol. 7, no. 11, p. e47768, 2012.
- [49] “Pacific biosciences.” <http://www.pacificbiosciences.com/>. [Online; accessed 20-July-2021].
- [50] J. I. Kammonen, O.-P. Smolander, L. Paulin, P. A. Pereira, P. Laine, P. Koskinen, J. Jernvall, and P. Auvinen, “gapFinisher: A reliable gap filling pipeline for SSPACE-LongRead scaffolder output,” *PLOS ONE*, vol. 14, no. 9, p. e0216885, 2019.
- [51] M. Boetzer and W. Pirovano, “SSPACE-LongRead: scaffolding bacterial draft genomes using long read sequence information,” *BMC Bioinformatics*, vol. 15, no. 1, pp. 1–9, 2014.
- [52] T. Christiansen, L. Wall, J. Orwant, et al., *Programming Perl: Unmatched power for text processing and scripting.* ” O’Reilly Media, Inc.”, 2012.

- [53] P. H. de Sa, F. Miranda, A. Veras, D. M. de Melo, S. Soares, K. Pinheiro, L. Guimarães, V. Azevedo, A. Silva, and R. T. Ramos, “GapBlaster—a graphical gap filler for prokaryote genomes,” *PLOS ONE*, vol. 11, no. 5, p. e0155327, 2016.
- [54] R. T. J. Ramos, A. R. Carneiro, P. H. Caracciolo, V. Azevedo, M. P. C. Schneider, D. Barh, and A. Silva, “Graphical contig analyzer for all sequencing platforms (G4ALL): a new stand-alone tool for finishing and draft generation of bacterial genomes,” *Bioinformatics*, vol. 9, no. 11, p. 599, 2013.
- [55] X. Huang, S.-P. Yang, A. T. Chinwalla, L. W. Hillier, P. Minx, E. R. Mardis, and R. K. Wilson, “Application of a superword array in genome assembly,” *Nucleic Acids Research*, vol. 34, no. 1, pp. 201–205, 2006.
- [56] S.-H. Lin and Y.-C. Liao, “CISA: contig integrator for sequence assembly of bacterial genomes,” *PLOS ONE*, vol. 8, no. 3, p. e60843, 2013.
- [57] D. D. Sommer, A. L. Delcher, S. L. Salzberg, and M. Pop, “Minimus: a fast, lightweight genome assembler,” *BMC Bioinformatics*, vol. 8, no. 1, pp. 1–11, 2007.
- [58] J. Luo, J. Wang, J. Shang, H. Luo, M. Li, F.-X. Wu, and Y. Pan, “GapReduce: A gap filling algorithm based on partitioned read sets,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 17, no. 3, pp. 877–886, 2018.
- [59] I. J. Tsai, T. D. Otto, and M. Berriman, “Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps,” *Genome Biology*, vol. 11, no. 4, pp. 1–9, 2010.
- [60] M. Boetzer and W. Pirovano, “Toward almost closed genomes with GapFiller,” *Genome Biology*, vol. 13, no. 6, p. R56, 2012.
- [61] S. Gao, D. Bertrand, and N. Nagarajan, “FinIS: improved in silico finishing using an exact quadratic programming formulation,” in *International Workshop on Algorithms in Bioinformatics*, pp. 314–325, Springer, 2012.
- [62] X. Yang, D. Medvin, G. Narasimhan, D. Yoder-Himes, and S. Lory, “CloG: a pipeline for closing gaps in a draft assembly using short reads,” in *2011 IEEE 1st International Conference on Computational Advances in Bio and Medical Sciences (ICCBMS)*, pp. 202–207, IEEE, 2011.
- [63] C. Chu, X. Li, and Y. Wu, “GAPPadder: a sensitive approach for closing gaps on draft genomes with short sequence reads,” *BMC Genomics*, vol. 20, no. 5, pp. 1–10, 2019.

- [64] R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, G. He, Y. Chen, Q. Pan, Y. Liu, et al., “SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler,” *Gigascience*, vol. 1, no. 1, pp. 2047–217X, 2012.
- [65] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, et al., “De novo assembly of human genomes with massively parallel short read sequencing,” *Genome Research*, vol. 20, no. 2, pp. 265–272, 2010.
- [66] D. Paulino, R. L. Warren, B. P. Vandervalk, A. Raymond, S. D. Jackman, and I. Birol, “Sealer: a scalable gap-closing application for finishing draft genomes,” *BMC Bioinformatics*, vol. 16, no. 1, pp. 1–8, 2015.
- [67] L. Salmela, K. Sahlin, V. Mäkinen, and A. I. Tomescu, “Gap Filling as Exact Path Length Problem,” *Journal of Computational Biology*, vol. 23, no. 5, pp. 347–361, 2016.
- [68] J. Wetzel, C. Kingsford, and M. Pop, “Assessing the benefits of using mate-pairs to resolve repeats in de novo short-read prokaryotic assemblies,” *BMC Bioinformatics*, vol. 12, no. 1, pp. 1–14, 2011.
- [69] M. Nykänen and E. Ukkonen, “The exact path length problem,” *Journal of Algorithms*, vol. 42, no. 1, pp. 41–53, 2002.
- [70] B. P. Vandervalk, S. D. Jackman, A. Raymond, H. Mohamadi, C. Yang, D. A. Attali, J. Chu, R. L. Warren, and I. Birol, “Konnector: Connecting paired-end reads using a bloom filter de Bruijn graph,” in *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 51–58, IEEE, 2014.
- [71] C. Ye, Z. S. Ma, C. H. Cannon, M. Pop, and W. Y. Douglas, “Exploiting sparseness in de novo genome assembly,” in *BMC Bioinformatics*, vol. 13, pp. 1–8, BioMed Central, 2012.
- [72] S. Chhangawala, G. Rudy, C. E. Mason, and J. A. Rosenfeld, “The impact of read length on quantification of differentially expressed genes and splice junction detection,” *Genome Biology*, vol. 16, no. 1, p. 131, 2015.
- [73] A. Conesa, P. Madrigal, S. Tarazona, D. Gomez-Cabrero, A. Cervera, A. McPherson, M. W. Szczesniak, D. J. Gaffney, L. L. Elo, X. Zhang, et al., “A survey of best practices for RNA-seq data analysis,” *Genome Biology*, vol. 17, no. 1, p. 13, 2016.

- [74] I. Vasilinetc, A. D. Prjibelski, A. Gurevich, A. Korobeynikov, and P. A. Pevzner, “Assembling short reads from jumping libraries with large insert sizes,” *Bioinformatics*, vol. 31, no. 20, pp. 3262–3268, 2015.
- [75] M. E. Talkowski, C. Ernst, A. Heilbut, C. Chiang, C. Hanscom, A. Lindgren, A. Kirby, S. Liu, B. Muddukrishna, T. K. Ohsumi, et al., “Next-generation sequencing strategies enable routine detection of balanced chromosome rearrangements for clinical diagnostics and genetic research,” *The American Journal of Human Genetics*, vol. 88, no. 4, pp. 469–481, 2011.
- [76] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, “Ultrafast and memory-efficient alignment of short DNA sequences to the human genome,” *Genome Biology*, vol. 10, no. 3, pp. 1–10, 2009.
- [77] F. García-Alcalde, K. Okonechnikov, J. Carbonell, L. M. Cruz, S. Götz, S. Tarazona, J. Dopazo, T. F. Meyer, and A. Conesa, “Qualimap: evaluating next-generation sequencing alignment data,” *Bioinformatics*, vol. 28, no. 20, pp. 2678–2679, 2012.
- [78] “Expected errors predicted by Phred (Q) scores.” <https://drive5.com/usearch/manual/readqualfiltering.html>. [Online; accessed 24-July-2021].
- [79] “Expected errors predicted by Phred (Q) scores.” https://drive5.com/usearch/manual/exp_errs.html. [Online; accessed 24-July-2021].
- [80] R. C. Edgar and H. Flyvbjerg, “Error filtering, pair assembly and error correction for next-generation sequencing reads,” *Bioinformatics*, vol. 31, no. 21, pp. 3476–3482, 2015.
- [81] “N50 statistics.” <https://sites.google.com/site/wiki4metagenomics/pdf/definition/assembly/n50>. [Online; accessed 24-July-2021].
- [82] G. I. S. Consortium et al., “Initial sequencing and analysis of the human genome,” *Nature*, vol. 409, no. 6822, pp. 860–921, 2001.
- [83] “Why is N50 used as an assembly metric (and what’s the deal with NG50)?” <http://www.acgt.me/blog/2013/7/8/why-is-n50-used-as-an-assembly-metric.html>. [Online; accessed 24-July-2021].
- [84] D. Earl, K. Bradnam, J. S. John, A. Darling, D. Lin, J. Fass, H. O. K. Yu, V. Buffalo, D. R. Zerbino, M. Diekhans, et al., “Assemblathon 1: a competitive assessment of de novo short read assembly methods,” *Genome Research*, vol. 21, no. 12, pp. 2224–2241, 2011.

- [85] V. Mäkinen, L. Salmela, and J. Ylinen, “Normalized n50 assembly metric using gap-restricted co-linear chaining,” *BMC Bioinformatics*, vol. 13, no. 1, pp. 1–5, 2012.
- [86] E. S. Lander and M. S. Waterman, “Genomic mapping by fingerprinting random clones: a mathematical analysis,” *Genomics*, vol. 2, no. 3, pp. 231–239, 1988.
- [87] Wikipedia contributors, “Maximum likelihood estimation — Wikipedia, the free encyclopedia,” 2019. [Online; accessed 22-July-2021].
- [88] B. Langmead and S. L. Salzberg, “Fast gapped-read alignment with Bowtie 2,” *Nature Methods*, vol. 9, no. 4, p. 357, 2012.
- [89] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, “The sequence alignment/map format and SAMtools,” *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.
- [90] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [91] J. T. Simpson and R. Durbin, “Efficient construction of an assembly string graph using the FM-index,” *Bioinformatics*, vol. 26, no. 12, pp. i367–i373, 2010.
- [92] D. R. Kelley, M. C. Schatz, and S. L. Salzberg, “Quake: quality-aware detection and correction of sequencing errors,” *Genome Biology*, vol. 11, no. 11, p. R116, 2010.
- [93] M. S. Fujimoto, P. M. Bodily, N. Okuda, M. J. Clement, and Q. Snell, “Effects of error-correction of heterozygous next-generation sequencing data,” *BMC Bioinformatics*, vol. 15, no. 7, pp. 1–8, 2014.
- [94] H. Li and R. Durbin, “Fast and accurate short read alignment with Burrows–Wheeler transform,” *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.
- [95] M. Hunt, C. Newbold, M. Berriman, and T. D. Otto, “A comprehensive evaluation of assembly scaffolding tools,” *Genome Biology*, vol. 15, no. 3, pp. 1–15, 2014.
- [96] S. Pabinger, A. Dander, M. Fischer, R. Snajder, M. Sperk, M. Efremova, B. Krabichler, M. R. Speicher, J. Zschocke, and Z. Trajanoski, “A survey of tools for variant analysis of next-generation genome sequencing data,” *Briefings in Bioinformatics*, vol. 15, no. 2, pp. 256–278, 2014.

Generated using Postgraduate Thesis L^AT_EX Template, Version 1.02. Department of
Computer Science and Engineering, Bangladesh University of Engineering and
Technology, Dhaka, Bangladesh.

This thesis was generated on Saturday 26th February, 2022 at 2:28pm.