Novel graph based algorithms for transcriptome sequence analysis

by
Dilip Ariyur DURAI

A dissertation submitted towards the degree
Doctor of Natural Sciences (Dr. rer. nat)
of the Faculty of Mathematics and Computer Science
of Saarland University

Saarbrücken, 2020

Tag des Kolloquiums:	June 2, 2021
Dekan:	Prof. Dr. Thomas Schuster
Prüfungsausschuss:	
Vorsitzende:	Prof. Dr. Kurt Mehlhorn
Berichtserstatter:	Prof. Dr. Marcel Schulz
	Prof. Dr. Volkhard Helms
Akademischer Mitarbeiter:	Dr. Pratiti Bhadra

"There is always a simple solution to the most complicated problems. All you need to do is consider it."

SAARLAND UNIVERSITY

Abstract

Mathematics and Informatics
Department of Computer Science

Doctor rerum naturalium

Novel graph based algorithms for transcriptome sequence analysis

by Dilip Ariyur DURAI

RNA-sequencing (RNA-seq) is one of the most-widely used techniques in molecular biology. A key bioinformatics task in any RNA-seq workflow is the assembling the reads. As the size of transcriptomics data sets is constantly increasing, scalable and accurate assembly approaches have to be developed.Here, we propose several approaches to improve assembling of RNA-seq data generated by second-generation sequencing technologies. We demonstrated that the systematic removal of irrelevant reads from a high coverage dataset prior to assembly, reduces runtime and improves the quality of the assembly. Further, we propose a novel RNA-seq assembly workflow comprised of read error correction, normalization, assembly with informed parameter selection and transcript-level expression computation.

In recent years, the popularity of third-generation sequencing technologies increased as long reads allow for accurate isoform quantification and gene-fusion detection, which is essential for biomedical research. We present a sequence-to-graph alignment method to detect and to quantify transcripts for third-generation sequencing data. Also,we propose the first gene-fusion prediction tool which is specifically tailored towards long-read data and hence achieves accurate expression estimation even on complex data sets. Moreover, our method predicted experimentally verified fusion events along with some novel events, which can be validated in the future.

SAARLAND UNIVERSITY

Kurzfassung

Mathematics and Informatics
Department of Computer Science

Doctor rerum naturalium

Novel graph based algorithms for transcriptome sequence analysis

by Dilip Ariyur DURAI

RNA-sequenzierung (RNA-seq) ist eine weit verbreitete Methode der Molekularbiologie. Eine Schüsselaufgabe kommt dabei auch der Bioinformatik zu, die Assemblierung der Sequenzfragmente. Da die Größe von RNA-seq datensätzen kontinuerlich steigt, ist die Entwicklung von skalierbaren und korrekten Assemblierungsverfahren von großer Bedeutung. Hier präsentieren wir verschiedene Methoden zur Optimierung der Assemblies von RNA-seq daten die mittels Sequenzierungs Technologie zweiter Generation erstellt wurden. Wir haben am Beispiel verschiedener komplexer Datensätze aufgezeigt, dass das systematische Entfernen von unrelevanten Sequenzfragmenten zu einer Verbesserung der Assemblierung und zu einer verkürzung der Laufzeit führt. Außerdem stellen wir einen neuen RNA-seq Ansatz vor, der sich durch Fehlerkorrektur der Sequenzfragmente, Normalisierung, Assemblierung mit systematischer Parameterauswahl und Expressionsquantifierzung auf Transkriptebene auszeichnet.

Sequenzierungstechnologien der dritten Generation finden seit geraumer Zeit vermehrt Verwendung, da diese eine genaue Quantifizrung von Isoformen und Genefusions Ereignissen ermöglichen. Diese Eigenschaften sind essentiel für biomedizinische Anwendungen. Wir haben eine neue Alignierungs- und Quantifierungsmethode für Sequencierungsdaten der dritten Generation entwickelt, welche auf dem sequenz-to-graph Ansatz beruht. Darüber hinaus stellen wir die erste Methode zur Erkennung von Genefusions Ereignissen vor, welche speziell für Sequenzierungsmethoden der dritten Generation entwickelt wurde und daher zu sehr genauen Expressionsquantifizierung führt. Neben einigen neuen Kandidaten, wurde einer Vielzahl von vorhergesagten Genefusions Ereignissen bereits experimentell validiert.

Acknowledgements

Its never a one man show. There were always a sea of people standing rock solid behind me at every stage of my doctorate. Before I list out the people, I want to clarify that just a mere thank you is not enough to acknowledge the immense contribution of these people.

First and foremost a special gratitude goes out to my supervisor Prof. Dr. Marcel H. Schulz who guided me through my Ph.D. journey. I am grateful to him for his ideas, support, initiation of collaborations and more importantly his patience (which I tested a lot of times). I also want to thank Prof. Dr. Tobias Marschall and Mikko Rautiainen for collaborating with me on the Aeron project. It was a very exciting project and I thank them for sharing their knowledge and expertise with us.

A major factor which made this research work conducive was the wonderful environment of the SchulzLab and the computational biology department of the institute. I was extremely lucky to have group members like Fatemeh Behjati, Florian Schmidt and Sivarajan Karunanithi. Pursuing Ph.D so far away from home can be stressful. However, my group including my supervisor gave me the strength and motivation to carry out such an ardous journey. I will miss the scientific discussions, the fun and laughter of our lab, the trips which we went and the group dynamics. I would like to thank the entire Computational Biology group for some great scientific discussions over lunch table and coffee breaks. I would also like to thank them for including me in various activities which were fun and very comforting. Documenting the research work on paper and presenting them in various international conferences and workshops were also important chapters of my journey. I would like to thank Prof. Dr. Marcel Schulz, Prof. Dr. Tobias Marschall and Mikko Rautiainen for co-authoring my research articles. I would like to thank Anna Hake, Fatemeh Behjati, Florian Schmidt, Lisa Handl, Nora K Speicher, Peter Ebert, Prabhav Kalghatgi, Rebecca Serramari, Shilpa Garg, and Sivarajan Karunanithi for listening to my practice talks, testing my software and algorithms, proofreading my papers and proofreading my thesis.

Life outside scientific research is equally important. I am grateful to Kaustuv Chakrabarty and Harini Priyadarshini Hariharan. They have been constantly with me and supported me during all the highs and lows of my life in Germany. I would also like to give a hat tip to Charlotte Sailard, Deepti Mittal, Luigi Giannelli, and Sonja Persch for their important part in my life.

Finally, I would like to thank my parents and my sister for their never ending support, their belief in me and their unconditional love. This thesis wouldnt have been possible without them.

Thank you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter briefly describes the motivation for the work done in this these. A technical background that is of general relevance is provided and an outline of the thesis is given at the end of the chapter.

1.1 Basic building blocks of life: DNA and RNA

All living organisms are made of individual and identifiable *cells*. The number of cells present in a human body is estimated to be around 37.2 trillion ([Bia+13]). The number of major *cell types*, i.e, cells with different morphology is estimated to be around 200-300. During the early 19th century, it was widely accepted that cells arise from the growth and division of other cells. The nucleus of a cell contains a thread-like structure named *chromosome* which is responsible for carrying the hereditary information of a cell. A chromosome consists of half nucleic acid and half protein. In the following sub-sections, we will focus on the molecular structure of the nucleic acids DNA and RNA. We will also touch upon gene expression which should give us a glimpse of how cellular diversity is initiated and maintained in a living organism.

1.1.1 Deoxyribo Nucleic Acid (DNA)

DNA is a macromolecule consisting of the genetic information of the cell. Isolated in the year 1869, the structure of DNA was not determined until 1953. In 1953, Francis Crick and James Watson proposed the structure based on X-Ray diffraction studies([WC53]) which is now widely accepted. The DNA is composed of four basic molecules called nucleotides. The nucleotides further consists of five-carbon sugar called *deoxyribose*, a phosphate group, and a nitrogenous base which can be either *Adenine*, *Thymine*, *Cytosine* or *Guanine*. The pair of Adenine and Guanine are classified as *Purines* and the pair of Cytosine and Thymine are considered as *Pyrimidines*. Purines consist of a double ring structure where a six-carbon ring is fused into a five carbon-ring whereas *Pyrimidines* consist only of a single six-carbon ring structure. The carbon of the carbon sugar is numbered 1′ through 5′. The nucleotide base is connected to the 1′ carbon of the sugar. The phosphate group connected to the 5′ carbon of the sugar of one nucleotide bonds with the hydroxyl group of the 3′ carbon of the sugar of the other nucleotide molecule and hence connecting the two molecules. A sequence of such connections forms a strand of a DNA where the sugar and the phosphate form the backbone of the strand.

Figure 1.1 depicts the orientation and structure of DNA. A DNA molecule consists of two anti-parallel strands twisted in the form of a double helix (fig. 1.1a). The two strands are held together by a purine base of one strand and a pyrimidine bases of another strand by a hydrogen bond (fig. 1.1b). The plane of the bases is

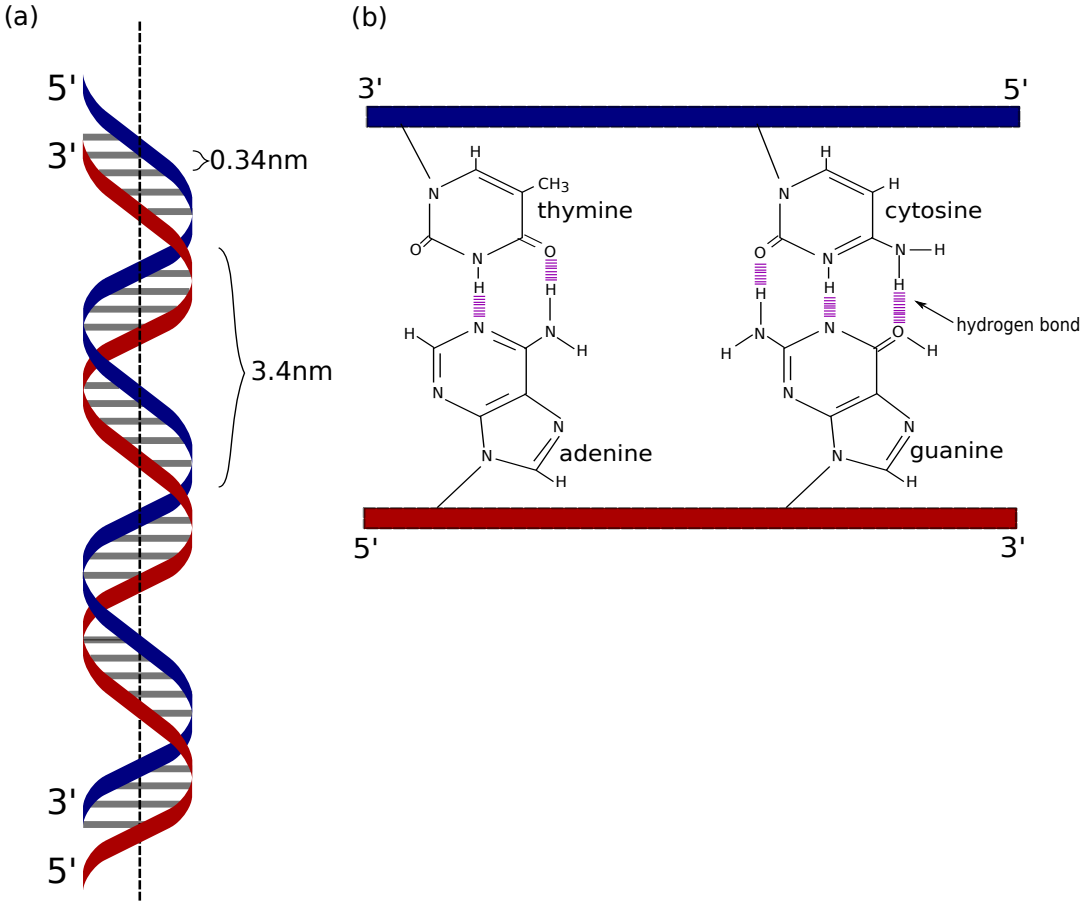

FIGURE 1.1: Structure of a DNA. (a) the double helix structure of a DNA consisting of two anti-parallel DNA strands forming a right-handed double helix. (b) Complementary base pairing within a DNA where adenine always bonds with a thymine and cytosine always bonds with a guanine. The hydrogen bonds between the bases is depicted as pink lines. Figure inspired from [Alb+02]

perpendicular to the axis of the helix. Adenine bonds with Thymine and Cytosine bonds with Guanine. These pairings are known as complementary base pairing. A chromosome consists of such DNA molecules tightly packed around a protein called *histones* which makes up the genetic structure of a species. The human species has 23 pairs of chromosomes which constitute around 3,2 billion nucleotide bases.

1.1.2 Ribo Nucleic Acid (RNA)

Similar to DNA, Ribo Nucleic Acid (RNA) are macromolecules consisting of chains of nucleotides. Although there are multiple types of RNA molecules, all of them have a similar basic structure made by adding the 5′-phosphate group of one nucleotide onto the 3′ hydroxyl group of the previous nucleotide in the chain. Like DNA, an RNA strand is composed of nitrogenous bases covalently bound to a sugar-phosphate backbone. But unlike DNA, RNA is single-stranded and consists of *ribose* instead of deoxyribose. Also, an unmethylated form of thymine known as uracil

forms the complementary base of adenine. Another major difference in the structure of RNA, as compared to that of a DNA, is the presence of a hydroxyl group attached to the 2′ position of the pentose sugar ring which makes RNA highly unstable in nature ([Sal+93]). The secondary structures of RNA involves folding of single RNA molecule to form hairpin loops. The loops are stabilized by intramolecular hydrogen bonds between complementary bases. Such secondary structures are critical for many RNA functions such as the ability of a tRNA to bind to the correct sequence of mRNA during translation.

1.1.3 Gene Expression

An important functional segment of a DNA is *gene*. According to the central dogma of molecular biology, a genomic DNA is transcribed into a mature RNA (mRNA). The mRNA gets either translated into a functional protein or remain as a non-coding RNA. The strands of a genomic DNA are labeled as "Template Strand" and "Coding strand". The template strand is used during the process of transcription to form RNA sequence.

Transcription

The process of transcription is initiated by binding of the enzyme RNA-polymerase, along with transcription factors, to a region of DNA sequences known as *promoter region*. This complex unwinds the DNA to form a single-stranded region of DNA known as the *transcription bubble*. RNA-polymerase along with transcription factors then binds to the *transcription start site* present in the bubble and starts synthesizing the RNA. The RNA-polymerase complements the sequence of the template strand from 3′ end to 5′ end except for the base Thymine(T) which is substituted by Uracil(U) ([Har+00]). In prokaryotes, the resulting RNA strand is directly translated to form functional proteins. However, in eukaryotes, the resulting RNA strand, also known as *pre-mRNA*, goes through enzyme enabled reactions that add the purine nucleoside *7-methylguanosine* to the 5′ end of the strand. If a polyadenylation signal sequence 5′-AAUAAA-3′ is present in the pre-mRNA, then the pre-mRNA is cleaved and a series of adenine(A) is added to the 3′ end of the sequence ([FS81]). Pre-mRNA consists of regions called *introns* and *exons*. Each intron contains a 5′ donor splice site and a 3′ acceptor splice site. An RNA-protein complex known as *spliceosome* cleaves the introns and splices consecutive exons together to form a mature-RNA(mRNA) ([Gil78]).

Figure 1.2 summarizes the two-step mechanism of splicing in eukaryotes. Cleavage of the 5′ donor site of the intron with the help of spliceosome marks the first step of splicing. This step yields a 5′ exon-intermediate with a free 3′ OH residue. The 5′ end of the intron is joined to an adenosine residue within the intron by a 2′-5′ phosphodiester bond (fig.1.2b). The second step comprises of cleavage of the 3′ acceptor site of the intron and splicing of the two exons by a 3′-5′ phosphodiester bond. The intron is released in its lariate form and then degraded (fig.1.2c). The exons splice together to form a mature RNA (mRNA) ([MQS93]). In most cases, the intron is removed as a single unit from pre-mRNA. However, in some cases where the pre-mRNA contains long introns, the intron is removed in parts as a recursive step. This process is known as *recursive splicing* ([Sib+15]).

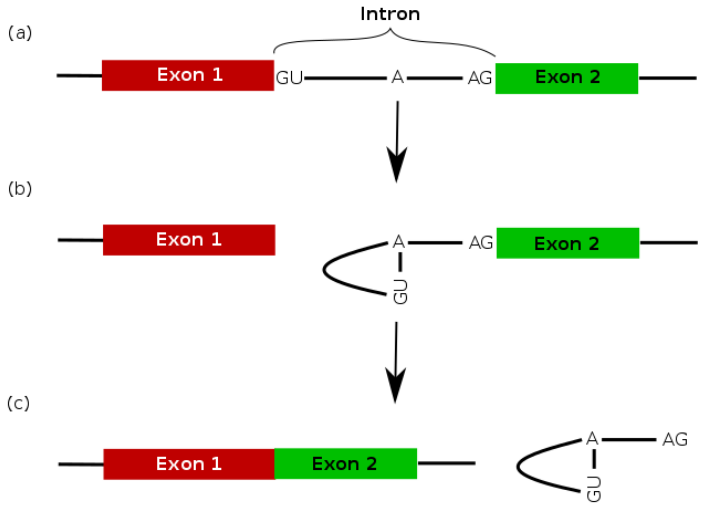

FIGURE 1.2: Intron cleaving and splicing of exons. (a) Each intron is flanked by a GU pair at its 5′ end and an AG pair at its 3′ end. (b) The spliceosome cleaves the 5′ end of the intron and bonds it to an adenosine residue within the intron. (c) The 3′ end of the intron is cleaved and the two exons are joined by a phosphodiester bond. Figure inspired from [MQS93]

Alternative splicing

Before the year 1978, it was believed that one gene can only code for one protein. However in 1978, Gilbert ([Gil78]) proposed the theory of alternative splicing which now explains the discrepancy between the number of protein-coding genes and the number of different proteins generated. The general roadmap for a transcription process comprises of intron removal and exon splicing. Alternative splicing deviated from this roadmap by skipping certain exons or retaining introns. This results in the generation of multiple transcripts of mRNA, also known as isoforms, from a single gene ([BMS77; MCS05]). It has been shown that more than 95% of the multi-exonic genes undergo alternative splicing ([CM09; Bla03; SFG08]). Depending on the region which is cleaved, alternative splicing event is classified into - 1) exon skipping in which an exon is cleaved out of the pre-mRNA, 2) mutually exclusive exons in which only one exon from a pair of exons is retained, 3) intron retention in which intronic sequence between a pair of exons is retained, 4) alternate donor where an alternative donor site is used for splicing the intron and 5) alternate acceptor where an alternate acceptor site is used during the process of splicing ([SFG08]). The mechanism of alternative splicing is highly dependant upon controlling splice site recognition by facilitating or interfering with the binding of spliceosome complex with the splice site ([CM09]). This is achieved by the activity of RNA sequence elements and protein regulators within an exon such as exonic splicing enhancers (ESE), exonic splicing silencers (ESS), intronic splicing enhancers (ISE) and intronic splicing silencers (ISS). ESEs are usually bound to SR proteins (family containing a long stretch of Serines and Arginines) which promote exon splicing while ESSs usually bind to heterogeneous nuclear RNPs (hnRNPs) to block exon splicing. Both types of proteins (SR-proteins and hnRNPs) are involved in the assembly and functioning of spliceosomes.

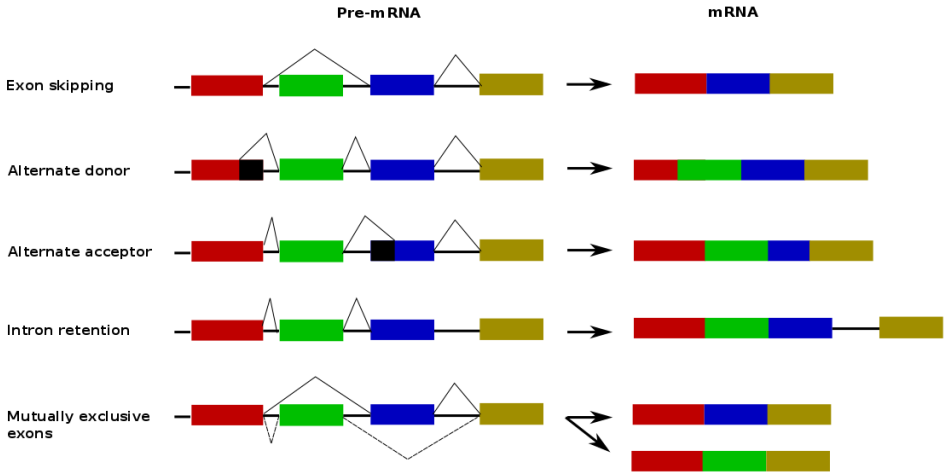

FIGURE 1.3: Different types of alternate splicing events. Figure recreated from [Sen18]

Translation

The mRNA formed during the transcription serves as the base for a ribosome mediated synthesis of one or more proteins. An mRNA consists of three regions - 1) a 5′ untranslated Region (5′UTR), 2) a protein-coding region or Open Reading Frame (ORF) and a 3) 3′ untranslated Region (3′UTR). The process of translation is initiated when a ribosome is assembled around an mRNA. The ribosome consists of two sub-units namely a 60S subunit and a smaller 40s sub-unit. Organic molecules containing amine and acidic carboxylic groups, also known as amino acids, are transferred to the ribosome by transfer RNA (tRNA). The 40S subunit of the ribosome positions the mRNA such that it can be read in groups of three bases (known as codons) at a time. Each codon matches a complementary triplet in the tRNA known as an anticodon. Each ribosome consists of three sites namely - an aminoacyl site (A-site), a peptidyl site (P-site) and an exit site (E-site). The tRNA first goes into the A-site where the codon and the anticodon are checked for a match. If matched, the tRNA is forwarded to the P-site where the 60S sub-unit removes the amino acid from the matched tRNA and joins it to a growing amino acid chain. The remaining tRNA is then transferred to the E-site and ejected from the ribosome. The linear chain of amino acid folds according to the sequence of amino acid to form a 3-dimensional protein structure.

1.2 Methods for measuring gene expression

In the previous section, we looked into the flow of information from DNA to RNA to protein. Proteins are responsible for the functionality of a cell. Hence, the amount of genes expressed in a cell dictates what a cell can do and what it cant. We also noted that, due to alternative splicing, a single mRNA can encode for multiple proteins. Hence, it can be deduced that the primary control point of gene expression is usually the initiation of transcription. The amount of mRNA present in the cell can give an accurate picture of the functionality of the cell. Measuring gene-expression

by measuring mRNA levels is an important step towards understanding cellular activity especially in a disease infected cell. In the coming sub-sections, we will look into two major strategies to measure gene expression namely DNA microarray and RNA-seq.

1.2.1 DNA-microarray

DNA microarray is a collection of microscopic DNA spots containing specific DNA sequences attached to a surface in a grid-like structure. These DNA sequences are termed as "probe sequences" and are about 20bps in length. They represent tiny but unique regions of the gene in the genome. Figure 1.4 depicts a typical microarray setup. The target mRNA, whose levels is to be measure, is first reverse transcribed to a more stable cDNA. The cDNA is then labeled with fluorescent dyes and added to the microarray. The cDNA, which is complementary to the probes, binds or hybridizes to the probe and stick to the array. The unbound cDNAs are washed away. The microarray is then scanned using a laser and signals emitted from the dye are detected. The strength of the signal depends upon the amount of target cDNA binding to the probe. Hence, by measuring the strength of the signal and matching the probe to its corresponding gene, the relative gene expression levels can be measured ([Bum13]).

Although efficient and widely used, DNA-microarray suffers from some major drawbacks. Accurate gene-expression estimation is hindered by background hybridization in the probes. Also, the technology is limited to the analysis of the genes for which the probes are designed ([Zha+14]). This makes microarrays unattractive for studying non-model species. In contrast to hybridization-based techniques, sequence-based methods quantify the cDNA by determining their sequence and mapping them to a genome. In the initial years, Sanger sequencing and ESTs were used for this purpose. However, their usage deteriorated due to their low throughput. Tag-based methods such as Serial Analysis of Gene Expression(SAGE) and Cap Analysis of Gene Expression (CAGE) were seen as an alternative. But their popularity soon faded due to the high costs ([WGS09]). In recent years, RNA-seq technologies are considered as a strong replacement for traditional sequencing technologies.

1.2.2 RNA-seq

With the advances in next-generation sequencing (NGS), RNA-seq has become a go-to technology for scientists who wish to sequence the mRNA. RNA-seq has a big advantage over DNA-microarray in terms of studying the fine prints of transcription such as allele-specific profiling, detecting novel transcripts and analyzing splice junction ([Zha+14]).

Library preparation

A general procedure for RNA-seq is initiated by isolating RNA from the tissue and treating it with deoxyribonuclease (DNAase). This reduces the amount of genomic

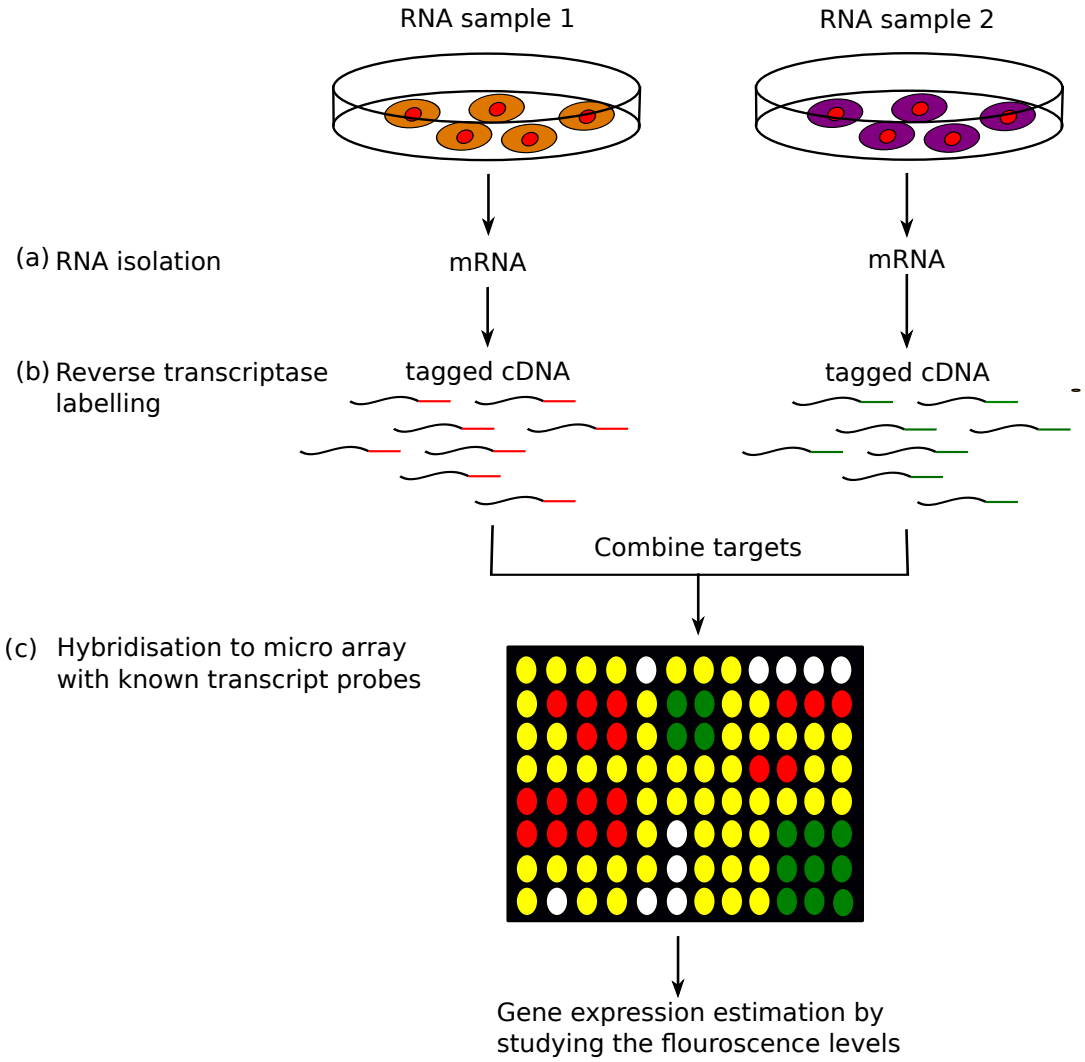

FIGURE 1.4: Workflow of gene expression measurement using DNA microarray. From the given samples, mRNAs are extracted and converted into stable complementary DNA (cDNA). These cDNAs are tagged using fluorescent dyes and are added to a microarray. The microarray is scanned using a laser to measure the strength of the signal from the fluorescent dyes.

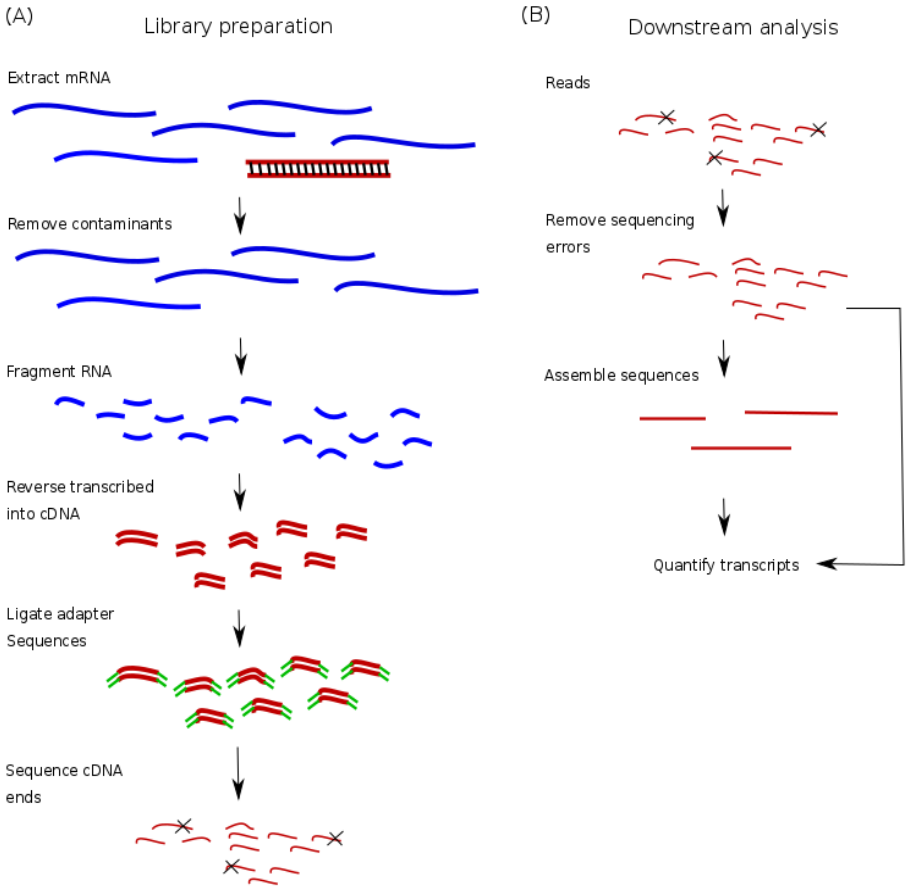

FIGURE 1.5: Workflow of a typical RNA-seq experiment. Given the samples, mRNA is first isolated and then fragemented into smaller pieces. The fragmented mRNA is highly unstable in nature. Hence it is reverse transcribed to form a stable complementary DNA (cDNA). The cDNA is then sequenced using various sequencing technologies. Figure recreated from [MW11]

DNA. The RNA degradation is measured and indicated by RNA Integrity Number (RIN). Figure 1.5 depicts the basic protocol followed by most of the sequencing technologies. To enable sequencing, the long mRNA is size selected and fragmented to smaller pieces of RNA. A single-stranded RNA is highly unstable in nature. Hence, RNA fragments are converted into double-stranded cDNA by the process of reverse transcription. A major challenge before the investigators is whether to amplify the fragments or not. Although PCR amplification increases the number of cDNA molecules that are sequenced, they might introduce errors in the sequence which propagates to later cycles ([Fu+18; CT93]). Recent technologies from Pacific Biosciences and Oxford Nanopore adopt an amplification-free sequencing protocol which has gained prominence. We will be introduced to these technologies later in the chapter.

Sequencing

Sequences are obtained from the fragments in the form of *reads*. Reads are either generated from only one end of the fragment (single-end sequencing) or generated from both the ends in opposite directions (paired end sequencing). The distance between the two adapters, in case of paired end sequencing, is usually represented in terms of fragment length or *insert size*. Reads generated by the above general protocol loose their strand information, i.e, the strand from which the sequence is generated cannot be deduced. However, there are alternate protocols which preserves the information in the read ([Zha+15]).

Sequencing by synthesis
The sequencers from Illumina follow the sequencing-by-synthesis process to generate reads from a sample. The sequencing workflow consists of two major steps which are termed as clustering and sequencing. As mentioned above, the library preparation steps adds adapter sequences to the end of the cDNA fragments. Using a reduced cycle amplification process, additional motifs such as the sequencing binding sites, indices and sequences complementary to the flow cell oligos are added to the cDNA fragments.

In the clustering phase, each fragments is isothermally amplified on a flow cell. Figure 1.6 depicts a workflow for the clustering procedure followed by Illumina sequencing technology. A flow cell is a planer optical transparent surface which is similar to a microscopic slide. Each flow cell consists of lanes and each lane consists of lawns of two types oligonucleotides bound to its surface (fig.1.6a). These oligos are complementary to adapter sequences in a cDNA fragment. One of the oligos is hybridized to the corresponding complementary adapter sequence in the fragment (1.6b). A polymerase creates a complement of the hybridized fragment (fig.1.6c). The double stranded molecule is denatures and the original strain of the fragment is washed away (fig.1.6d). The remaining strand is clonally amplified using bridge amplification. In this process, the strand bends over such that the adapter sequence of the free end hybridizes to the second type of oligonucleotide (fig.1.6e). This forms a bridge like structure between the two types of oligonucleotides. A polymerase then creates the complementary strand from the bridge to form a double stranded bridge (fig.1.6f). The double stranded bridge is denatured to form two strands (forward and reverse strand) of the molecule tethered to the flow cell (fig.1.6g). The process is repeated several times to form millions of such strands attached to the flow cell. After the process, the reverse strand is cleaved and washed off leaving only the forward strand which is termed as the template strand. The 3′ ends of the templates are blocked to prevent unwanted priming.

In the sequencing step, primer attached to the template are extended to produce the first read. Fluorescent labelled nucleotides are introduced to the flow cell which bind to their complementary bases in the template sequence. After each binding, the cluster is excited by a light source and a characteristic fluorescent signal is emitted. The order of the nucleotide bases in the fragment is determined by measuring the strength of these signals. This process is repeated several times to generate millions of reads. Paired end sequencing extends the above procedure to generate reads from the reverse strand of an cDNA fragment. After the generation of the first read, the read product is washed away and the remaining template strand is bend over. The adapter sequence at the free end of the template strand hybridizes to the second type

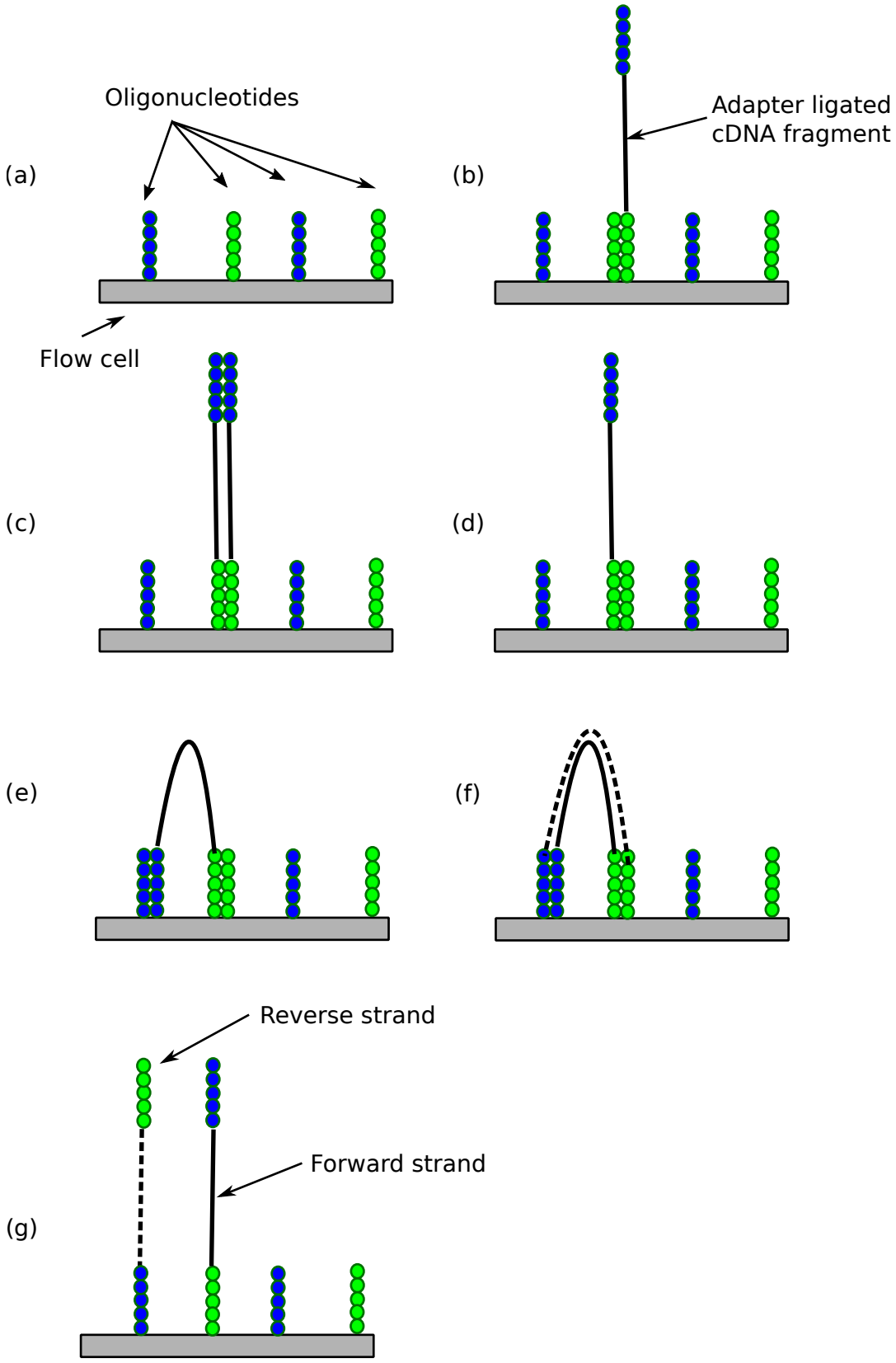

FIGURE 1.6: Clustering process workflow of Illumina sequencing. (a) Flow cell is optical surface consisting of two types oligonucleotides. (b)Adapter sequence at the end of a cDNA fragment hybridizes to one of the oligonucleotide. (c) A polymerase generates the complementary copy of the molecule. (d) The original fragment is washed away. (e) The remaining molecule copy is bent and its other end hybridizes to the second oligonucleotide. (f) A polymerase produces a complementary copy of the bridge. (g) The bridge is denatured to form a forward strand template and a reverse strand template.

of oligonucleotide sequence present in the flow cell to form a bridge. Similar to the single-end sequencing procedure, a polymerase creates the complementary strand of the bridge to form a double stranded bridge. The double stranded bridge is denatured to form a forward strand and reverse strand template. The forward template is now washed away and the read is generated from the reverse template.

The throughput of the Illumina technology is measured in terms of read coverage which the number of unique reads covering any given nucleotide in a reference sequence. Given the length of the reference sequence (G), the number of reads sequenced (N) and the average length of the reads sequences (L), the coverage is calculated using the following formula:

$$cov = NX\frac{L}{G} \tag{1.1}$$

.
For example, if the length of the reference genome is 10,000bps and the sequencer produces 400 reads of length 50bps, then the coverage is calculated to be 2. In scientific terms, we say that the dataset has a 2x coverage. In general, Illumina sequencing technologies have a high throughput. For instance, the Illumina sequencer HiSeq X Ten can produce sequences equivalent to 7 human genomes at 30x coverage daily.

Single Molecule Real Time sequencing (SMRT)
As mentioned above, Illumina sequencing technologies have an advantage of high throughput. However, the read length produced by these technologies are only in the order of 100-500bps. The reads might not be long enough to cover the entire transcripts. This is problematic especially in case of accurate expression estimation or for detection of gene-fusion events. Single Molecule Real Time sequencing (SMRT) from Pacific Biosciences (PacBio) extends the sequencing by synthesis approach by incorporating a real-time imaging of fluorescently tagged nucleotides producing longer and accurate reads ([Eid+09]). Briefly, DNA/cDNA molecule is isolated during the library preparation phase and adapters are ligated to the ends of the strand forming a circular template. Polymerase and primers are added to the template and the entire setup is placed in a sequencer. The sequencer consists of SMRT-cell, similar to flow cell, which consists of millions of tiny wells known as Zero Mode Waveguides (ZMW). As single molecule DNAs are incorporated into these ZMWs, the polymerase begins to incorporate nucleotides in these wells. As each nucleotide is incorporated by the polymerase, a light is emitted from the wells which can be measures in real time to determine the sequence. On an average, SMRT technologies produce read-length in the range of 10-14 kilobases with an average error rate of 15% ([Ard+18]).

Nanopore sequencing
The other technology which produces longer reads is the nanopore sequencing technique. Nanopre sequencing deviates from the fluorescence detection step incorporated in Illumina and PacBio sequencing technologies. First proposed by David Deamer and Daniel Branton ([DB02]), Nanopore sequencing depends upon different ionic burst by nucleotide bases when passes through an ionic channel. Figure 1.7a depicts the general setup of a nanopore sequencing workflow. Nanopore sequencing uses an orifice, also known as nanopore, that is approximately 10^{-9} meters in diameter. It is embeded on a membrane bathed in electrophysiological solution through which an ionic current is be passed. Motor proteins are added to the end

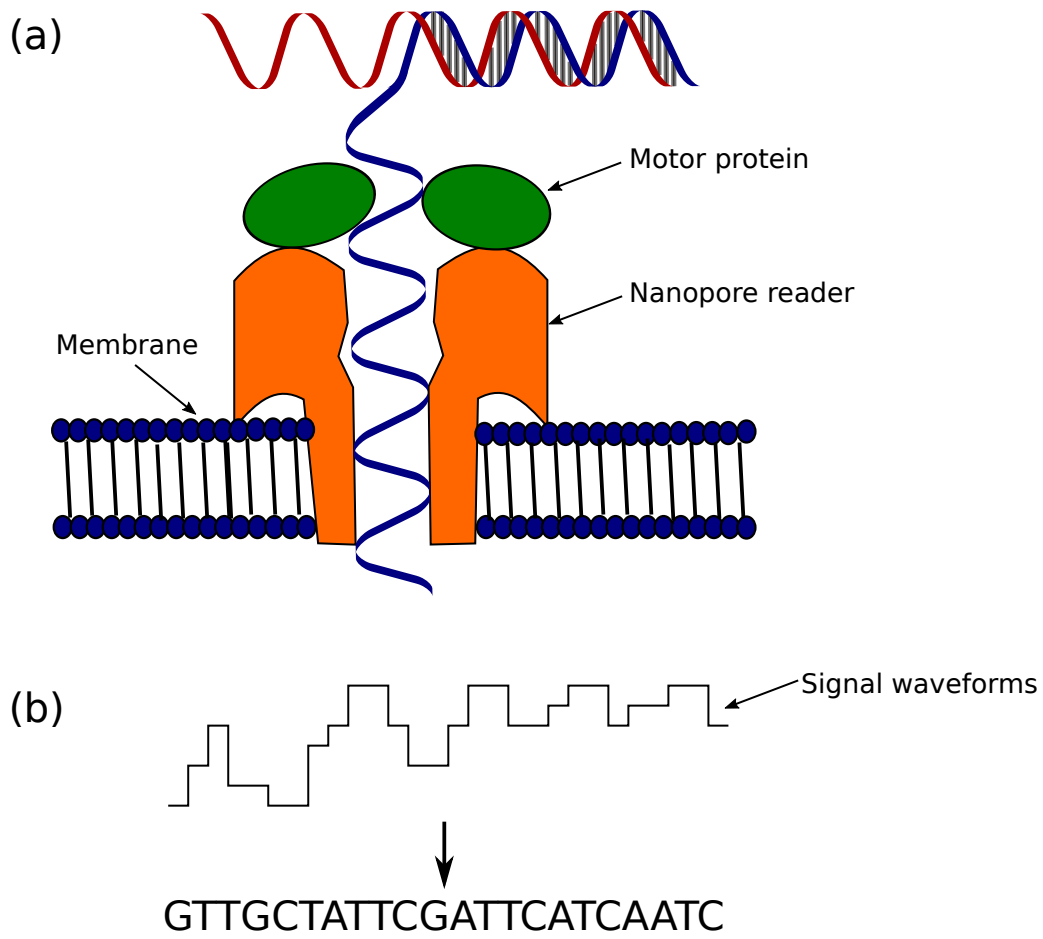

FIGURE 1.7: Workflow of nanopore sequencing. (a) The DNA or the RNA molecule is passed through a nanopore with the help of a motor protein attached to the end of the molecule fragment. The nanopore is attached to a membrane and a steady ionic current passes through it. (b) Passing of nucleotide through the membrane causes a disruption in the electric current which is evaluated to determine the sequence of the fragment.

of DNA or RNA molecules during the library preparation step. These motor proteins allow the DNA or the RNA molecule to pass through the orifice at a certain speed (approx 450bps/second for DNA and 70bps/second for RNA). The motor protein also unzips the double strand of the DNA when it is being feeded to the nanopore. When the molecule passes through the pore, a nanopore reader reads the nucleotides which creates a characteristic disruption in the ionic current passed through the membrane (fig. 1.7b). Each strand is read four bases at a time and the overlap between the signals are evaluated using various base calling algorithms to determine the sequence. ([Sto+09; AB12; Alv+15]).

During sequencing, the nanopore technologies analysis the entire fragment of DNA or RNA presented to it. Hence, the read length produced is directly proportional to the length of the fragment being sequenced. This enables Nanopore sequencers to produce read length upto 100s of kilobases. By varying the sample extraction methods, users can achieve a read length upto 2 megabases. Such read lengths are enough to cover the entire span of transcripts. However, nanopore sequencing suffers from a high error rate (can reach upto 40%) due to inaccurate base calling and noises in the signal waveform ([OWD13; Lav+15]). Designing an effective base calling algorithm for nanopore sequencing is a computational challenge

which is currently in progress.

1.3 Downstream analysis

Using the obtained sequences, investigators design various pipelines depending upon the species they are working with. Reads obtained from sequencers are generally marred by presence of adapter sequences and sequencing errors which might result in faulty downstream analysis. Hence it is always beneficial to remove adapter seqeunces and correct for error using softwares such as CutAdapat ([MW11]) and SEECER ([Le+13]). If the species is well studies and has its genome well annotated, reads can be mapped onto it using aligners such as TOPHAT ([TPS09]), Blat ([Ken02]) or Minimap ([Li16]) to identify and quantify transcripts. In some cases, practioners assemble the reads using state-of-the-art assemblers to identify transcripts. This can be achieved by either mapping the reads onto a reference genome if available and generating contigs based on the mappings or implementing a reference-free *de* novo approach ([Con+16; MW11]). The later one is mostly used when dealing with non-model species. Accurate gene expression can be quantified by feeding either the alignment of reads or the assembly to quantification softwares such as Salmon ([Pat+17b]) or Kallisto ([Bra+16]). Some of the above approaches will be discussed in the next chapter. RNA-seq data can also be combined with functional genomics methods to enhance and strengthen gene expression estimation ([Con+16; Con+05; KTT13]). RNA-seq data can also be couple with various biochemical assays to study RNA-protein or RNA-RNA interactions. These approaches wont be discussed as they are beyond the scope of this thesis.

1.4 Motivation for this work

As mentioned above, transcriptome assembly is one of the key steps in any gene expression analysis using RNA-seq. In recent times, with the reduced cost of sequencing, it is highly recommended to sequence deep as this would ensure a proper coverage of transcripts ([Cah+12; Ung+17; Sze+17]). As mentioned above, in the absence of a well-annotated reference sequence, the reads have to be assembled using a de novo strategy. This has resulted in the development of many state-of-art algorithms such as TranABySS ([Rob+10]), Trinity ([Gra+11]), and Oases ([Sch+12b]).

However, there are issues regarding the de novo transcriptome assembly namely misassemblies due to artificial chimeras, the inability of the assembler to resolve reads originating from repetitive regions, and sensitivity of alignment error due to paralogs ([Til+15; Ung+17; PS+16]). Another major hindrance is the memory and runtime required to assemble high coverage datasets ([Bro+12; DS16; Wed+17]). This problem can be mitigated to a certain extend by considering only the relevant set of reads for the process of assembly. The quality of assembled sequences is also downgraded by inefficient pre-processing of the data and selection of inappropriate assembly parameters. Researchers generally rely on the pre-processing step of the assembler which is not efficient especially for high coverage dataset ([Le+13]). Also, while selecting the parameters for the assembler, researchers either use the default setting of the algorithm or use a random set of parameters. Since, the assemblers are trained on only a limited set of data, the default parameter does not guarantee high quality for most of the datasets ([DS16]). Hence, we felt that there is a need

for a proper workflow which can be referred to the scientific community to avoid the common mistakes and improve the assembly quality. The workflow can guide the users through necessary pre-processing steps and appropriate data-dependant parameter selection.

Third generation sequencing technologies such as Oxford Nanopore and Pacific Biosciences are also gaining prominence in the current scientific community. These technologies produce reads which are long enough to cover an entire transcript which enables accurate expression estimation. However, they are marred by sequencing errors which hinder their usage for various applications ([Byr+17]). Currently, researchers combine the long read and short-read data to reduce the effect of sequencing errors in the analysis ([Des+16; Sov+16; FWA19]). Two of the areas in which long reads can be applied is the detection of alternate isoforms in a species and prediction of gene fusion events. But, there are only a few long read specific tool which detects and quantify isoforms and to our knowledge, there is no long-read specific tool to predict gene-fusion events.

1.5 Structure of the thesis

In Chapter 2, we will visit some of the most commonly used tools used for de novo transcriptome assembly. We will learn about their functionalities, their advantages, and their shortcomings. Chapter 3 would try to answer the question of what percentage of the data is actually used by the assembler. We propose a set multi-cover based algorithm to remove redundant reads from a high-coverage dataset without affecting the quality of the final assembly. In chapter 4, we suggest a workflow around de novo transcriptome assembly which has a high positive effect on the performance of an assembler. In chapter 5, we will move our focus towards long reads. We will learn about an algorithm that quantifies transcripts and predicts gene fusion events in different cell lines. Finally, we will summarize the work done and look towards the future direction in the field of transcriptome assembly and analysis.

Chapter 2

Algorithms for de novo transcriptomic assembly

Advances in sequencing technologies have enabled the assembly of an entire transcriptome even without the presence of a reference. These assemblies are further used in applications such as gene-expression analysis, differential expression of allelic variants, and detection of fusion genes. In the first half of this chapter, we will revisit some of the concepts from computer science and computational biology which are required to understand the functioning of an assembly algorithms. Second half of the chapter will introduce some of the state-of-the-art algorithms which are commonly used for de novo transcriptome assembly.

2.1 Sequence

In bioinformatics, biological sequences are subjected to wide variety of analytical treatment to mine information about its structure, function and evolution. Generally, a *sequence* is represented as a concatenation of characters from the set $\sigma = A, C, T, G, N$. The characters A, C, T and G represents the four nucleotide bases, details about which were given in the previous chapter. Character N is used if the base at a particular position is unknown. The length of a sequence is denoted as $l = |s|$. If i is an index where $1 \leq i \leq l$ then any position in the sequence s is denoted by $s[i]$ such that $s[1]$ denotes the first character of the sequence and $s[l]$ denotes the last character of the sequence. Similarly, if i and j are two indices where $1 \leq i \leq j \leq l$, then the sub-sequence between i and j is denoted as $s[i..j]$. In this work, the termed k-mer will be used to denote a sub-sequence of length k. If $S = s_1, s_2, ..., s_n$ denotes a set of n sequences, then the number of occurrences of a k-mer in S is denoted as $occ_S(k\text{mer})$. The entire set of k-mers present in the set S is termed as $k\text{-universe}_S$.

2.1.1 Sequence Alignment

Perhaps one of the most fundamental step in any downstream analysis of biological data is sequence alignment. In simple terms, sequence alignment is arranging sequences to find the region of similarity between them. If only two sequences are arranged against each other then the process is known as *pairwise alignment*. If more than two sequences are compared against each other, them the process becomes multiple alignment. The alignment can span over the entire length of the sequence. Such an alignment is termed as *global* alignment. On the other hand, a *local* alignment deals with only a certain region with in the sequences. Though multiple methods have been developed for sequence alignment, most of them are based on dynamic programming. It is applied via *Needleman-Wunsch* algorithm to

achieve global alignment ([NW70]) and *Smith-Watermann* algorithm to obtain local alignment ([SW81]). In general, dynamic programming fills up a substitution matrix with scores which represents match, mismatch or a gap penalty. Alignment is achieved by retracting paths in the substitution matrix which minimizes the total score. Although efficient, dynamic programming is computationally expensive both in time and memory. An alternative to this is a word based alignment method which is especially useful for large scale datasets. Although, word based methods are heuristics and do not guarantee optimal results, they are shown to be efficient as compared to dynamic programming. The most common word based alignment strategy is *seed-and-extent* wherein short regions of exact match is found in the query (seed step). The alignment is initiated from the seed and extended in both the directions of the match (extend step) allowing for mis-matches and substitutions ([Alt+97; Del02]). Although efficient, this strategy is time consuming and the performance is affected if there are repetitive seeds in the reference ([Jai+17a]). Many applications do not require accurate base-to-base mapping. Rather, they look towards finding regions of similarity within the query sequence. This can be achieved by calculating the Jaccard similarity coefficents between two subsections of sequences which can computed efficiently ([Bro97]). This concept gave rise to new mapping strategies used in tools like Minimap, MinHash alignment and BALAUR. Effectiveness of such algorithms is still a topic of research which is being researched upon.

2.1.2 Graphs

Since the advent of graph theory in the early 1700s, it has been successfully applied to solve complex mathematical and computer science problems. More recently, it has been extensively used for biological problems. In general terms, a *graph* is represented as an ordered pair $G = (V, E)$ where V is the set of vertices(or nodes) and E is the set of edges which has a binary relation on V. Graphs can be *finite* and *infinite* depending upon its order. All graphs in this work are finite unless otherwise stated. An *empty* graph is a graph with no vertex and edge set. It is denoted by (ϕ, ϕ) or simply ϕ. The vertex set of graph G is denoted as $V(G)$ and the edge set of G is denoted as $E(G)$. Each edge in the set $E(G)$ connects two vertices from the vertex set $V(G)$. Hence each element of $E(G)$ is a two element subset of $V(G)$, i.e, $E \subseteq [V]^2$. Figure 2.1a represents a schematic diagram of a graph $G = (V, E)$ where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (1, 3), (2, 3), (3, 4)$. The number of vertices in the graph is termed as the *order* of the graph and is denoted as $|G|$. The number of edges in the graph is denoted as $||G||$. For any vertex $v \in V$, an edge $e \in E$ is said to be incident on v if $v \in e$. If edge $e = (v_1, v_2)$ connects two vertices v_1 and v_2, then v_1 and v_2 are termed as *end-vertices*. If one of the two end-vertices is assigned as initial vertex and the other end is assigned as terminal vertex, then the edge is referred as a *directed edge* otherwise it is *undirected*. If a graph G_D consists of only directed edges, then it is termed as a *directed graph*. Figure 2.1b depicts a directed graph $G_D = (V, E)$. A graph $G' = (V', E')$ is a *sub-graph* of G, denoted as $G' \subseteq G$, if $V' \subseteq V$ and $E'' \subseteq E$. Figure 2.1c represents a sub-graph G' of the directed graph G_D. If edges or vertices of a graph have a weight associated with them, then the graph is known as *weighted graph*. If such as a graph is also directed then it is termed as a*directed weighted graph* otherwise it becomes a *undirected weighted graph*. The number of egdes originating from a vertex v is known as the *out-degree* of v. Similarly, the number of edges terminating at v is known as the *in-degree* of v. A vertex in a graph is *balanced* if its in-degree is equal to its out-degree.

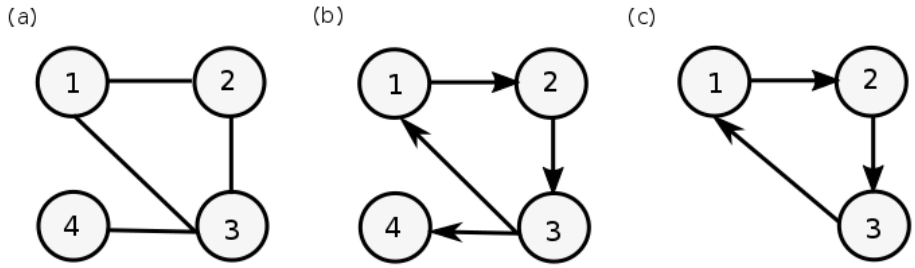

FIGURE 2.1: The graph on vertex V=1,2,3,4

Path or a *walk* in a graph $G = (V, E)$ from vertex $v_1 \in V$ to vertex $v_k \in V$ is a sequence of pair of vertices $(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)$ such that $(v_{i-1}, v_i) \in E$. Vertex v_1 is termed as the *source* vertex and v_n represents the destination vertex. In fig. 2.1a, the path from vertex 1 to vertex 3 is either the set {1,2,3} consisting of edges {(1,2),(2,3)} or the set {1,3} consisting of edges {(1,3)}. The number of edges in the path is termed as the *length* of the graph. If no edge is repeated in a path, then the path is termed as a *trail*. A graph is said to be *cyclic* if it has a trail where the source vertex is same as the destination vertex. If there is no cycle in the graph then it is termed as *acyclic*. The graph G is called *connected* if there is a path from all the vertices to all the other vertices in G. Figure 2.1a represents a connected graph since all the vertices can be connected to other vertices via a path. However, fig 2.1b is not a connected graph as there is no path from vertex 4 to vertex 2 or to vertex 1. A sub-path of the a path p is any contiguous sub-sequence of p. For instance, if $p = ((v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$ represents a path from vertex v_0 to v_k, then the sub-path of p can be represented as the sequence $((v_i, v_{i+1}), (v_{i+1}, v_{i+2}..., (v_{j-1}, v_j)$ where $v_i, v_j \in p$ and $0 < i < j < k$.

Eulerian graphs

First used by Leonard Euler in the year 1736 while solving the famous *Seven Bridges of Königsberg* problem, a eulerian graph is extensively used in bioinformatics especially in the field of sequence assembly ([PTW01],[NP13]). A *eulerian path* of a graph G is a trail in G if every edge in the path is traversed only once. However, a vertex can be traversed more than once in a eulerian path. A *eulerian cycle* or a *eulerian circuit* is a eulerian path which starts and ends at the same vertex. A graph with such a circuit is termed as a *eulerian graph*. A graph where every vertex has an even degree is also termed as a eulerian graph. Figure 2.2 represents a simple eulerian graph which has a eulerian circuit $p = ((1,2),(2,3),(3,4),(4,2),(2,1))$. Here, the path starts and end on the same vertex (vertex 1) and each edge of the path is visited only once.

Hamiltonian graphs

Another interesting variant of a graph is a *hamiltonian graph*. A *hamiltonian path* is a path in which each vertex is visited exactly once. Figure **??** represents a hamiltonian path in a dodecahedron graph. Dashed lines represents the hamiltonian paths in the graph. If a hamiltonian path is also a cycle then the path is termed as *hamiltonian cycle*. If a graph consists of a *hamiltonian cycle*, then it is termed as a *hamiltonian graph*. The problem of whether a hamiltonian path exists in a graph is a *hamiltonian path*

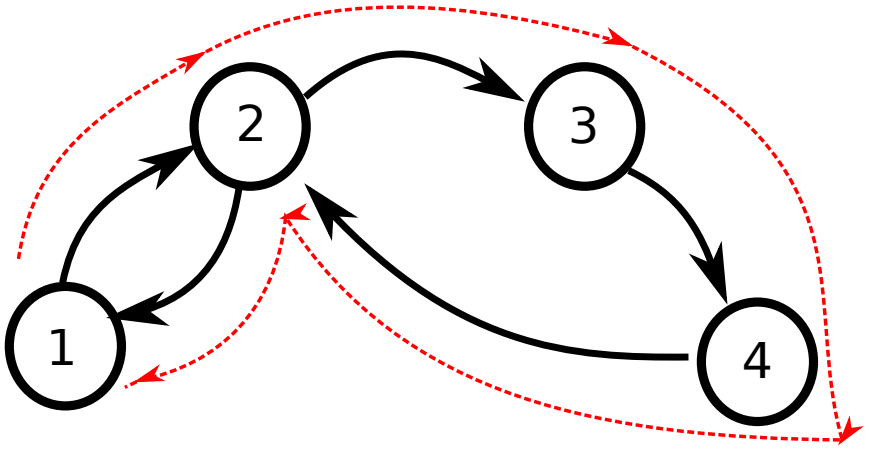

FIGURE 2.2: Eulerian graph consisting of circuit $p = ((1,2),(2,3),(3,4),(4,2),(2,1))$ depicted by red dashed line

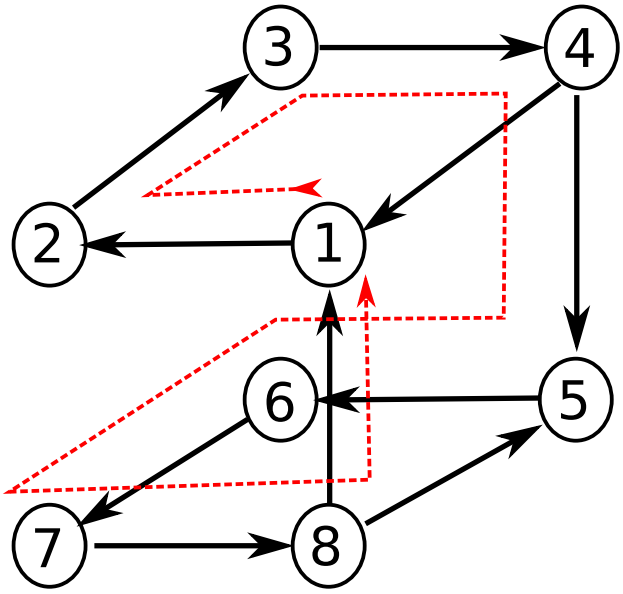

FIGURE 2.3: Hamiltonian graph containing a hamiltonian circuit depicted by the red dashed line

problem which is NP complete. Figure 2.3 depicts a hamiltonian graph containing the hamiltonian circuit $p = ((1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,1))$. It can be noted that the hamiltonian circuit can be converted into a hamiltonian path by removing just the last edge (in the above case the edge $(8,1)$) from the path.

De Bruijn graph

In this thesis, we will mostly deal with de novo transcriptome assembly algorithms which are based on de Bruijn graphs ([PTW01; CPT11]). Introduced by Nicolaas Govert de Bruijn ([Bru46]), a de bruijn graph is a directed graph which represents overlaps between series of symbols. We consider $S = \{s_1, s_2, ..., s_m\}$ as a set of m symbols and the vertices of the de Bruijn graph as $V = \{\{s_1, s_1, s_1, ..., s_1\}, \{s_1, s_1, s_1, ..., s_2\}, \{s_1, s_1, s_1, ..., s_m\}, \{s_1, s_1, s_1, ..., s_2, s_1\}, ..., \{s_m, s_m, s_m, ..., s_m\}\}$. In other words, vertices are strings pf length n generated by concatenating various combinations of the symbols. Two vertices $v_1, v_2 \in V$ would be connected by a directed edge if v_1 can be converted into v_2 by

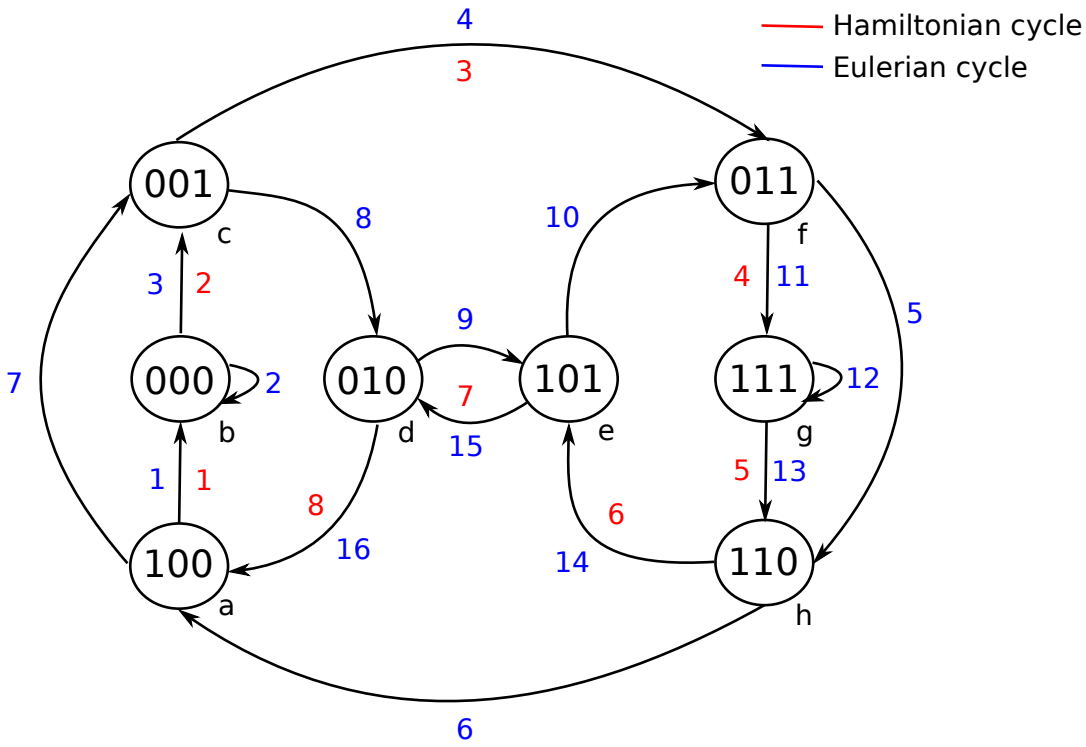

FIGURE 2.4: De bruijn graph generated strings of length 3 generated by concatenating symbols 0 and 1. Two vertices are connected by a directed edge if removing the first symbol and adding a symbol towards the end of the source vertex converts it into the destination vertex. The sequence of edges in the eulerian cycle of the graph is depicted in blue. The sequence of edges in the hamiltonian cycle of the graph is depicted in red.

removing the first symbol and adding one symbol at the end of v_1. Every de Bruijn graph is eulerian and hamiltonian. The sequences resulting from the eulerian cycle and the hamiltonian cycle in the de Bruijn graph are known as de bruijn sequences. The de Bruijn sequence is formed by taking the sequence of the first vertex and sequentially concatenating the last character of each vertex along the path. We show a sample de Bruijn graph consisting of vertices, each of length 3, generated by various combinations of the symbols 0s and 1s. A possible de Bruijn sequence can be generated by following the eulerian path (denoted as blue numbers in the figure) along the graph which would lead to the sequence 10001100101111010.

Most of the de novo assemblers generate a de Bruijn graph from RNA-seq data. The k-mers (sub-strings of length k) from the reads are considered as the vertices of the graph and two vertices are connected by a directed edge if they have a $k-1$ overlap. Contigs are generated by following various eulerian paths in the graph. In the following sections, we will look into some of the well-utilized practices in de novo assembly procedures. For the sake of brevity, the techniques and algorithms related to transcriptome assembly are shortly summarized. However, the algorithms which we have used throughout this work are explained in detail.

2.2 Pre-processing of RNA-seq data

Data generated by sequencing technologies are marred by artifacts such as sequencing errors, PCR contamination, over-represented *k*-mers and redundant adapter sequences. These artifacts have a huge effect on the quality of the assembly produced. For instance, sequencing errors results in the loss of overlap between two read sequences. Since most of the assemblers rely on the overlap between read sequences to connect two regions of a genome, a error in the sequence results in fragmented assembly. Also, contaminants like adapter sequences and PCR artifacts increase the memory and runtime requirements of an assembler without adding any useful information to the final assembly ([Le+13]). Hence, it is important to pre-process the reads before the assembly process.

2.2.1 Quality control

The quality control of reads includes adapter trimming, checking for the GC content, and checking for over-representation of *k*-mers. Software packages such as FastQC ([And+15]) and NGSQC ([Dai+10]) provide an efficient way to do the above analysis. They provide a detailed report on aspects such as per base sequence quality, per base GC content and average read quality. Practitioners can run these tools not only to gauge the quality of sequences produced by the sequencing run but also to measure the quality of the starting material used for library preparation. Once the low-quality regions are identified using the above tools, other software packages such as FASTX-toolkit ([GHG14]), trimmmomatic ([BLU14]), cutadapt ([MW11]) and AfterQC ([Che+17]) can be used to remove low quality reads and trim adapter sequences.

2.2.2 Error correction

Although the RNA-seq technologies are more accurate than DNA-microarrays, they still are hindered by sequencing errors. In case of Illumina sequencing, these errors are more probable towards the 3′ end of the read than the 5′ end ([Liu+12; ME13]). The above-mentioned methods are efficient in removing low-quality bases and reduce the number of absolute errors from the data, But it also leads to a significant loss in sequence data which in turn affects the ability of an assembler to reconstruct transcripts especially from the low expressed regions ([ME13; SBJZ17]).

In the genomic world, there are algorithms such as SHREC ([Sch+09]), Quake ([KSS10]), and HiTec ([IFI11]). However, they are all designed with the specifics of DNA sequencing. The *k*-mers generated from DNA-sequencers display a uniform coverage distribution. Hence, any *k*-mer, which has a low abundance (generally between 1-3) can be considered as an error and be subsituted with an alternate similar *k*-mer having a higher abundance. However, reads generated from RNA-seq experiments show a non-uniform distribution. Hence, the above error correction methods are not suited for RNA-seq data. RNA-specific error correction algorithms includes tools like Hammer ([Med+11]), BayesHammer ([NKA13]), SEECER ([Le+13]), and RCorrector ([SF15]). Hammer, BayesHammer and RCorrector identify a group of "solid *k*-mers" based on their abundance information. These solid *k*-mers are considered as correct *k*-mers and generally have high abundance. The erroneous *k*-mers or "weak"

k-mers are corrected based on their edit distance to the solid k-mers. However indels, which are insertions and deletions in the genome, are not corrected by these models.

In our work, we used SEECER which has sensitive error correction approach towards indel errors. We will now look into the workflow of SEECER. The description and the terminologies used here are based on the manuscript written by [Le+13].

SEECER: error correction of RNA-seq data using Hidden Markov Model (HMM)

Given a read to correct, SEECER first tries to reconstruct the transcript/contig from which the read has originated. This is done by looking into the overlapping reads which might belong to the same transcripts. It then characterizes the contig as a profile HMM and identifies and corrects the error according to it. It performs the above tasks in the following way:

1. Given a read dataset $\mathcal{P}$, SEECER begins by counting the k-mers in $\mathcal{P}$ and storing all the k-mers whose count is above a threshold in a hash table. The hash table stores the read indices and the position of each k-mer within the reads and can be accessed via the k-mer sequences as the keys. The counting of k-mers is done by the JellyFish program ([MK11]) and the read sequences are stored and analysed using the SeqAn library ([Dör+08]).

   SEECER proceeds by selecting a random read r from $\mathcal{P}$ and considers it as seed read. It then obtains a set of reads $\mathcal{S} \subseteq \mathcal{P}$ such that each read in $\mathcal{S}$ shares at least one k-mer with r. Simultaneously, it generates a multiple sequence alignment (MSA) matrix M using the reads from $\mathcal{S}$. All reads in $\mathcal{S}$ are said to be *assigned* to r. This procedure is repeated for multiple seed reads and the reads which remain *unassigned* to any of the seed read are kept in a pool of unassigned reads.

2. To build a homogeneous contig, the read sequence belonging to the contig should originate from the same transcript. However, the sequences in $\mathcal{S}$ might come from different transcripts sharing the same/similar sequence. Hence, the next step of SEECER is to perform cluster analysis on $\mathcal{S}$ to find the largest subset $\mathcal{S}^*$ which satisfies a quality measure. This is done by clustering columns of the MSA matrix M and getting $\mathcal{S}^*$ based on the clusters. This ensures the fact that reads in $\mathcal{S}^*$ arise from a real biological difference as they would share a similar set of mismatches. This subset of reads is then used to generate the initial set of contig HMM.

3. SEECER learns the initial HMM and estimates the parameters based on the alignment of reads and the k-mer positions within the reads. It then uses the consensus sequence defined by the contig HMM and extracts additional reads, which are not assigned to any other read. These additional reads overlap the edges of HMM and hence are used to extend the HMM in both the directions. When no more reads can be added to the HMM, a likelihood of the read being generated from the contig HMM is calculated. If the likelihood is above a threshold, then the contig HMM is used to correct the errors in the read. The error-corrected reads are removed from the pool of unassigned reads.

With regards to the performance, SEECER is effective in identifying and correcting errors in reads generated from RNA-seq sequencers. This was evident from the

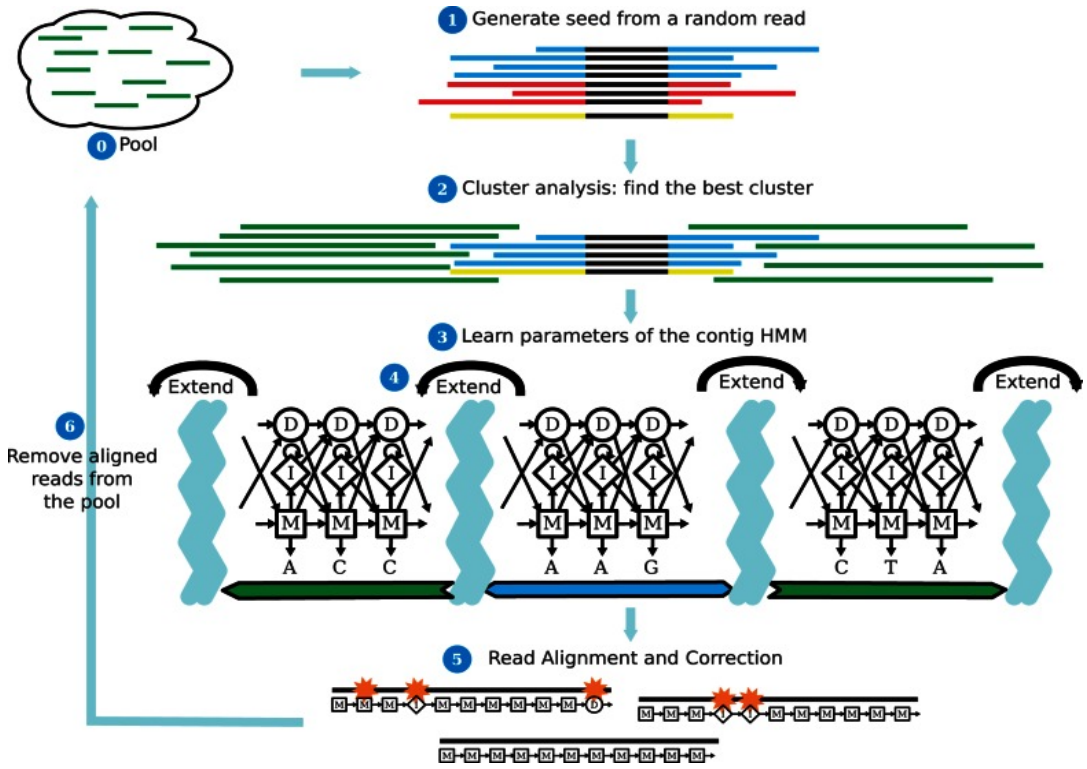

FIGURE 2.5: Workflow of the SEECER. 0) The input dataset is considered as the intial pool of reads. 1) SEECER selects a random read from the pool as a seed and collects a set $\mathcal{S}$ of reads such that each read in the set shares at least one k-mer with the seed read. Simultaneously, SEECER generates a multiple sequence alignment of reads in $\mathcal{S}$ and generates a matrix M. 2) SEECER analysis the matrix and obtains a set $\mathcal{S}^*$ which consists of reads originating from the same transcripts. 3) The reads in $\mathcal{S}*$ are used for building an initial HMM. 4) and 5) The consensus sequence defined by the contig HMM is used to extract additional reads which are not assigned to any other reads. These additional reads are used to extend the contig HMM. The final HMM sequence is used to correct errors in the read. 6) The error corrected reads and the reads aligned to the HMM are removed from the pool of reads. *Illustration taken from [Le+13]*

improvement in the quality of assembly when error-corrected datasets were given as an input to the assembler ([Mac15]). Also, SEECER has been shown to perform well on both substitution as well as indel errors ([SBJZ17]). The assemblies generated from SEECER corrected data were also successful in detecting differential expressed genes between two conditions ([Le+13]).

2.2.3 Read normalization

It has been shown that 100x or more sequencing coverage is required to cover the entire human genome ([Gne+11]). Transcriptomes and metagenomes also require similar coverage as they have low expressed regions which needs to covered using deep sampling ([Bro+12]). This results in generation of millions of short reads which needs to be assembled within the limits of the available computational resources. While there is a considerable improvement in hardware technologies, the sequencing capacity is growing faster than the computational capacity. Hence, to make the assembly of high coverage dataset possible, practitioners normalize the input read datasets using k-mer based algorithms such as Diginorm ([Bro+12]), BigNorm ([Wed+17]), and NeatFreq ([McC+14]). The latest version of the Trinity ([Gra+11]) assembler has its own k-mer abundance based normalization step which is integrated in its assembly algorithm.

Although efficient, all the above mentioned algorithms come with a risk of losing important sequence information which effects the quality of the final assembly. We proposed a normalization algorithms, named ORNA and ORNA-Q/K, which reduce the input RNA-seq data without losing any sequence information from the original dataset. The details of the algorithms is given in chapter 3 of this thesis. We compared our normalization algorithms with Diginorm, Trinity's In Silico (TIS) normalization and Bignorm which is described here. The description and the terminologies used here are based on the manuscript written by [Bro+12] (Diginorm), [Gra+11] (TIS) and [Wed+17] (Bignorm).

Diginorm

Digital normalization or "Diginorm" aims to reduce the input dataset $\mathcal{P}$ to $\mathcal{P}'$ by removing erroneous and redundant reads and limits the coverage of $\mathcal{P}'$ to a user-defined threshold C. But how do we determine whether a read contains errors without mapping them to a reference genome first? The authors of Diginorm had the following idea. It is implied that all k-mers within a read should have similar abundances since they originate from a single source molecule. If a read contains an error, k-mers overlapping the error would have lower abundances as compared to their surrounding k-mers. Using simulated data from an artificial genome and real data from *E.Coli*, [Bro+12] established a correlation between the median k-mer abundances of the dataset and the coverage of the dataset. They found that reads which contains multiple sequencing errors had a skewed median k-mer abundance.

Based on the above observation, Diginorm formulates its normalization as a three-step approach as described below.

1. In the first pass, Diginorm extracts all the k-mers from a given read $r \in \mathcal{P}$ and calculates their abundances in $\mathcal{P}'$. In order to save memory consumption, Diginorm uses the CountMin sketch data structure ([CM05]) to obtain an appropriate estimate of the k-mer count. Count-min sketch is a probabilistic data

structure which maps events, in our case k-mers, to their frequencies. If the median abundance of k-mers in $\mathcal{P}''$ is below the user-defined threshold C, the read r is kept otherwise it is rejected.

2. In the second pass, Diginorm counts the k-mers present across the accepted reads. It then goes through the accepted reads and removes the 3′ ends of reads having low-abundance k-mers.

3. In the third pass, Diginorm executes a second round of normalization on the accepted reads, with the same parameters as the first pass, and eliminates redundant data.

The three-pass approach of Diginorm was shown to eliminate most of the reads with sequencing error. However in the case of reads from RNA-seq technologies, it was observed that a lot of non-erroneous k-mers were also being eliminated along with the erroneous ones. This is because Diginorm is unable to distinguish between sequencing errors and k-mers from under sampled regions. Also, Diginorm obtains k-mer abundance information from the count-min sketch. When the memory set for the data structure is not too large as compared to the k-mer space, the k-mer counts may include false positives.

In terms of assembly performance, it was shown the time and memory requirements for genome assemblies reduced by a factor of 10 whereas for transcriptome assemblies, the resource requirement reduced by a factor of 3. The genome assemblies produced by Diginorm reduced datasets were able to overlap upto 98% to known reference sequences. Whereas in case of transcriptome, assemblies produced by normalized datasets overlapped upto 97% with the assemblies generated from the original datasets.

Bignorm

Although efficient, the performance of Diginorm can be improved substantially by incorporating Phred score information in its decision making. The Phred score of a base is inversely proportional to the probability of the base being called incorrectly. Hence, a low Phred score indicates that the base is a sequencing error. Also, Diginorm handles Ns (uncertain bases in the read) inefficiently as it converts all the Ns to As. The idea of incorporating Phred scores was tested out by [Wed+17] in their algorithm Bignorm. Bignorm extends the approach of Diginorm by utilizing a quality score cut-off based on Phred scores and formulates the normalization approach in the following way.

1. Let r be a read from a dataset $\mathcal{P}$ of length s. Every position $t \leq s$ in r would have a Phred score $q_r(t)$ associated with it. Bignorm extracts the k-mers set K_r from r.

2. For each k-mer $l \in K_r$ beginning from a position p in r, it extracts the Phred scores of all nucleotides belonging to l. Let $Q_r(l,p) = \{q_r(p), q_r(p+1), ..., q_r(p+k-1)\}$ be the set of Phred scores of all nucleotides belonging to k-mer l starting from position p.
Bignorm calculates the quality score $qs_r(l)$ of the k-mer l by taking the lowest

Phred score from the set $Q_r(l, p)$. This procedure is followed till the position $s - k$ in r.

3. Bignorm then keeps a read r if there are at-least b k-mers in r such that:
   (a) the quality scores of all the b k-mers are above a user defined threshold Q_0
   (b) the abundance of all the b k-mers in the original dataset is above a *rarity threshold* and below an *abundance threshold*.

   Condition (a) ensures that only high quality k-mers are included in the decision making. Condition (b) ensures that the k-mers which are kept are too frequent to be considered as a sequencing error (as they are above rarity threshold) and not too abundant to be considered as a redundancy.

Like Diginorm, Bignorm also uses CountMin sketch for counting k-mers. However, it provides a faster implementation by efficient hashing and usage of OpenMP functionality (C programming language) for parallel processing. Bignorm is also able to reduce more number of reads as compared to Diginorm. However, the speedup in time and memory comes at a cost of the assembling full length transcripts as the Bignorm reduced datasets resulted in a shorter contigs as compared to Diginorm reduced datasets.

Trinity's In Silico (TIS) normalization

The Trinity assembler has its own normalization algorithm integrated with it. The procedure is optional and can be switched off if not required. The algorithm can also be run independently using a provided script. The normalization procedure is formulated in the following way:

1. Given a read dataset $\mathcal{P}$ and a desired coverage c, TIS runs the k-mer counting software jellyfish to store the abundances of k-mers present in the reads of $\mathcal{P}$. Next, given a read $r \in \mathcal{P}$, TIS calculates the average ($avg(r)$), median ($med(r)$) and standard deviation ($stdev(r)$) of abundances of k-mers present in r.

2. For each read r, TIS selects a random number (*rand*) between 0 and 1. The read r is kept if both the following conditions are met:
   (a) The random number *rand* is greater than the ratio of the desired coverage c and median k-mer coverage $med(r)$.
   (b) The ratio of standard deviation of k-mer coverage $stdev(r)$ and average k-mer coverage $avg(r)$is below a cutoff (set as 100 by default).

   Condition (a) ensures that only reads with very high k-mer coverage are part of the decision process and the reads with low k-mer coverage are always kept. Reads which contain multiple errors would have a high standard deviation as the k-mer abundance across the read would not be similar. Hence, condition (b) ensures that such reads are always discarded.

2.3 Transcriptome assembly

The complexity of a transcriptome makes the procedure of assembly a rigid task to fulfil. There is a high variability in the expression levels of various transcript. On one

hand, there are transcripts which are highly expressed, whereas on the other hand their might be many transcripts which have a shallow expression level. Alternate splicing event makes the process even more complicated as one locus can produce many transcripts due to which resolving the isoforms becomes a major challenge. An efficient transcriptome assembler should successfully deal with the above issues and should be able to recover accurate full-length transcripts at various levels of expression. This challenge and the scope has resulted in the development of many transcriptome assemblers. The present assemblers are divided into two categories namely - reference based and de novo.

2.3.1 Reference based assembly

For a lot of well studied species, a high-quality reference genome is available via public databases such as Ensembl([Zer+18]) and Gencode ([Har+12]). Many assemblers use these reference genomes generate the target transcriptome. A reference-based assembly consists of three major steps. The first step involves aligning of RNA-seq reads against the reference genome using splice-aware aligners such as TopHat ([TPS09]) and Blat ([Ken02]). This step is followed by building a graph based on overlapping reads from each locus. Reads from RNA-seq datasets form the nodes of the graph. Two nodes are connected if the constituting reads have bases overlapping with each other. The final step involves the traversal of the graph to resolve individual isoforms. Some of the well known assemblers which work on the basis of a reference genomes are Cufflinks ([Tra+10]), Scripture ([Gut+10]), IsoLasso ([LFJ11]), SpliceGrapher ([Rog+12]), StringTie ([Per+15]) and scallop ([SK17]).

All the above algorithms have the same step of aligning reads against a reference genome using a splice aware aligner such as TopHat. However, the difference arises in the way the assemblers utilize the alignments to generate sequences. For instance, Cufflinks uses the alignments to assembles transcripts based on mapped fragments sorted by reference position. One can imply the intronic positions in the fragments based on the reference genome coordinates. Cufflinks first divides the non-overlapping fragments into classes and assembles each class separately. Inside a class, each fragmented alignment is assigned to a node in the "overlap graph" G. Two nodes are connected by a directed edge if their alignments overlapped in the genome and the two fragments are compatible. Two fragments are *compatible* if the implied intron of one fragment matches the implied intron of the other. Cufflinks resolves the *overlap graph* by finding the minimum path cover of the graph. In other words, it finds a minimum cardinality set of paths such that each fragment of the graph is a part of at least one path. Each path in the set is a series of mutually compatible fragments connected from left to right in the graph. A major drawback of Cufflinks is the low number of transcripts assembled from a locus.

Scripture, on the other hand, has less stringent conditions. From the alignments, Scripture first considers all the spliced reads, i.e, reads which have gaps in their alignment. It then forms a connectivity graph where each base in the chromosome acts as a node. A directed edge connects a base to the next base in the genome and also to another base if there is a read supporting this connection. Scripture then finds all the paths through the graph which have statistically significant read coverage. In this way, Scripture produces far more transcripts from a locus than any other assembler. However, it also produces a lot of false positives and its assembly generally results in a low precision ([Per+15]). The IsoLasso algorithm introduced by [LFJ11]

extends the approach to Scripture. It begins by enumerating all the isoforms from the connectivity graph as done in Scripture. IsoLasso then uses the Least Absolute Shrinkage and Selection Operator (LASSO) ([Tib96]) method to estimate the isoform abundance given the read alignment. It then filters out all the isoforms for which the abundance estimation does not comply with the number of reads mapped to it.

The SpliceGrapher approache makes use of Expressed Sequence Tags (EST) alignments along with gapped and ungapped alignment to infer a *splice-graph* from the read dataset. A splice graph is a graph where each node represents an exon and each path in the graph represents an alternate isoform. SpliceGrapher then uses a machine learning approach to construct a database of potential splice site and removes any assembled isoform which does not match with the database. On some occasions, short reads tend to map ambiguously to multiple regions of the genome which complicates the assembly process. The recently developed algorithm StringTie tackles this issue by first creating a *super read* from the initial set of data. A super read is generated by considering a seed read and extending it in both directions using other reads which overlaps with the seed read ([Zim+13]). StringTie maps the super read to the genome to form an *Alternate Splice Graph* (ASG) for each gene locus.Like in SpliceGrapher, an ASG represents all possible transcripts of a gene where each node represents a contiguous regions of the genome defined by uninterrupted splice-aware alignment of reads and edges represents reads which align across two nodes from 5′ to 3′ of the genome. StringTie then considers the path, also known as the *heaviest path*, with the highest read per base coverage as an assembled transcript. In addition to this, StringTie estimates the coverage level of the transcript by solving a maximum-flow problem which equates to the maximum number of fragments that can be associated with the chosen transcript. The above iterations are stopped when the coverage of the heaviest path in the ASG drops below some fixed threshold.

Advantages of reference based assembly

Presence of a reference sequence serves as the major advantage for reference-based transcriptome assemblers. Artifacts such as sequencing errors have a negligible impact of the assembly process as they would hardly get aligned to the reference genome. Also, gaps in the alignment can be filled since the original sequence is already known. This allows the assembly of regions having low expression quite efficient and accurate. Similarly, reference based assembly produces longer stretch of UTRs which have a low read coverage.

Disadvantages of a reference based assembly

Presence of the reference genome which serves as an advantage to such assembler can also prove to be the achilles heel of the assembler. A reference based assembly highly depends on the quality of the reference sequence available. Except for a few species, the genomic sequence contains large fraction of deletions in the sequences which might cause misassemblies. Also, as the field of next generation sequencing is advancing, more and more non-model species are getting sequenced whose reference genome are not available. For some non-model species, practitioners use a genome from closely related species. However, using a closely related genome does not guarantee the capture of divergent genomic regions. Lastly, reference based assemblers miss the assembly of trans-spliced genes ([MW11]) which are shown to be

useful in studying the genetic pathways in cell lines related to cancer.

In general, a reference based assembly is only preferred when a high-quality reference sequence is available. However as more and more non-model species are being sequenced, assembly using *de novo* approaches is becoming a routine. In this work, we worked only with algorithms related with *de novo* transcriptome assemblers which we discuss in the next few sections

2.3.2 De novo transcriptome assemblers

In the absence of a reference genome, practitioners assemble RNA-seq data using the *de novo* approaches. These approaches use short sub-strings from reads to construct a sequence graph and traverse eulerian paths within the graphs to build contigs. Most of the *de novo* approaches like TransABySS ([Rob+10]), Trinity ([Gra+11]), Oases ([Sch+12b]), and SOAPdenovo-trans ([Xie+14]) construct a de Bruijn graph (DBG) from the reads. Briefly, they disintegrate reads into *k*-mers and consider them as vertices of the graph. They connect two vertices by a directed edge if they have *k*-1 overlap. A simple "walk" within the graphs generates an initial set of contigs. The assembly algorithms then apply various heuristics on the set of contigs to generate the final assembly. An alternative of de Bruijn graphs is the splice graphs which is used by algorithms such as Bridger ([Cha+15]), BinPacker ([Liu+16]), and TransLiG ([Liu+19]). The splice graph representation is as accurate as the de Bruijn graphs. However, their computational resource requirements are lower as compared to the de Bruijn graphs. For our work, we use four de Bruijn graph based assembler namely TransABySS, Trinity, Oases and SOAPdenovo-trans and one splice graph based assembler namely TransLiG. We describe the functioning of each algorithm below.

Trinity

Trinity is an RNA-specific de novo assembler which combines the functioning of three *k*-mer based contig generation algorithms namely *Inchworm*, *Chrysalis* and *Butterfly*. Figure 2.6 depicts the functioning of trinity assembler.

1. *Inchworm (Figure 2.6a)*: Given the RNA-seq reads as input, Inchworm counts all 25-mers from the dataset. The erroneous *k*-mers which have low abundances are removed from the *k*-mer dictionary. Inchworm considers a *k*-mer having the highest frequency as a contig *k*-mer. Each contig is extended in both the direction by finding high frequency *k*-mers having *k*-1 overlap with the contig terminus. Inchworm concatenates the last base of the overlapping *k*-mer to grow the sequence of the contig. The above procedure is repeated for the next high frequency of *k*-mer until all the *k*-mers in the dictionary are exhausted.

2. *Chrysalis (Figure 2.6b):* Chrysalis builds on the contigs generated from the Inchworm step. It clusters the contigs into sets of connected components. Two clusters belong to the same component if (i) they have a *k*-1 overlap with each other (ii) there is a minimal number of reads which span the junction of the two contigs. Chrysalis then builds a de Bruijn graph (DBG) from each component with (*k*-1)-mers as nodes and *k*-mers as edges. It assigns a weight to each edge which is equal to the number of reads which supports the edge. Chrysalis then

assigns a read to a component with which it shares the maximum number of k-mers.

3. *Butterfly (Figure 2.6c):* Butterfly marks the last step of the trinity assembler. It takes the DBG from Chrysalis and merges consecutive nodes in linear paths. It prunes tips and minor edges which might have originated from sequencing errors. In the more recent versions of Trinity, the error correction step is done by removing k-mers whose abundance is less than 5% of the abundance of the adjacent k-mer in the graph. This is done keeping in mind that low k-mer abundance might not necessarily originate from a sequencing error. Hence, keeping a fixed abundance cut-off might result in some useful k-mers being clipped out/ Butterfly then identifies paths in the graph which are supported by reads (and their pairs) and traverses them to form contigs. The support is used mainly to solve ambiguities and reduce the combinatorial number of paths to a smaller number of contigs.

In recent times, the Trinity assembler was tried on many datasets and has been shown to be efficient in producing full-length transcripts. However, Trinity does not use the sequencing depth information to the full extent. It uses a brute force strategy to find transcript-representing paths which results in a high false-positive rate. Another major drawback of Trinity is that it uses only a single k-mer size (k=25) for constructing the graph. A single k-mer size might be too small to resolve repeat regions or might be too large to assemble low expressed regions of the genome.

TransABySS

Unlike Trinity, TransABySS generates assemblies using multiple k-mer sizes and merges them to form a single non-redundant assembly. It extends the implementation of genome assembler ABySS ([Sim+09]) to make it suitable for RNA-seq data. The working of TransABySS is as follows:

1. *DBG construction and initial contig set generation:* Given the input set of transcriptomic reads, TransABySS uses the graph construction step of ABySS to build a DBG. Repeat sequences with minor variations or sequencing errors form bubbles and tips in the graph. ABySS resolves those by removing the variant, with the lower coverage, from the graph. After the basic error correction, a "walk" in the graph generates *single-end* contigs. If paired-end information is available, then the pairs are aligned to the single-end contigs. Mates of a pair might get aligned to different single-end contigs. In such cases, contigs are merged unambiguously if the length of the contigs is above a certain threshold and they have sufficient mate-pair support.

2. *Transcript assembly and processing:* Starting from an initial k-mer size k_1, step 1 is repeated iteratively for several k-mer sizes. In the manuscript, [Rob+10] used a k-mer range starting from k=26 till k=50. For a k-mer size k_i, contigs longer than (2k-1)bps are considered as *main contigs* of k_i. The main contigs of k_i are aligned against the contigs generated using k_{i+1}. Similarly, the main contigs of k_{i+1} were aligned against the contigs of k_i. After each alignment, contigs which were completely subsumed by any other contigs are discarded. The above procedure is repeated for all the possible combinations of k-mer sizes in a hierarchical manner till a single set of homogeneous contigs is obtained. From the final set of contigs, any contig which is smaller than 150bps is eliminated from the final assembly.

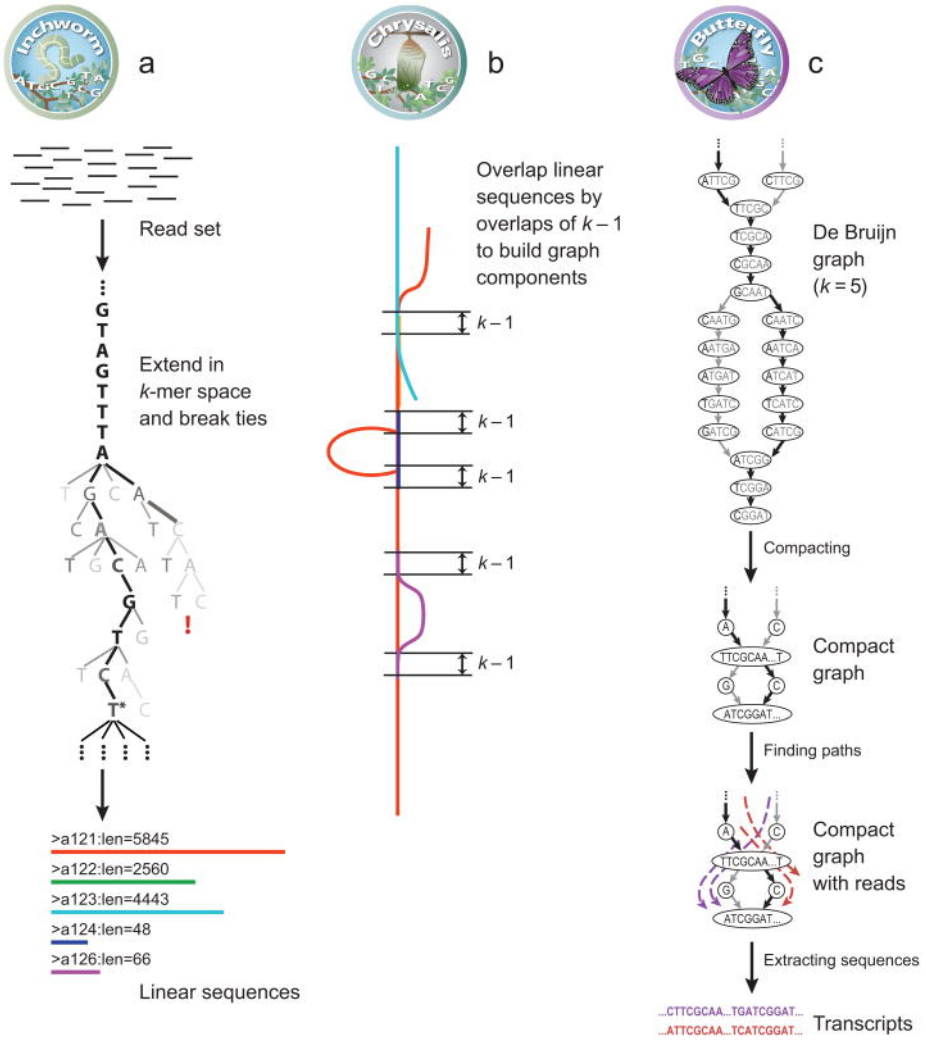

FIGURE 2.6: Workflow of the Trinity assembler. (a) The initial set of contigs are formed on the bases of overlap of *k*-mers from the input reads. (b) The set of contigs is clustered into a set of connected components. Two contigs are part of the same component if they have *k*-1 overlap and enough read support. DBGs are made from each of the components. (c) The DBGs generated from the previous step are simplified and traversed on the basis of a number of reads supporting a path. *Illustration taken from [Gra+13]*

Oases

Similar to TransABySS, Oases also merges assemblies from multiple *k*-mer sizes and recreates low expressed regions (using small *k*-mer sizes) and resolves repeat regions (using large *k*-mer sizes). Oases reuses the graph constructed by the velvet assembler ([ZB08]) and tweaks the graph traversal process to assemble transcripts from reads. Figure 2.7 depicts the functioning of Oases. Briefly,

1. *Contig Assembly and correction:* Like most of the de novo assemblers, Oases builds a de Bruijn graph from the reads and traverses them to form the initial set of fragmented assemblies. The graph building and storing steps are reused from the genome assembler Velvet ([ZB08]). Oases receives the set of fragmented assemblies and mapping of reads onto the fragments from Velvet (Figure 2.7-2). It then finds parallel paths in the graph which have the same starting and ending nodes (bubbles). Oases merges the path with a low read coverage with the path having a higher read coverage. In addition to this, Oases examines every node and removes an outgoing edge from the node if its coverage is less than 10% of the sum of the coverage of all outgoing edges from the node. The final correction step includes the removal of all contigs below a static coverage cutoff (<3x coverage). A contig is labeled as *long* if its length is greater than a pre-defined value (by default (50+k1) bp), otherwise it is labelled as *short*.

2. *Scaffold construction and filtering:* Oases summarizes the distance information between contigs as a set of distance estimates known as *scaffolds*. The distance estimate for a connection between two contigs can be supported by spanning single-end reads or paired-end reads. A connection is said to be direct if there are reads which span the connection, otherwise it is termed as indirect. A short contig can only be connected to a long contig by a direct connection. Oases removes a contig and its corresponding connections if the length of the contig is below a static threshold (by default (50-k+1) bp where k is the *k*-mer size). Similarly, Oases removes a connection if it has low read support, i.e, the number of reads spanning the connection is low. Also, indirect connections which have a read support less than a static threshold are eliminated.

3. *Loci construction and reduction:* Oases organises the contigs into clusters called *loci*. First, Oases considers long connected contigs as they are unique and are likely to come from the same gene (Figure 2.7-3). To this connection, Oases adds short contigs which are connected to one of the long contig (Figure 2.7-4). Oases elimates long distance and redundant connections (Figure 2.7-5).

4. *Transcript assembly from loci:* The sequence information lies in the loci. Oases analyzes the topology of each loci and classifies them as *chains*, *bubbles* and *forks*. These topologies are easily identifiable by the connections originating from the contigs. Oases detects these topologies and traverses all possible transcripts from them. There might be a complex topology which does not fit into any of these categories. For such cases, Oases uses the graph traversal algorithm suggested by [Lee03] to generate the transcripts.

5. *Merging assemblies from multiple k-mer sizes:* To capture transcripts from different regions of varied expression levels, the Oases-M script of the Oases package runs the assembler over multiple *k*-mer sizes. The lower *k*-mer sizes capture transcripts from low expressed regions whereas the higher *k*-mer sizes

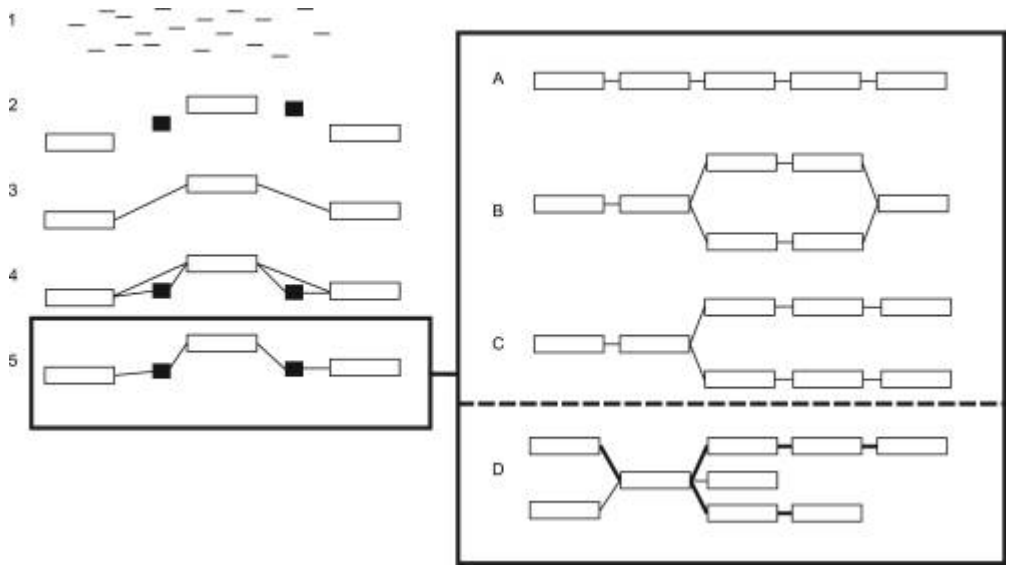

FIGURE 2.7: Workflow of the Oases assembler. (1) Reads from RNA-seq are given as input to velvet assembler. (2) Velvet assembler generates the initial set of contigs which is further used by Oases. (3) Oases clusters the contigs into loci and connects long contigs based on the reads spanning the two contigs. (4) Oases iteratively adds short contigs to the long contigs based on the reads spanning the short and the long contig. (5) Oases removes redundant and long distance connections from the graph. It then classifies the contigs into topologies namely (A) chain, (B) bubble, and (C) fork. If the toplogy doesnt fall in any of the above categories, then it is termed as (D) complex. Oases traverses each of these toplogies seperately to form the transcript assemblies. *Illustration taken from [Sch+12b]*

resolve repeat regions present in the genome. The assemblies or *transfrags* generated from the individual *k*-mer sizes are then fed to contig correction stage of the assembler with a different user defined *k*-mer size. Apart from applying all the correction steps described above, Oases-M also removes a *transfrag* if it is similar to a transfrag generated in any other *k*-mer iteration. The final assembly is constructed by following the remaining transfrags through the merged graph.

In their manuscript, [Sch+12b] tested Oases on Human and Mouse datasets and found it to outperform both Trinity and TransABySS in terms of assembly quality. Also, they show that running assemblies over multiple *k*-mer sizes and merging them produces more sensitive assemblies as compared to running assembler using a single *k*-mer size. However, running the assembler using multiple *k*-mer sizes results in a high number of transcripts most of which are misassemblies ([DS16]). Hence, an efficent reference-free post-processing step is required to remove such fraudulent transcripts. [DS16] proposed a clustering based technique to identify and remove misassemblies from the final assembly set. Such methods can studied and integrated into Oases to improve the quality of the final assembly.

SOAPdenovo-trans

As mentioned above, Oases produces a high percentage of redundant transcripts due to running assembler on multiple *k*-mer sizes and also due to a lack of efficient error removal mode ([LZS13]). On the other hand, Trinity does not use pair end information for contig connection and hence misses out on certain full-length transcripts. SOAPdenovo-trans (ST) extends the implementation of Trinity and Oases to

improve the assembly performance. Figure 2.10 depicts the functioning of SOAPdenovo-trans assembler.

The assembler consists of two main steps:

1. *Contig assembly:* Given the input RNA-seq dataset, the genome assembler SOAPdenovo2 ([Luo+12]) builds the initial de Bruijn graph using the k-mers from the reads. SOAPdenovo2 implements a variation of de Bruijn graph where it stores a large number of linear and unique k-mer as one combined unit. This reduces the runtime and memory consumption of the whole algorithm. For error correction, SOAPdenovo2 removes low-frequency k-mers to resolve bubbles, tips, and low-frequency edges. This is based on the assumption low-frequency k-mers are a result of sequencing errors and hence can be removed from the graph. By default, the low-frequency cut-off is set to 2. However, for highly expressed genes in a transcriptome, the abundance of the erroneous k-mers exceed this cut-off. Hence, SOAPdenovo-trans implements an additional step similar to the one used in Trinity. A local threshold is set for each k-mer which is equal to 5% of the abundance of adjacent graph elements (adjacent node k-mer). An unambiguous walk in the de Bruijn graph generated the initial set of contigs which is used further for scaffold construction.

2. *Transcript assembly:* ST maps the single-end reads and paired-end reads back to the contigs generated from SOAPdenovo2 (Figure 2.10A.2). It obtains the contig linkage information from either single-end reads spanning two contigs or from paired-end information (mate pairs mapped to two contigs). The distance information is estimated based on the insert size of the paired-end dataset. ST removes any contig that is shorter than a defined threshold (by default-100bps). This removes ambiguous contigs that are formed due to sequencing errors and repeat regions of the genome (Figure 2.10A3-1 and 2.10B). The assembler then proceeds by linearizing chains of contigs. Given three contigs c_1, c_2, and c_3, SOAPdenovo-trans linearizes the contigs if there exists an explicit linkage between c_1-c_2, c_2-c_3 and c_1-c_3 (Figure 2.10A3-2 and 2.10C). In other words, the assembler considers the linkage between c_1 and c_3 as redundant and removes it keeping only the linkage information between c_1-c_2 and c_2-c_3. Like Oases, SOAPdenovo-trans then clusters the contigs into sub-graphs (linear, bubble and fork) and traverses each sub-graph separately to assemble a transcript.

SOAPdenovo-trans combines the best parts of Oases and Trinity assembler and comes up with an memory-efficient way to generate accurate assembly. However, one of the major drawbacks of SOAPdenovo-trans is its inability to use strand information which results in generation of misassemblies and missing out of certain transcripts. Also, SOAPdenovo-trans, by default, runs only in single k-mer size mode. Assemblies can be generated using multiple k-mer sizes and merged together. However, efficient merging algorithm is required to perform this.

TransLiG

The above methodologies use a de bruijn graph to represent a transcriptome. However, Trinity, Oases and TransABySS required a huge amount of computational resources (memory and runtime). SOAPdenovo-trans outperforms the three algorithms in terms of computational resource requirements. But as mentioned above, it

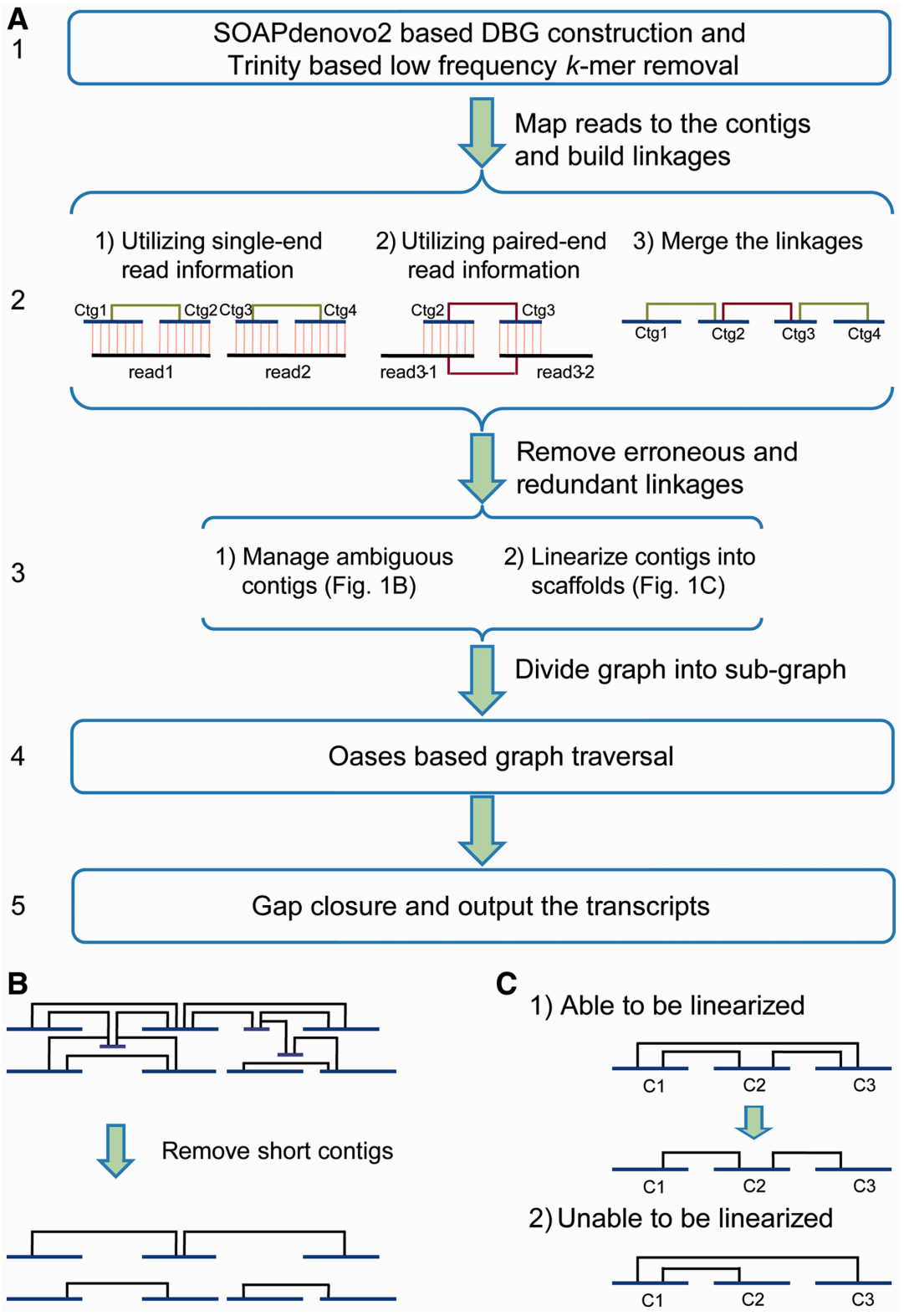

FIGURE 2.8: Workflow of the SOAPdenovo-trans assembler. (A) (1) DBG is constructed using SOAPdenovo2. Potential *k*-mers arising from sequencing errors are removed by cobining the strategies used in SOAPdenovo2 and Trinity. A walk in the graph generates the initial set of contigs (2) Reads are mapped back to contigs are linkage information is obtained based on the mapping. (3) Contigs shorter than a length threshold are removed. Remaining contigs are linearized according to their linkage information. (4) Contigs are clustered into sub-graphs and each sub-graph is traversed individually to generate transcripts. (5) Gaps in the contigs, which were generated due to the removal of short contigs, are filled using the semi-mapped paired-end reads. *Illustration taken from [Xie+14]*

produces a high percentage of misassemblies and fragmented assemblies which results in low precision ([Cha+15]). The Bridger algorithms, introduced by [Cha+15], generates a splice graph from the RNA-seq data and uses a minimum path-cover model to enumerate transcripts. An extension of Bridger, BinPacker, was proposed by [Liu+16] uses the bin packing model without limiting the minimum number of paths to enumerate transcripts. However, it does not integrate paired-end information which leaves a large proportion of transcripts unassembled. TransLiG ([Liu+19]) progresses the algorithm of Bridger and BinPacker by integrating sequencing depth information and paired end information in its implementation. It generates a splice graph from the RNA-seq reads and from the splice graph iteratively constructs weighted line graphs and traverses paths in the graph to generate the final assembly. Figure 2.9 depicts the workflow of TransLiG. Below, we describe the algorithm of TransLiG. The description is based on the manuscript authored by [Liu+19].

- **Splice graph construction** (fig. 2.9a): TransLiG reuses the splice construction step of BinPacker. Given the RNA-seq reads, TransLiG selects the most frequent k-mer as the main contig of the initial splice graph and extends the ends by finding the most frequent unused k-mer which has k-1 overlap with the ends of the main contig. The main contig is extended till there are no unused k-mer which can extend the contig. For each k-mer in the splice graph, TransLiG checks whether there is an alternate extension that is not present in the main contig. If present, TransLiG terms these k-mers as *bifurcation k-mers*. The bifurcation k-mer is extended (using the above procedure) until either an already used k-mer is encountered or no further extension is possible. In the case of the former, the splice graph is modified by adding an edge from the bifurcation k-mer to the k-mer in the main contig which matched the bifurcation branch. If the latter case is encountered, i.e, if there are no unused k-mer available for extending the bifurcation branch, then the branch is kept after checking its validity using the paired-end information. Each edge of the splice graph is weighted by the number of reads supporting the edge.

- **Pair supporting paths** (fig. 2.9b): If the paired end information is available, the splice graph is converted into a pair-supporting path and a set P_G of such paths is maintained. Considering a pair (r_1, r_2), if r_1 spans a path p_1 of G and r_2 spans a path p_2 of G, TransLiG checks if there is a path p_c connecting the last node of p_1 to the first node of p_2. If such a path exists, then the path $p_f = p_1 \rightarrow p_c \rightarrow p_2$ is considered as pair-supporting path and is added to P_G. Different reads may generate the same pair-supporting path. So, each pair-supporting path is assigned a weight, $cov(P)$, which is equal to the number of reads generating the pair-supporting path.

- **Line graphs and iterations** (fig. 2.9c): TransLiG converts each splice graph into a line graph. For each splice graph $G = (V, E)$, a line graph $L(G)$ of G is a directed acyclic graph where nodes u, v are edges of G. The nodes u and v are connected if they share an node in G. For instance, consider three nodes in G namely a, b and c with edges connecting a to b and b to c. In the line graph $L(G)$, the edges (a, b) and (b, c) would correspond to the nodes u and v. The nodes u and v would be connected as they have a common node of G which is b. The line of graph of n^{th} order is defined by $L^n = L(L^{n-1}(G))$ where $L^0(G) = G$. Each isolated node generated during the line graph iteration can be expanded into a transcript representing path.

Starting from $L^0 = G$, TransLiG goes through all the nodes of the line graph and at each node v, minimizes the deviation between the weights of in-coming and the out-going edges of v. This is done to find the correct connections in the line graph which would represent a transcript. Based on the found connections, TransLiG generates the line graph $L^1 = L(L^0)$. In the next iteration, L^1 is taken as the input to generate $L^2 = L(L^1)$ and so on. The iterations are continued till a line graph L^n is formed where all the nodes are isolated, i.e, they cannot be connected by an edge.

- **Transcript generation** (fig. 2.9d): Transcripts are generated by expanding the isolated nodes of the final line graph L^n.

KREATION: informed parameter selection of multi *k*-mer based assembler

As mentioned repeatedly in the above sections, an assembly generated using multiple *k*-mer sizes always outperforms assembly using a single *k*-mer size. But selecting an appropriate *k*-mer size is still a challenge. Practitioners either use the default *k*-mer size set by the assembler or run the assembly over the entire set of possible *k*-mer sizes. The problem with the former approach is that the algorithms are tested on a limited set of data. Since the assembly algorithms are mostly heuristics, a single *k*-mer range is barely suitable for all datasets. The problem with the second approach, which is using the entire set of *k*-mer sizes, results in generation of misassemblies and a massive consumption of computational memory and runtime. At present, only a few algorithms such as velvetoptimizer ([ZB08]) and *k*mergenie ([CM14]) predict an optimal *k*-mer size for assembly. However, these algorithms are designed on the specifics of genome sequencing and hence are not suitable for transcriptome assembly. To our knowledge, there is only one RNA-specific algorithm named KREATION ([DS16]) which predicts the upper bound of a *k*-mer range for de Bruijn graph based assemblers. We have integrated KREATION in our pipeline to streamline de novo transcriptome assembly which we will describe in chapter 4. Here, we describe the procedure for KREATION which consists of the following steps.

1. *Estimating the contribution of a k-mer size:* Given a set of reads R and an initial *k*-mer sizes k_1, three single *k*-mer assemblies are generated using the *k*-mer size k_1, k_2 and k_3 where $k_{i+1} = k_i + s$ (s is a step size given by the user). Let $C(k_1)$, $C(k_2)$ and C_{k_3} be the contigs sets generated by the assembler. In order to quantify the useful sequence added in the iteration k_2 as compared to the iteration using k_1, KREATION clusters $C(k_1)$ and $C(k_{i+1})$ using the program CD-HIT-EST to extract the number of extended sequences. An extended sequence is representative sequence of a cluster which is a member of C_2 and completely overlaps at least one sequence from C_1. It has been shown in the manuscript of [DS16] that the extended sequences are mostly composed of correctly assembled transcripts and hence can be considered as a measure to the importance of a *k*-mer iteration.

2. *Predicting the upper bound of k-mer range:* The number of extended sequences declined with the increase in *k*-mer size in an exponential manner. This trend becomes linear when the numbers of extended sequences are converted into a log scale. KREATION calculates the number of extended clusters from iteration k_2 and k_3 and plugs them in a linear regression model to calculate the

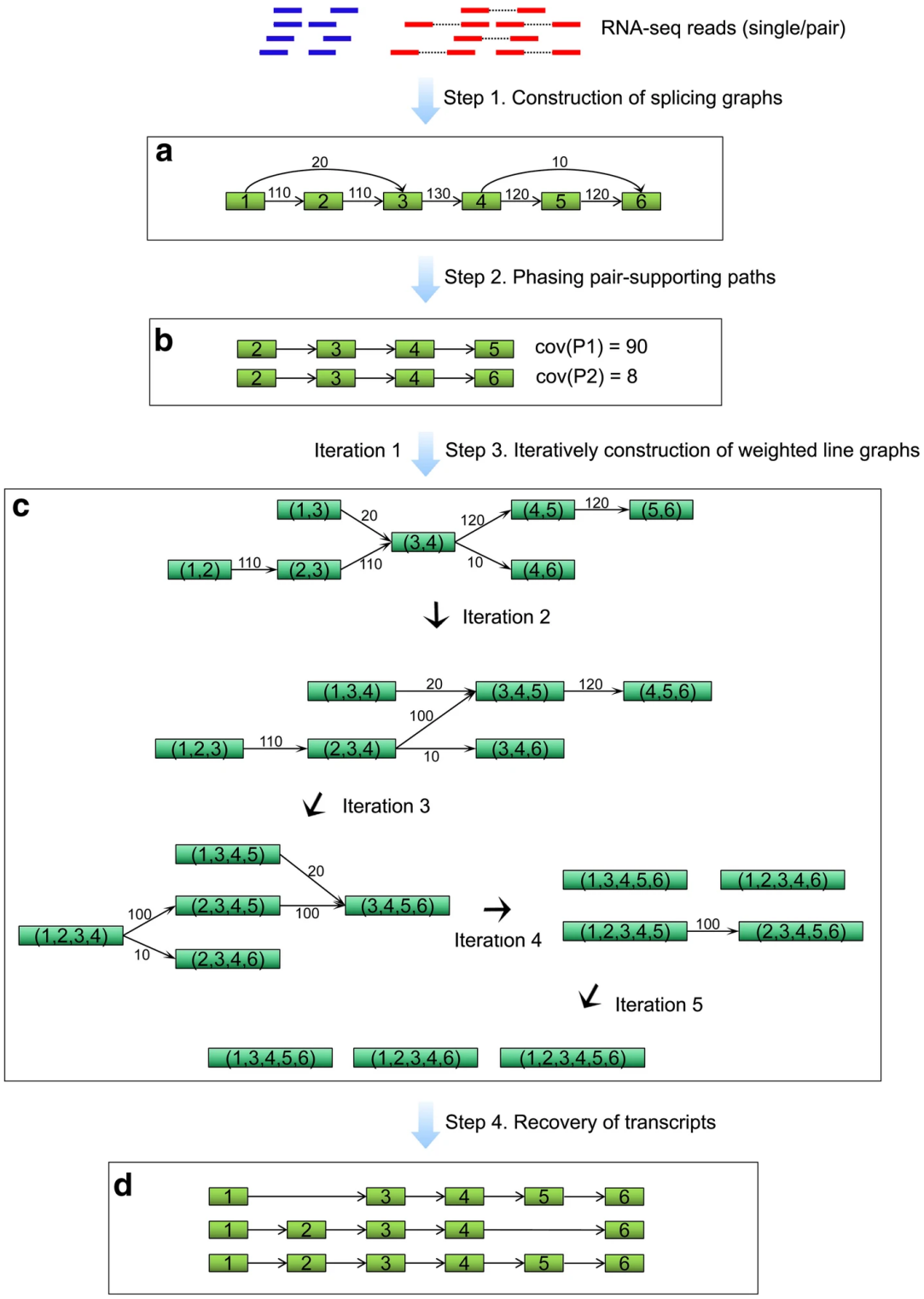

FIGURE 2.9: Workflow of TransLiG: a) Splice graphs are generated from RNA-seq reads. The algorithm similar to BinPacker is used here for the construction of the graphs. b) From the splice graphs, pair-supporting paths are extracted. c) Line graphs iteratively till a point where all the nodes of the line graph are isolated. d) The isolated nodes are expanded to generate transcripts. Figure taken from [Liu+19]

regression coefficients. Based on the predicted coefficients, KREATION predicts the number of extended cluster, say ex_p from the assembly using the next k-mer size (k_4). Assembly is generated using the k-mer size k_4 and the resultant contig set C_4 is clustered with C_1, C_2 and C_3 to calculated the actual number of extended sequences. If the difference between the actual number ex_a and the predicted number ex_p, also known as d_score, exceeds a user defined threshold, the assembly algorithm is stopped otherwise the whole process is repeated for a k-mer size k_5 and beyond.

KREATION was shown to accurately predict the upper bound for various transcriptome assembler. It was shown to reduce the number of misassembled transcripts in the final assembly and there was a significant reduction in the runtime of the assembler.

2.3.3 De novo transcriptome assembly evaluation

As can be noted from the above section, each assembly algorithm has its own heuristics for transcriptome assembly. Moreover, all the assembly algorithms have their own parameter settings which can be tweaks as per the requirements of the users. As a result. different assemblies can be generated from the same read dataset by either using a single assembler (by varying the parameter settings) or using multiple assemblers. Given such diversity, there is a constant need to access the quality of a de novo assembly and learn about the algorithm and/or parameter settings which best reconstructs the transcriptome. Commonly used evaluation statistics include median contig length, number of contigs and N50 ([KB10; Sch+12a]). But in the case of transcriptome assembly, these metrics are often misleading ([OE13]). For instance, N50 denotes the longest length x of a contig such that 51% of the assembled contigs are longer than x. In case of genome assembly, higher the value of N50 better the quality of the assembly. However in case of transcriptome assembly, an assembled contig should reflect the genes isoform which might be short in length. Hence, a high N50 does not always imply a good quality assembly.

Current algorithms like REF-EVAL ([Li+14]) and RNAQuast ([Bus+16]) align the assemblies against the provided reference genome and calculate the sensitivity and specificity at nucleotide level. Similarly, [Sch+12b], [Rob+10] and [Gra+11] in their work aligned assemblies against a reference genome using Blat ([Ken02]) and calculated the nucleotide sensitivity, nucleotide specificity and the number of full length transcripts assembler. BUSCO ([Sim+15]) aligns the assemblies against a set of known orthologous genes to access the completeness of the assembly. Reference free algorithms such as RSEM-eval ([Li+14]) and TransRate ([SU+16]) estimate the assembly likelihood given the read dataset. The algorithms REF-EVAL and RSEM-EVAL is combined into a single software package known as DETONATE ([Li+14]).

Throughout this work, we have used the metric of nucleotide sensitivity, specificity and the number of full-length transcripts assembler. In addition to this, we also evaluated our assemblies using the referenced based methods REF-EVAL and BUSCO which we describe below.

Nucleotide sensitivity, specificity and 100%-hits

Given a reference sequence, the assemblies are aligned again the sequence using a splice-aware aligner. The *nucleotide sensitivity* is defined as the proportion of reference genome which is correctly assembled in the transcriptome assembly. Similarly, the *nucleotide specificity* is the proportion of assembled transcripts which is correctly generated by the assembler. In other words, if n_r is the number of nucleotide bases in the reference transcriptome, n_a is the number of bases in the assembly and t_a is the number of bases correctly aligned to an annotated base in the assembled contig, the nucleotide sensitivity is calculated by the following formula

$$\text{sensitivity} = \frac{t_a}{n_r} \tag{2.1}$$

and the nucleotide specificity is calculated by

$$\text{specificity} = \frac{t_a}{n_a} \tag{2.2}$$

Further, the number of annotated transcripts which are completely overlapped by an assembled transcript is termed as *full length hits* or *100%-hits*. We will use these two terms interchangeably throughout this work.

REF-EVAL

REF-EVAL is a reference based assembly evaluation algorithm which is the part of the DETONATE package. Both, RSEM-EVAL and REF-EVAL of the DETONATE package work with *true assembly*. True assembly is a set of transcript sub-sequences which is covered by reads having at least w overlapping bases where w is a user defined non-negative integer. In other words, if r_i denotes a read i which aligns to a transcript $t(r_i)$ from the position $x_i(t)$ to $y_i(t)$ in $t(r_i)$ then, a transcript sub-sequence T is a true assembly if their exists a set of n reads $R = \{r_1, r_1, ..., r_n\}$:

1. $\forall i < n, \quad t(r_i) = T$
2. $\forall r_i \in R, \quad y_i(t) - x_{i+1}(t) \geq w$
3. T is not subsumed in any other true assembly

REF-EVAL estimates the true assembly using the following procedure:

1. The reads are aligned against a reference sequence using a short read alignment program. For each alignable read, REF-EVAL samples an alignment using a posterior probability that it is a true-alignment. The posterior probability is calculated using RSEM-EVAL.
2. REF-EVAL considers the sampled alignment as true alignment and builds true assembly according to it.
3. If the member of the paired-end reads spans multiple true assemblies, the assemblies are joined with the distance determined by their position in the parent transcript.

REF-EVAL performs a bidirectional alignment of assembled transcripts against the estimated true assemblies and calculates the nucleotide and contig level precision and recall. A Recall is the fraction of the reference element (contigs or nucleotides) correctly recovered in the assembly. Precision is the fraction of the assembly which correctly reflects the reference element. REF-EVAL also output the F1 score which is the harmonic mean of precision and recall.

BUSCO

Algorithms such as REF-EVAL and RSEM-EVAL measure the fraction of sequences correctly assembled. However, they are limited to the species which are sequenced and annotated in a database. With a vast number of species being sequenced in the current times, the knowledge of the orthologous gene content can be used to develop an evolutionary measure. Orthologous genes are genes in different species which is a direct descent from a single gene in the last known common ancestor of the two species ([Fit70]). Databases such as OrthoDB ([Wat+13]) determine the orthology of a gene pair based on the bidirectional alignment of genes from different specie pairs. OrthoDB has identified orthologous genes from Metazoan, Vertebrate, Arthropod, and Fungal lineages and clustered similar genes using CD-HIT-EST ([Fu+12]). For each of these lineages, OrthoDB has also compiled a set of genes (abbreviated BUSCOs) which are representatives of an ortholog group and exist as a single-copy in at least 90% of the species belonging to the phylum.

To differentiate between the software package BUSCO and the set BUSCO compiled by OrthoDB, we will use the term S-BUSCO from hereon for the software package. S-BUSCO aligns the assemblies against BUSCOs from OrthoDB. Based on the alignments and a precalculated hidden Markov model profile from amino acid alignment, S-BUSCO determines whether an assembled transcript is a BUSCO or not. It divides the assembled BUSCOs into three categories namely complete, fragmented and missing. An assembled BUSCO is *complete* if its length is not more than a defined threshold length. In S-BUSCO, the threshold is different for different BUSCO group and is set to two times the standard deviation of lengths of the genes belonging to the group. If the complete BUSCO is present as multiple copies in the assembly then it is termed as *complete and duplicated* otherwise it is termed as *complete and single*. BUSCOs which are recovered partially are classified as *fragmented* and those which are not recovered are classified as *missing*.

Assemblers or assembly settings are compared in terms of number of complete BUSCOs assembled. A setup is said to outperform another if it assembles a higher number of complete and single BUSCOs.

2.4 Transcript quantification

Transcript quantification estimates the levels of alternate isoforms within a RNA-seq sample. It enables the detection of differences in the levels of isoforms under different conditions. This is especially useful if the aim is to detect biomarkers between diseased and the normal tissue sample. Various quantification softwares have been proposed in recent times. Read mapping based algorithms such as Cufflinks ([Tra+10]), RSEM ([LD11]), Kallisto ([Bra+16]), RNASkim ([ZW14]) and salmon ([Pat+17b]) calculate the probability of a read originating from a transcript

by maximizing the joint likelihood of read alignments based on distribution of transcript fragments. Cufflinks and RSEM map the reads using a short-read splice aware aligner such as TopHat and Bowtie.

However, base-by-base alignment of reads to transcripts is time consuming process and consumes a lot of memory. Moreover, it is not always necessary to know the exact alignment of reads against a transcriptome. Algorithms such Kallisto, RNASkim and Salmon use a pseudo mapping approach where instead of mapping a read to a transcript end-to-end, they approximately determine the position of a read within a transcript by matching k-mers from the read to k-mers from the transcript. This results in a drastic reduction of runtime. Throughout our work, we compared the expression estimates from assemblies generated using different assemblers and different parameter settings. For this purpose, we used salmon which we briefly describe below.

Salmon runs in three phases namely lightweight alignment to map the reads onto the transcript using a k-mer matching based approach, an online phase to estimate intial transcript expression and predict model parameters and an offline phase to refine the expression estimates.

1. *Lightweight mapping:* Given an input reference transcript or the assembled transcript set, salmon builds a suffix array and a BWT (Burrow Wheeler Transfor) based index. Given the reads and using the index, salmon calculates the set of Maximal Exact Matches (MEMs) between the reads and the transcripts. Based on the position of the MEMs with respect to the transcript, continuous chains of MEMs are joined together to estimate an approximate position of the read in a transcript.

2. *Online phase:* In the online phase, salmon uses the mapping and a variant of bayesian inference to infer initial transcript level abundance and learn the parameters for auxiliary models such as fragment length distribution. During this phase, salmon also builds a set of "rich" equivalence classes over the observed sequence fragments, which consists of any pair of sequence fragments which map to the same set of transcripts. This vastly reduces the data representation for subsequent inferences.

3. *Offline phase:* In the offline phase, salmon applies the standard Expectation maximization (EM) algorithm (or a variational Bayesian inference method) on the reduced data representation using equivalence classes to refine the initial estimation of transcript expression until a data-dependent convergence criterion is satisfied

Salmon has been shown to outperform other methods on several synthetic RNA-seq datasets ([Pat+17b; Zha+17]). Its innovation in terms of lightweight alignment, stochastic inference and reduced representation of datasets in terms of equivalence classes has drastic improvement the runtime of the algorithms. However since the mapping is only based on approximate positioning of reads on the transcript, it accuracy might drop for certain cases. Hence, salmon also gives an option of giving pre-aligned BAM files as input which directly feeds into the online phase.

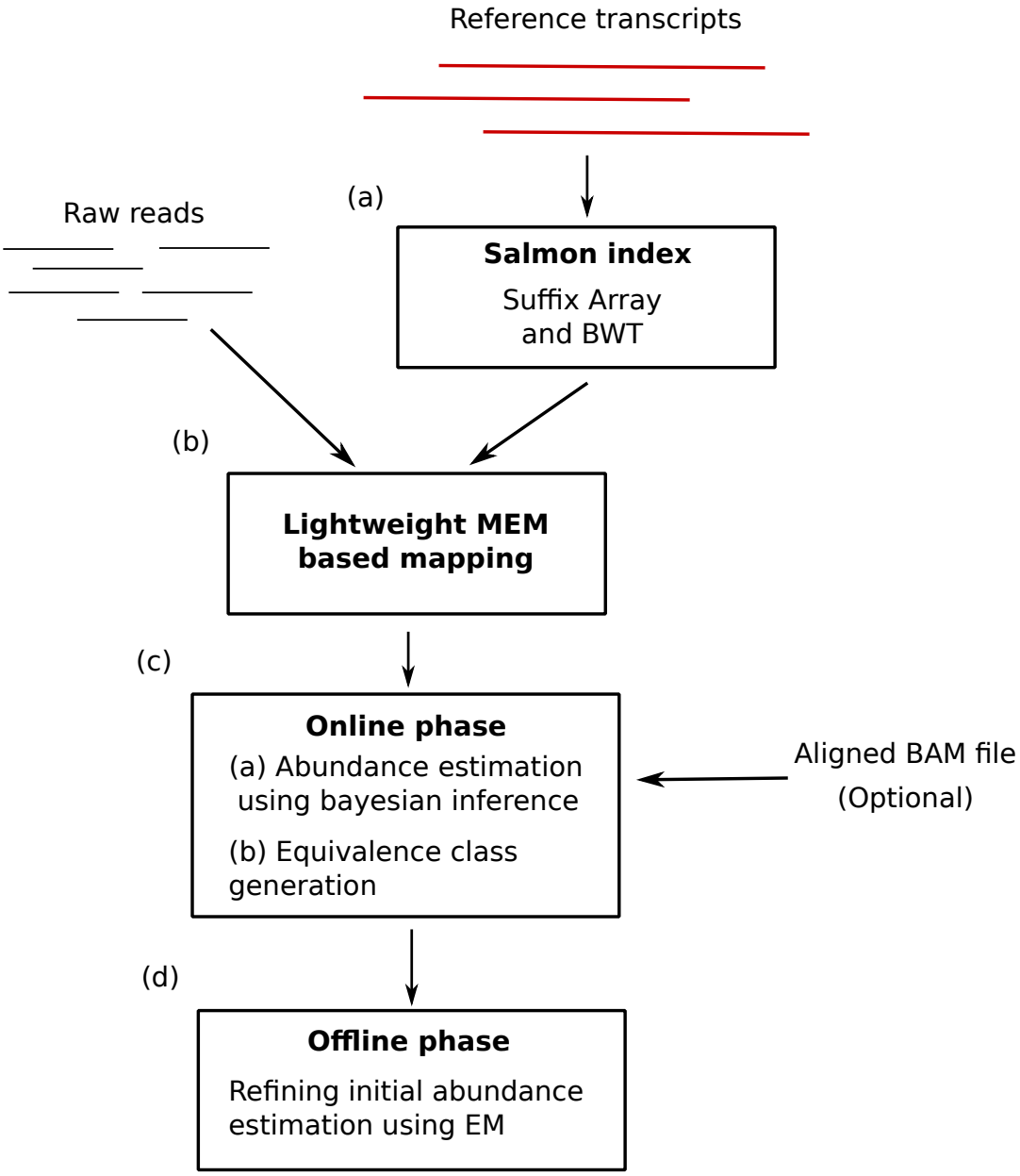

FIGURE 2.10: Workflow of the salmon. (a) An suffix array and BWT based index is generated from the reference transcript sequence or the assembly output. (b) Using the index, reads are mapped to the transcript set by finding the read's Maximal Exact Matches (MEM). Chains of sequential MEMs are joined together to estimate an approximate position of the read on the transcript. (c) Bayesian inference model is used to estimate an initial transcript abundance. Equivalence class is generated to reduce data representation. (d) Expectation maximization is applied to refine the intial transcript abundance. *Illustration inspired from [Pat+17b]*

2.5 Discussion

In this chapter, we introduced some of the most commonly used graph based assembly algorithms. All these algorithms start from the same basic step which is to construct a de Bruijn graph and traverse paths in the graph to generate the initial contig set. The difference between the algorithms comes from the procedure they use to construct the DBG and the heuristics they follow to process the contig set to generate the final transcriptome assembly. We also saw some of the pre-processing (error correction and normalization) and post-processing algorithms (quantification) which makes the overall process of assembly efficient. Here, we only discussed in detail the algorithms which is used throughout this work. However, there are many more pre-processing and post processing algorithms which were not discussed due to the limited scope of this thesis.

The field of transcriptome assembly is vast and constantly needs an upgrade. Although there are multiple transcriptome assemblers and they have been evaluated and compared in various studies, there is no optimal assembly tool which is suitable for all RNA-seq datasets. There are many factors which effects the performance of an assembly algorithm such different species, variation in sequencing protocol and different sequencing conditions. Combining assemblers from multiple assemblers seems to be a way forward as the disadvantages of one assembler can be negated by the other assembler ([JKP13]). However, this process needs to be further looked into. Most of the de novo assemblers suffer from a set of high memory requirements for storing the graph structure ([Zha+11; DS18]). There are constant updates made especially in the area of compressed de Bruijn graphs ([BO16; Tur+18]) which is applied for pan genomic analysis. Such updates needs to brought more in use for RNA-seq data. The current algorithms are also coming up their updates to make them memory efficient. The updates include implementation of efficient data structures to store k-mers, integration of modern graph traversal procedures and in-silico normalization of the input data. In the coming chapters, we will introduce an algorithm and discuss some methods which could further enhances the procedure of transcriptome assembly.

Chapter 3

In-silico read normalization using set multi-cover optimization

Last chapter introduced some of the transcriptome assembly algorithms which are considered to be state-of-art. We noted that, some of these algorithms require a reference transcriptome to function whereas some of them are de novo. As mentioned, the available computational resources, especially computational memory required by de novo assemblers, have not been able to catch-up with the high demands of assembling the high-coverage datasets ([DS16; DS18]). Practitioners are looking out for different methods and data-structures to make the process of assembly space-efficient for one or several datasets ([CLM16; Sze+17; Pel+12]). In this chapter, we look into the concept of data-reduction, also known as data normalization, and its effect on transcriptome assembly. Precisely, we try to answer the question "which parts of the data are being used by the assembler". Finally, we propose a set multi-cover based approach for normalizing a dataset that does not compromise on the structure of the *de* Bruijn graph produced from the data.

3.1 Current normalization approaches are sub-optimal

The previous chapter introduced read-normalization approaches namely Diginorm, Trinity's *in silico* normalization, and Bignorm. These approaches normalize datasets based on the average or median abundance of k-mers present in a read. For instance, Diginorm keeps or discards a read based on the median abundance of k-mers present in the read. Similarly, Trinity's In Silico (TIS) normalization calculates the median coverage of k-mers in the read and discards the read if the median is more than the desired coverage. Bignorm, an extension of Diginorm, incorporates a cutoff on the base quality values of reads to ensure high-quality reads in the final dataset. We direct the reader to chapter 2 for details of the above algorithms

These k-mer based approaches have three main advantages i) reads with high redundancy are removed which reduces the memory and runtime requirements of the assembly process, ii) erroneous k-mers might get removed in the process especially if the quality filter is applied, and iii) these approaches are fast and the memory requirements are low. Hence, scientists have considered including these algorithms as part of the pre-processing step. However, the above algorithms do not preserve the structure of the original DBG. These algorithms also remove a high percentage of low abundance k-mers especially from the read terminals ([Bro+12]). Some of the lost k-mers form connections between two sections of the de Bruijn graph (DBG). Hence, final assembly produced is short and fragmented. To be more clear in this aspect, let as assume that the DBG shown in fig 3.1 is generated from a sample dataset

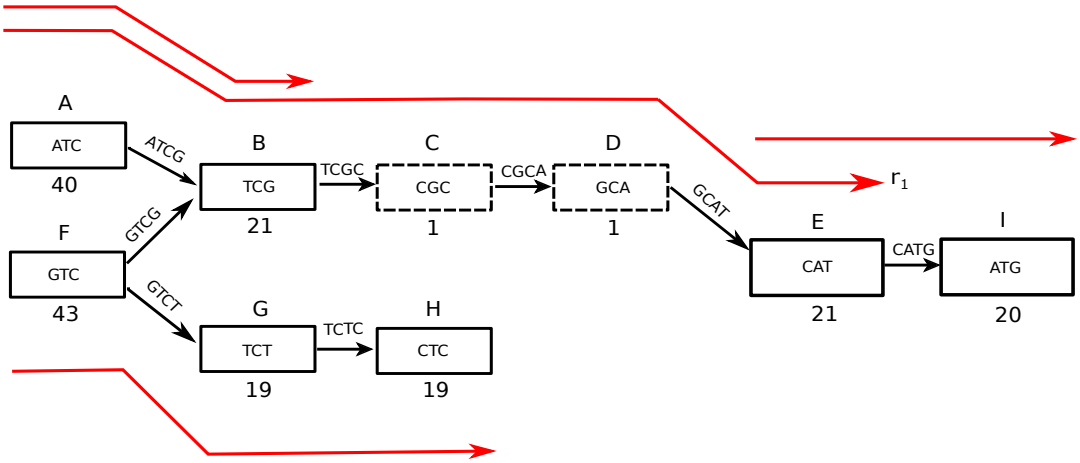

FIGURE 3.1: A toy de Bruijn graph with nodes representing k-mers and the number below every node represent the abundance of corresponding k-mer in the dataset. The solid red lines represents some of the reads which contribute k-mers forming the nodes. Read r_1 contributes k-mers to the node A, B, C, D and E. The dashed node C and D are the k-mers which get removed when read r_1 is discarded by a normalization algorithm. This results in the loss of connectivity between node B and E.

$\mathcal{R}$. The nodes of the DBG are 3-mers and the connections are labelled as 4-mers such that the source node is the prefix of the label and the destination node is the suffix of the label. The k-mers C (CGC) and D (GCA) have an abundance of 1 while its surrounding k-mers A (ATC), B (TCG) and E (CAT) have k-mer abundances greater than 20. So if we were to apply Diginorm on $\mathcal{R}$ with a median cut-off of 20, read $r_1 \in \mathcal{R}$ covering nodes A, B, C, D, and E and G will be removed. This would result in the loss of the dashed node C and D. The connection between the node B and E would be lost which would result in a fragmented assembly. Hence, it is can be argued that there is a value in retaining all k-mers from the original dataset.

But how can we reduce the dataset without losing any connections from the original graph? One possible solution would be to remove duplicated reads. But this strategy would affect the in-built error correction step of the assembler. For instance, in the toy example depicted in fig. 3.2, a bubble is formed due to an error in the read sequence. The error in the read sequence results in the formation of node E and node F. We see that the connection labels involving true k-mers (nodes B and C) have higher abundances as compared to the connections labels involving erroneous k-mers (nodes E and F) (fig.3.2a). To resolve this bubble, an assembler would convert node E and node F to nodes B and C respectively. In the final assembly step, a path through A, B, C and D (depicted as the red-dashed line in the figure) would be traversed. However, if we only keep 2 copies of each read, all the connections within the bubble would end up with the same abundance. When all the abundances are the same (fig 3.2b), the assembler would randomly choose a path between the two possibilities which might end in false positive transcripts.

A similar situation might occur for tip removal. An assembler removes tips from the graph if they have a lower abundance as compared to its neighboring nodes. However, normalization using the above method would result in similar abundances of k-mers in the region. This would generate contigs of shorter length.

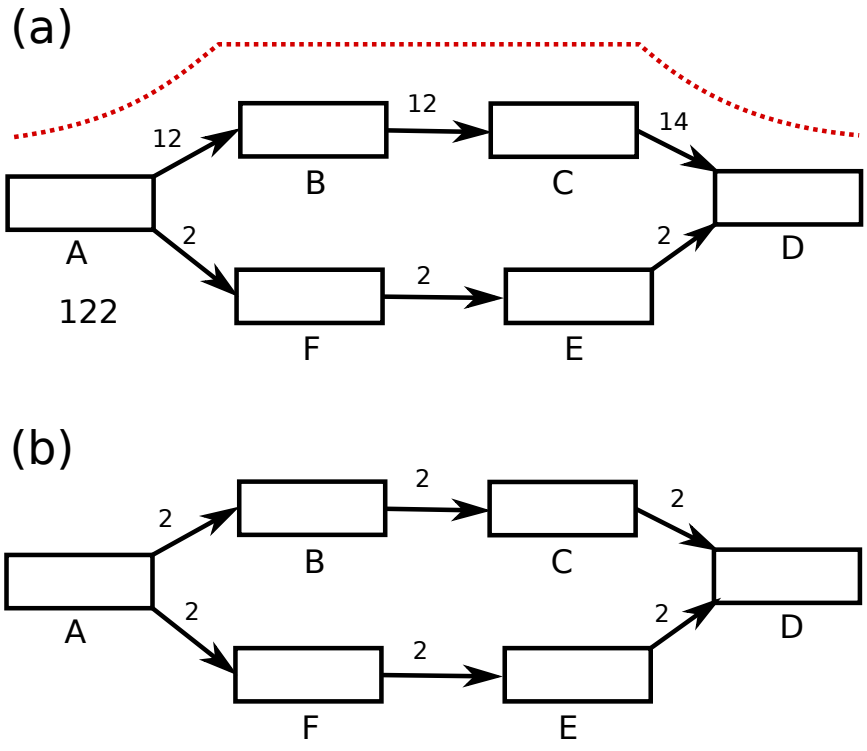

FIGURE 3.2: A toy de Bruijn graph (DBG) with a bubble formed due to error. The nodes of the graph represent k-mers from the dataset and the connections are represented as k'=($k+1$)-mers. The number above the connections represent the abundance of the k'-mer corresponding to the connection. (a) Error controlling step of an assembly algorithm where the path with the highest abundance is taken as the correct path (red-dashed line). (b) Modified DBG where only 2 copies of each read is kept.

3.2 Normalization as an optimization problem

In this chapter, we propose an Optimized Read Normalization Algorithm (ORNA) with the following two ideas:

- All the connection information from the unreduced dataset, which form the backbone of the DBG, should be covered in the normalized dataset. Like in the above example, if the nodes are formed by k-mers, we label the edges between two nodes as $k' = (k+1)$-mer such that the source node of the edge is the prefix of the k'-mer and the destination node of the edge is the suffix of the k'-mer. Hence, we aim to retain all the labels, i.e, k'-mer from the original dataset.
- The relative abundance difference between adjacent connections should be preserved.

Given a set of n reads, where each read consists of m k'-mers, we phrase normalization as a set multi-cover (SMC) optimization problem on reads. We suggest an $\mathcal{O}(nmlog(nm))$ time heuristics read normalization algorithm called ORNA which we describe below. But first, a brief introduction to set multi-cover (SMC) optimization problem is given.

3.2.1 Set multi-cover optimization

An important class of optimization problem which deals with covering objects with a certain set of characteristics is the set-cover optimization problem. In set-cover optimization problem, we are given a pair $\sum = \{X, S\}$ with the set $X = \{x_1, x_2, ..., x_n\}$ consisting of n objects and set $S = \{s_1, s_2, ..., s_m\}$ consisting of m sets. Each element of X can be mapped to at least one set in S. We term the set X as the *universe*. The *cover* of S, say S', is defined as a subset of S such that the union of sets in S' consists of all elements of X. For instance in fig 3.3a, the elements of X are represented as black dots and sets of S namely s_1, s_2, s_3, s_4 and s_5 are represented as rectangles. We notice that a union of s_4 and s_5 and a union of s_1, s_2 and s_3 would comprise of all elements of X (fig.3.3b). Hence, the *cover* of S would either be $S' = \{s_4, s_5\}$ or $S' = \{s_1, s_2, s_3\}$.

Based on the above definitions, we can formulate the *set-cover problem* as: Given a pair $\sum = \{X, S\}$ where X is the universe and $S = \{s_1, s_2, ..., s_m\}$ is a set of m sets such that each element of X can be mapped to a set in S, compute S' of minimum cardinality such that

$$\bigcup_{s \in S'} s = X \tag{3.1}$$

The set-cover (SC) problem is an NP-HARD problem with many approximate algorithms towards solving it with a different degree of completeness ([Kar72; CCS09]). A simple approach is the classical greedy algorithm for a polynomial-time approximation that iteratively chooses a set from S which contains the highest number of uncovered elements of X and adds them to S' ([Chv79]).

A variant of the set-cover problem is the *set multi-cover* (SMC) problem, where each element $x \in X$ should occur at least t_x distinct sets in the cover of minimum cardinality. The number of distinct sets in which element x should occur in the cover is termed as the *demand* of x. The SMC problem is formulated as computing $S' \subseteq S$ of minimum cardinality such that equation 3.1 holds and for any $x \in X$, there are at least $d(x)$ distinct sets in S' containing x.

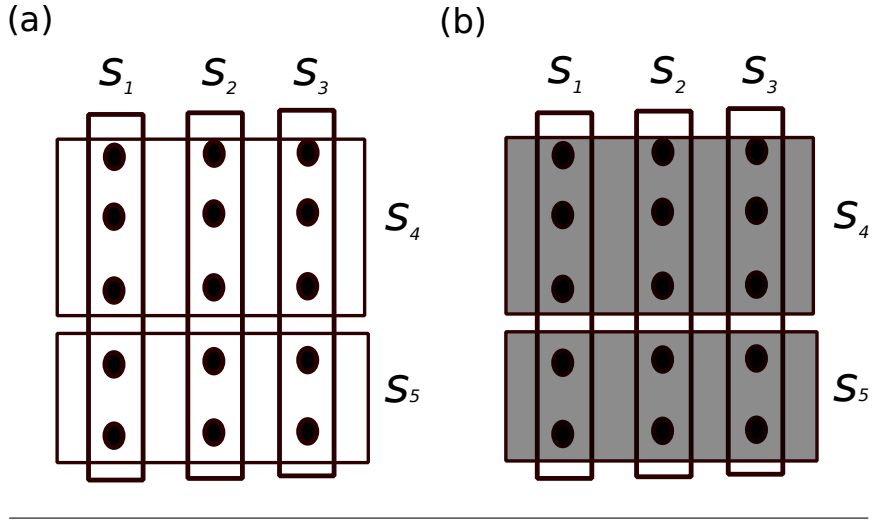

FIGURE 3.3: A toy example of a set-cover problem. (a) The elements of the universal set X are depicted with black dots and the sets are represented as rectangle. (b) A possible solution to the set cover problem where sets S_4 and S_5 (depicted as shaded boxes) cover the universe X.

3.2.2 Normalization as a set multi-cover problem

The set multi-cover optimization problem explained above can be extended to our normalization problem. We consider a dataset $R = \{r_1, r_2, \ldots, r_n\}$ as a set of n reads. Note, that a dataset might contain duplicated reads, i.e, two reads might have the same sequence. Each read is a sequence of DNA bases of fixed length a and would consist of a set of k-mers. As explained in the previous chapters, a de Bruijn graph is constructed using k-mers as nodes and two nodes are connected if they have a $k-1$-overlap. We label each edge with a unique label l which is equal to the $k' = (k+1)$-mer corresponding to the edge. The source node is a prefix of l and the destination node is a suffix of l. Conceptually, each label is generated from the reads in the dataset. Hence, each read $r \in R$ can be considered as a set of $p = a - k$ labels, i.e, $r = \{l_1, l_2, ...l_p\}$ where l_i is the k'-mer obtained from the i^{th} position of the read. The union of all the reads is equivalent to a set of all the labels present in the dataset. Intuitively, a k'-mer is composed of two k-mers (from the source and destination node). So, when a k'-mer/label is retained, the corresponding k-mers are also retained.

The objective of a normalization algorithm is to reduce the data as much as possible without having a significant impact on the quality of the assembly produced. As explained earlier, apart from keeping all the k'-mers from the original dataset, it is also important to preserve the relative abundance information of k'-mers. With these points in mind, we formulate the normalization problem as a slight variation of the set multi-cover statement as follows:

Instance: A dataset R of n reads, a set of labels L obtained from R such that $\bigcup_{r \in R} r = L$ and a demand t_l for every $l \in L$. Since, we aim to retain all the labels from R, t_l is always greater than 1.

Valid solutions: $R' \subseteq R$ such that $\bigcup_{r \in R'} r = L$ and $\forall l \in L, abund(l, R') \geq t_l$ where $abund(l, R')$ denotes the number of occurrences of l in R'. A read $r_i \in R'$ can have

multiple instances of a label l.

Objective:

$$\arg\min_{R'} \ |R'| \tag{3.2}$$

There are multiple approaches to solve the SMC problem. In a classical greedy approach, we start from a read which has the maximum number of uncovered labels (k'-mers) and iteratively go through the dataset R. The number of uncovered labels would change after every iteration. Hence, a data structure has to be maintained which would contain the reads in a decreasing order starting from the one which has the highest number of uncovered elements. This data structure has to be updated after every iteration. Given n reads each having m labels, the entire procedure would require $\mathcal{O}(n^2 mlog(nm))$ runtime which might not be efficient for large datasets.

3.2.3 Optimized Read Normalization Algorithm (ORNA)

Here, we propose a slightly different approach of the greedy algorithm. We summarize the approach in the form of an algorithm (algorithm 1). A read set R consisting of n reads and a k-mer size k are the inputs of the algorithm. As mentioned in the previous section, a read is a set of p labels. Hence, we first extract all labels or $k' = (k+1)$-mers from the reads and store them with their abundance information using the *BuildBloom(R,k')* function (line 3) in a data structure L. We can consider L as the universe as it contains all labels from the original dataset. It is very important to store them efficiently as it can blow up the memory required quite easily especially for huge datasets. With this point in mind, we store the labels in bloom-filters implemented using the GATB library ([CR13; Dre+14]). GATB uses the *BBHash* algorithm for building a *minimal perfect hash function* after counting the k-mers with the DSK algorithm ([RLC13]). For label counting and storing, ORNA requires $\mathcal{O}(nmlog(nm))$ time. We maintain an array *NodeCounter* for each label in the bloom filter which is initialized to zero (line 5). Entries in the array depict the number of times the label has been encountered during the iterations. This operation requires $\mathcal{O}(nm)$ time.

The dataset is iterated in the order of occurrence in the input file. For each read, we extract all the labels in the read using the $ObtainKmers(r, k')$ function and store it in a temporary array L' (line 8). For each label $l \in L'$, we calculate the corresponding demand using *ObtainDemand* function (line 11). This demand value corresponds to the number of times l should appear in the normalized dataset. We check the corresponding entry for l in our *NodeCounter* array and increment the value if the current value is less than the demand (line 12-15). We accept a read and transfer it to the pool of normalized dataset R_{out} if it contains at least one label for which the corresponding entry in the *NodeCounter* array is incremented (line 17-19). If no such case is found, we reject the read. The entire procedure of iterating the data and selecting the reads for the normalized dataset (line 7-20) requires $\mathcal{O}(nm)$ time. The overall time complexity of the algorithm is $\mathcal{O}(nmlog(nm))$.

An important parameter for ORNA is the demand value t. This can be set to a fixed value, which means that all labels would have the same demand. But as mentioned earlier, it is important to maintain the relative difference in abundances between k-mers for the sake of error correction. We lose the relative difference in

Algorithm 1 Set multi-cover based approach for read normalization

Input: Reads R, k-mer size k
$k' = k + 1$
L = BuildBloom(R,k')
b = NumberOf$k$mers(L)
NodeCounter[1...b]$\leftarrow$0
$R_{out} = \phi$
for all $r \in R$ **do**
    L' = ObtainKmer(r,k', L) ▷ $L' \subseteq L$
    flag=0
    for all $l \in L'$ **do**
        t=ObtainDemand(l, R)
        if $NodeCounter(l) < t$ **then**
            NodeCounter(l) = NodeCounter(l) + 1
            flag=1
        end if
    end for
    if $flag == 1$ **then**
        $R_{out} = R_{out} \cup r$
    end if
end for
Output: Reads R_{out}

abundance if all the labels have a same demand. Therefore, we set the demand of each label equal to the log of the abundance of the label in R.

$$\forall l \in L, t_l = log_b(abund(l, R)), \tag{3.3}$$

where $abund(l, R)$ is the abundance of l in R. L is the universe consisting of all labels from the original dataset. A read $r_i \in R'$ can have multiple instances of l. The log function used in the formulation has two advantages: 1) the labels having a high abundance in R would be reduced significantly in R' whereas the labels having low abundance in R would have a similar number of occurrences in R' and 2) the relative abundance differences between adjacent k-mers would be maintained. The level of the demand can be varied by varying the base (b) of the logarithm function which would in turn vary the level of reduction.

3.2.4 Incorporating quality and k-mer abundance information into ORNA

ORNA retains all k-mers from the original dataset which includes true as well as erroneous k-mers. One way to identify erroneous base, as done in Bignorm, is to use Phred score. A Phred score is inversely proportional to the probability of the base being incorrectly called by the sequencer. So a low Phred score means that there is a high chance that the base is most likely an error. In a FASTQ file, Phred scores are encoded as ASCII characters along with the read sequence. Another indication that a read contains erroneous bases is the distribution of k-mer abundance along the read sequence. The k-mers containing erroneous bases generally have lower abundance as compared to its neighbouring k-mers. This information was also used in

Diginorm and TIS.

ORNA iterates reads in a file in the same order by which it was generated by the sequencer. This might result in some high-quality reads, which are towards the end of the dataset, being left out of the final dataset. To avoid this situation, we propose two separate extensions of ORNA's SMC namely ORNA-Q and ORNA-K. ORNA-Q and ORNA-K assign a weight to each read of the input dataset based on either the Phred quality of bases in the read (ORNA-Q) or abundances of k-mers present in the read (ORNA-K). We consider the sum of the read-weights belonging to the original dataset R as the weight of the dataset $W(R)$. We obtain a normalized dataset R' from R, which fulfills the constraints of ORNA (equation 3.2) and at the same time, minimizes the overall weight $W(R')$ of R'. The weight of a read is set in one of the following two ways:

Base quality aware formulation: For a given read r_i of length s, let q_i^j represent the Phred score of r_i at position j. The *read quality score* qr is then defined as:

$$qr_i = \sum_{j=1}^{s} q_i^j \,. \tag{3.4}$$

Then, the *phred quality weight* qw_i can be defined as the inverse of read quality score:

$$qw_i = \frac{1}{qr_i} \,. \tag{3.5}$$

Label abundance aware formulation: For a given read r_i of length s, let l_i^j be label ($k' = (k+1)$-mer) starting at position j in r_i. The abundance of the label l_i^j in the original dataset is represented as a_i^j. Let $a_i = (a_i^0, a_i^1, ...a_i^{(s-k)})$ be the set of abundances of labels in r_i. We define the *read abundance score* kr_i as:

$$kr_i = median(a_i) \,. \tag{3.6}$$

We define the *label abundance weight* kw_i as

$$kw_i = \frac{1}{kr_i} \,. \tag{3.7}$$

Based on the above definitions of the weights, we formulate the extensions of ORNA's SMC as a *weighted set multi-cover problem* (WSMC):

Instance: A dataset R of n reads, a set of $(k+1)$-mers (defined as labels) L obtained from R such that $\bigcup_{r \in R} r = L$ and a threshold $t_l \geq 1$ for every $l \in L$.

Valid solutions: $R' \subseteq R$ such that $\bigcup_{r \in R'} r = L, \forall l \in L, abund(l, R') \geq t_l$ where $abund(l,')$ denotes the number of occurrences of l in R' and $t_l = log_b(abund(l,))$.

$$\textbf{Objective: } \arg\min_{R'} \; W(R') \tag{3.8}$$

In other words, we want to pick a subset R' from R that:

- retains all k-mers from the original dataset R a certain number of times (t_l)

- has the minimal read weight $W(R')$

As the WSMC problem is a generalization of SMC, it is also considered as NP-hard ([Var10]).

3.2.5 ORNA-Q and ORNA-K

Currently, the weighted set multi-cover problem (WSMC) is an area of active research. The classical greedy approach generally gives preference to sets having the maximum number of active elements and minimum weights. As in the case of SMC, a data structure has to be maintained to keep a track of sets having the maximum number of inactive elements and minimum weight. This data structure has to be updated after every iteration which makes the runtime of the whole process in polynomial order concerning the number of sets. Hence, for our scenario where we consider each read as a set, this approach is infeasible especially for large datasets.

Again, we follow a simplified version of the greedy algorithm where we reorder the reads only once using a counting sort based strategy. We use two approaches to keep track of weights in the dataset:

Phred quality based weight (ORNA-Q): Given a read dataset R with n reads (each of length s) as input, we first calculate the read quality score qr_i for a read i using equation 3.4. Let the value of the largest possible read quality score be denoted as qr_{max}. Hence, the value of qr_i would range from 0 to qr_{max}. We initiate an array T of size qr_{max} which is used to record the number of times a quality score is encountered in R.

Label abundance weight (ORNA-K): Given a read dataset R consisting of n reads (each of length s), we first calculate read abundance score kr_i for each read r_i using the equation 3.6. We divide the reads into d bins of size b based on their read abundance score. Similar to the Phred quality based weights, we maintain an array T of size d such that each index i in T represents a bin. $T[i]$ then records the number of elements in bin i. For instance, if the size of each bin is 1000, then say $T[10]$ contains the number of reads having label abundance weight between 9000 and 10000.

We proceed by performing a counting sorting step to sort reads by their read weights ([SC54]). Briefly, we create an array B of size n to store the ordered list of reads. We divide this array into f chunks where f is the number of non-zero entries in T. We iterate over reads in dataset R and calculate the read weight (qr_i or kr_i) of each read $r_i \in R$. We calculate the sorted position of r_i in B using T and store it to the corresponding chunk. We apply algorithm 1 to the sorted list of reads. In ORNA-Q, the input set of reads is stored in the order of decreasing Phred scores. Since, the read weight is inversely proportional to the quality score (equation 3.5), the algorithm will process a read which has the lowest weight in each iteration. Hence, there is no need to change the order of the reads after every iterations. Similarly, in ORNA-K, each iteration would process a read which has the lowest or close to the lowest weight, since the reads are sorted into a bin based on label abundances. We assume that the bin-size in ORNA-K is small as compared to the number of reads. Hence, we ignore the ordering of reads within the bin.

3.2.6 Normalization of paired-end data:

In the process of assembly, paired-end information is used to extend contigs as it can resolve longer repeats ([Mas+12]). Hence, it is important to retain the paired-end connection while normalizing the dataset. In ORNA, we do not care about the ordering of the reads in the datasets. In ORNA-Q and ORNA-K, we calculate *pair-score* by taking the sum of individual scores of the reads belonging to a pair. We calculate *pair-weight*, which is the inverse of the pair-score. We proceed by sorting the pairs in increasing order of pair-weights. The remaining steps are the same for ORNA, ORNA-Q, and ORNA-K. Briefly, we iterate the entire dataset once and check the acceptability of each read in the pair using algorithm 1. A pair is accepted if and only if both the reads satisfy the accepting conditions. In other words, both the reads should have at least one label which has not reached its demand in the normalized dataset. If only one read of the pair satisfies the condition, then the pair is put into a pool named *Marked*. The pair is rejected if both the reads of the pair fail the conditions. After the complete iteration of the dataset, there might still be labels that are not covered. Hence in the second stage, we iterate through the *Marked* pool again. We accept a pair from the pool if there is at least one read in the pair which fulfills the acceptance conditions.

3.3 Data retrieval and normalization

For the analysis of ORNA, we used seven different datasets. We downloaded two datasets from the SRA run browser namely - Brain ([BM+12], SRR332171) which consisted of 147M paired-end reads and hESC ([Au+13], SRR1020625) which consisted of 142M paired-end reads. Both the datasets consisted of 50bps long reads. The datasets were error corrected using SEECER version 0.2 ([Le+13], with default parameters). Further, we generated an *ENCODE dataset* of 883M reads by concatenating five individual ENCODE datasets: 101M paired-end reads from hESC (GSM758573), 192M paired-end reads from AG04450 (GSM765396), 207M paired-end reads from GM12878 (GSM758572), 165M paired-end reads from A549 (GSM767854) and 216M paired-end reads from HeLa (GSM767847).

For the analysis of ORNA-Q and ORNA-K, we reused the Brain dataset from above. The hESC dataset from above didnt have the proper Phred score information. Hence, we used 216M paired-end reads from the HeLa cell line (SRR317049). Reads error corrected by SEECER lose their quality score information. As far as we investigated, there were no methods to map the quality values from the original read to it corresponding error corrected version. Hence, we used the uncorrected version of both datasets.

Normalization: We normalized the error corrected Brain and hESC data, downloaded from the SRA browser, using ORNA, Diginorm and Trinity's In-Silico (TIS) normalization. For the analysis of ORNA-Q and ORNA-K, we normalized uncorrected Brain and HeLa datasets using ORNA-Q, ORNA-K, Diginorm, and Bignorm.

Since the heuristics of ORNA, ORNA-Q and ORNA-K are different from those of Diginorm, TIS and Bignorm, it is unfair to compare the algorithms using just one constant parameter setting. Hence, we normalized the dataset to different levels of reduction and compared the individual assemblies generated from them. In ORNA, ORNA-Q, and ORNA-K, we can control the level of reduction by tuning logarithm

TABLE 3.1: Parameters used for different normalization algorithms. ORNA requires base b of the logarithm function. Diginorm and TIS requires a coverage cut-off value c. The first column depicts the label (k'-mer size) used for the normalization algorithms. The second column contains the datasets used for the algorithms. A star next to the value indicates the default coverage/base of the algorithm.

k'-mer	Dataset	ORNA(b)	Diginorm(c)	TIS(c)
22	SRR332171	(1.3,1.5,1.7*,3,5,7,10)	(5,10*,15,20,25,30)	(5,10,15,20,25,30)
	SRR1020625	(1.3,1.5,1.7*,3,5,7,10)	(15,20,25,30,35,45,50,55,60,65,70)	(30,35,45,50*,55,60,65,70)
	ENCODE	5	10	10
26	SRR332171	(1.3,1.5,1.7*,3,5,7,9)	(5,10*,15,20,25,30)	(5,10,15,20,25,30)
	SRR1020625	(1.7*,3,5,7,9,15,25 ,35,100,200,300)	(1,5,10*,20,30,40,50,60,70,80)	(5,10,20,30,40,50*, 60,70,80,90,100,110)

base value (b). Similarly, in Diginorm and TIS, the level is controlled by the coverage cut-off c. We varied these parameters to obtain different levels of reduction. For Bignorm, we varied the quality cutoff parameter to get different levels of reduction. We report these parameter settings used for comparisons of ORNA in table 3.1 and the values used for the comparisons of ORNA-Q and ORNA-K in table 3.2.

TABLE 3.2: Parameters used for different normalization algorithms. ORNA-Q, ORNA-K and ORNA requires base b of the logarithm function. Diginorm and Bignorm requires a coverage cut-off value c and quality score cutoff q respectively. The first column contains the datasets used for the algorithms. A star next to the value indicates the default coverage/base of the algorithm.

Dataset	ORNA-K(b)	ORNA-Q(b)	ORNA(b)	Diginorm(c)	Bignorm(q)
SRR332171	(1.7*,3,5,7,9)	(1.3,1.7*,3,5,7,11)	(1.3,1.7*,3,5,7,9)	(5,10*,15,20,25)	(5,10,15,20*,25)
SRR317049	(1.3,1.7*,3,5,7)	(1.3,1.5,5,7,9)	(1.3,1.5,2,3,5,7,9)	(5,10*,15,20,25)	(1,5,10,15,20*)

3.4 Transcriptome assembly and evaluation

The idea of ORNA stems from the de Bruijn graph (DBG) based assemblers. Hence, we wanted to analyze the quality of the assemblies produced from the normalized datasets. For this, we applied three DBG based assemblers namely - Oases-M ([Sch+12b]), TransABySS ([Rob+10]) and Trinity ([Gra+11]) on our reduced datasets. For the analysis of ORNA-Q and ORNA-K, we normalized the uncorrected Brain and HeLa dataset and assembled it using TransABySS.

The assemblers were executed in their default settings except for the k-mer size. We ran the multi k-mer based assembler Oases-M with a k-mer size ranging from 21 to 49 having an increment of 2. We merged the assemblies generated from individual k-mer sizes using the Oases merge script. We ran TransABySS a single k-mer size of 21. Trinity uses k-mer size of 21 for all its assembly which we didn't change. By default, Trinity runs its normalization on the data before assembling them. We disabled this option to get the correct estimate of the assembly quality. The assembled transcripts were evaluated using the REF-EVAL program of the Detonate package ([Li+14], version 1.11). REF-EVAL quantifies the assembly quality in terms of nucleotide level precision, recall and F1 score which is the harmonic

mean of precision and recall. Nucleotide F1 score evaluates an assembly by comparing its coverage of nucleotides with the reference. However, it does not measure the assembly quality. To achieve the assembly contiguity, we aligned the assembled transcripts against a reference genome and matching the overlaps against ENSEMBL transcripts ([Cun+15], GRCh38). We obtained the number of ENSEMBL transcripts which overlapped completely at least one distinctly assembled transcript and termed it as *100%-hit transcript*. The number of 100%-hit transcripts obtained from an unreduced dataset was considered as *complete*. We evaluated the performance of a normalization algorithm in terms of *% of complete*. For example, if an unreduced dataset result in an assembly of 2000 *100%-hit* transcripts and assembling a normalized dataset produces 1000 *100%-hit* transcripts, then the normalized data has achieved *50% of complete*. When comparing normalization algorithm, we consider an algorithm A to be better than B if the *% of complete* of A is higher than B.

3.4.1 ENCODE dataset analysis

For the analysis of *combined datset* (see section: 3.3), we normalize the dataset using ORNA and assembled them using TransABySS with k-mer size=21. We term the generated assembly as *combined assembly*. The goal of the analysis was to obtain transcripts which would not have been assembled from the individual dataset. But how can we determine whether an assembled transcripts is only possible when multiple datasets are combined? For this, we used a technique introduced by [DS16]. We assembled the individual datasets (using TranABySS) and clustered the assemblies with the combined assembly. For clustering, we used CD-HIT-EST ([Fu+12]) with the sequence identity of 99%. If a transcript in the combined assembly is also assembled by any of the individual datasets, then it would be clustered with the sequences assembled by that dataset. Hence, we obtained all clusters which only contained sequences from the combined assembly and termed them as *missed clusters*. We termed the longest sequence from the cluster as *missed transcript*. All the missed transcripts were then compared against annotations from ENSEMBL ([Cun+15], GRCh38) and GENCODE ([Har+12], version 17).

3.4.2 Correlation analysis

To obtain the gene-expression estimates from assemblies, we used Salmon ([Pat+17b], v0.8.2) with k-mer size set to 21. Salmon provides transcript level quantification in terms of Transcripts Per Million (TPM). To obtain the gene-level quantification, we summed up the TPM values for all transcripts belonging to a gene. We calculated the Spearman correlation between the gene-expression estimates from reduced datasets against estimates from the original datasets.

3.5 Results

3.5.1 ORNA retains all k-mers and their relative difference in abundance from the original dataset

An important parameter for any assembler is its k-mer size. As previously mentioned, a small k-mer size would assemble transcripts from low expressed regions of the transcriptome. However, a k-mer size too small would leave the repeat regions unresolved. Also, the assembly would produce a high percentage of false positives

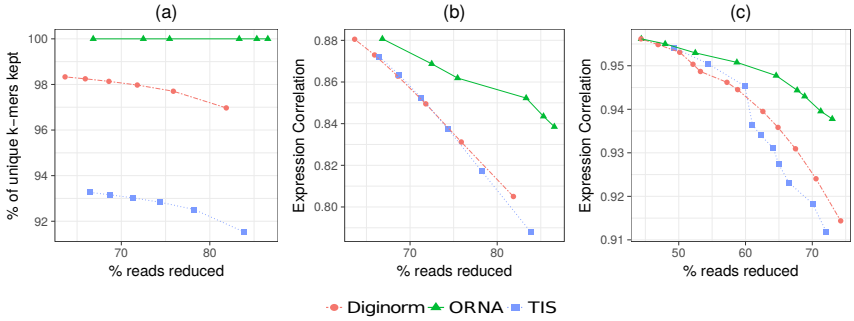

FIGURE 3.4: Information retained from the original dataset in the reduced datasets. (a) percentage of unique k-mers retained (y-axis) from the original dataset compared to the level of reduction by varying the normalization algorithm parameters (x-axis). Spearman's rank correlation values for gene-level TPM values obtained from the assembly of unreduced and reduced (b) Brain dataset and (c) hESC dataset

as there would be too many false connections in the DBG ([Sch+12b]). On the other hand, a k-mer size too high would result in a fragmented assembly. For the assembly of our reads having a read length of 50bps, we chose a k-mer size of 21 which was previously shown to produce a high percentage of correctly assembled transcripts ([DS16]) for our tested datasets. The nodes of the DBG thus generated by the assemblers would have a length 21 and the label (k'-mer) of the edge connecting two nodes would have a size of 22. Since the aim of our algorithms is to retain all the labels from the original dataset, we first obtained all the 22-mers from the original and reduced datasets. Figure 3.4a shows the retention of all 22-mers in reduced Brain datasets obtained from various normalization algorithms. We see that for any level of reduction, ORNA retains all the 22-mers from the original dataset. Diginorm loses nearly 2% and TIS loses nearly 8% of the 22-mers from the original dataset. ORNA treats the sets of all 22-mers present in the original data as the universe. It then selects the minimum number of reads, which is required to cover all the elements of the universe a certain number of times. Hence, all 22-mers or edge labels from the original dataset are retained.

Another important goal of ORNA is to maintain the relative difference in abundance between k-mers. The in-built error correction step of the assembler uses the relative difference in abundances to resolve bubbles and tips. We maintain the relative difference by keeping different abundance thresholds for each label or k'-mer present in the dataset. To analyze the effect, we compared the expression estimates from ORNA reduced datasets against the estimates produced from Diginorm and TIS reduced dataset. Figure 3.4b and c show the correlation between TPM values obtained from reduced datasets against the TPM values obtained from the original data. We see that the spearman correlation values for ORNA are always similar or higher than the values obtained using Diginorm and TIS reduced datasets. This is because ORNA can produce more number of correct transcripts as (a) it retains all the k-mers from the original dataset and (b) it maintain the relative difference in abundances between k-mers better than Diginorm and TIS.

3.5.2 ORNA produce better quality assemblies

For accessing the quality of the transcripts produced from the normalized datasets, we used REF-EVAL from the Detonate package ([Li+14]) to calculate nucleotide precision, recall and F1 score. Further, we calculated the number of *100%-hit* transcripts from all the assemblies. We evaluate the performance of a normalization algorithm in terms of *% of complete* where *complete* is the number of 100%-hit transcripts obtained from unreduced dataset (see section 3.4). We compare the performance of ORNA against TIS and Diginorm using a *k*-mer parameter value of 22 with assemblies generated using Oases and TransABySS on Brain and hESC dataset. For assembly using Trinity, we use a *k*-mer parameter value of 26 as trinity use only *k*-mer size=25 to construct its DBG. It was our intuition that the three algorithms would behave differently with respect to the number of reads reduced in a data dependant manner since the heuristics followed by the algorithms are different. Hence, we varied the parameters of all algorithm and obtained various normalized datasets.

We assembled these normalized datasets using Oases, Trinity and TransABySS. As shown in fig 3.5 from brain dataset and from fig 3.6 for hESC dataset, with higher reduction values, the nucleotide recall obtained from the corresponding assemblies reduced for all three assemblers especially at higher levels of reduction. Nucleotide recall is calculated by dividing the number of correct nucleotides assembled by the number of total nucleotides present in the reference. At higher levels of reduction, the number of transcripts assembled reduces which, in turn, reduces the correctly assembled nucleotides. As a result, the nucleotide recall reduces. But this drop is negated with the increase in nucleotide precision making the overall F1-score for all assemblies relatively stable as seen in table 3.3. In most cases, assemblies generated from ORNA datasets had slightly better F1 score as compared to other normalization algorithms except for Oases assemblies of the Brain datasets.

TABLE 3.3: Comparison of average F1 scores using REF-EVAL. Each entry in a cell denotes the average of F1 scores obtained by assembling Brain and hESC datasets normalized by the three algorithms (ORNA, Diginorm and TIS) in the rows. Averages are taken over results obtained with several parameters for each algorithm. The F1 score obtained for the original (unreduced) dataset is shown in the first row. Columns denote the assembler used. The highest mean obtained by any normalization algorithm is underlined.

Method	Brain			hESC		
	Oases	TransABySS	Trinity	Oases	TransABySS	Trinity
unreduced	0.402	0.441	0.414	0.304	0.577	0.621
ORNA	0.404	<u>0.440</u>	<u>0.421</u>	<u>0.302</u>	<u>0.582</u>	<u>0.616</u>
Diginorm	0.411	0.418	0.419	0.280	0.578	0.601
TIS	<u>0.419</u>	0.437	0.413	0.283	0.579	0.599

However, the F1 score does not capture assembly contiguity and hence we investigated the number of full-length assemblies (100%-hits). In fig 3.8, we compare the amount of read reduction (x-axis) against the assembly performance as *% of complete* (y-axis). In general, the quality of the assembly remains at lower levels of reduction (50%-70%) but starts degrading at a higher percentage of reduction (70%-90%). However, for Brain datasets, assemblies generated from ORNA datasets perform better as compared to assemblies generated from Diginorm and TIS at a higher percentage of reduction. In other words, assembly of ORNA reduced datasets retains equally or more 100%-hit transcripts for all. In the case of hESC, we see similar

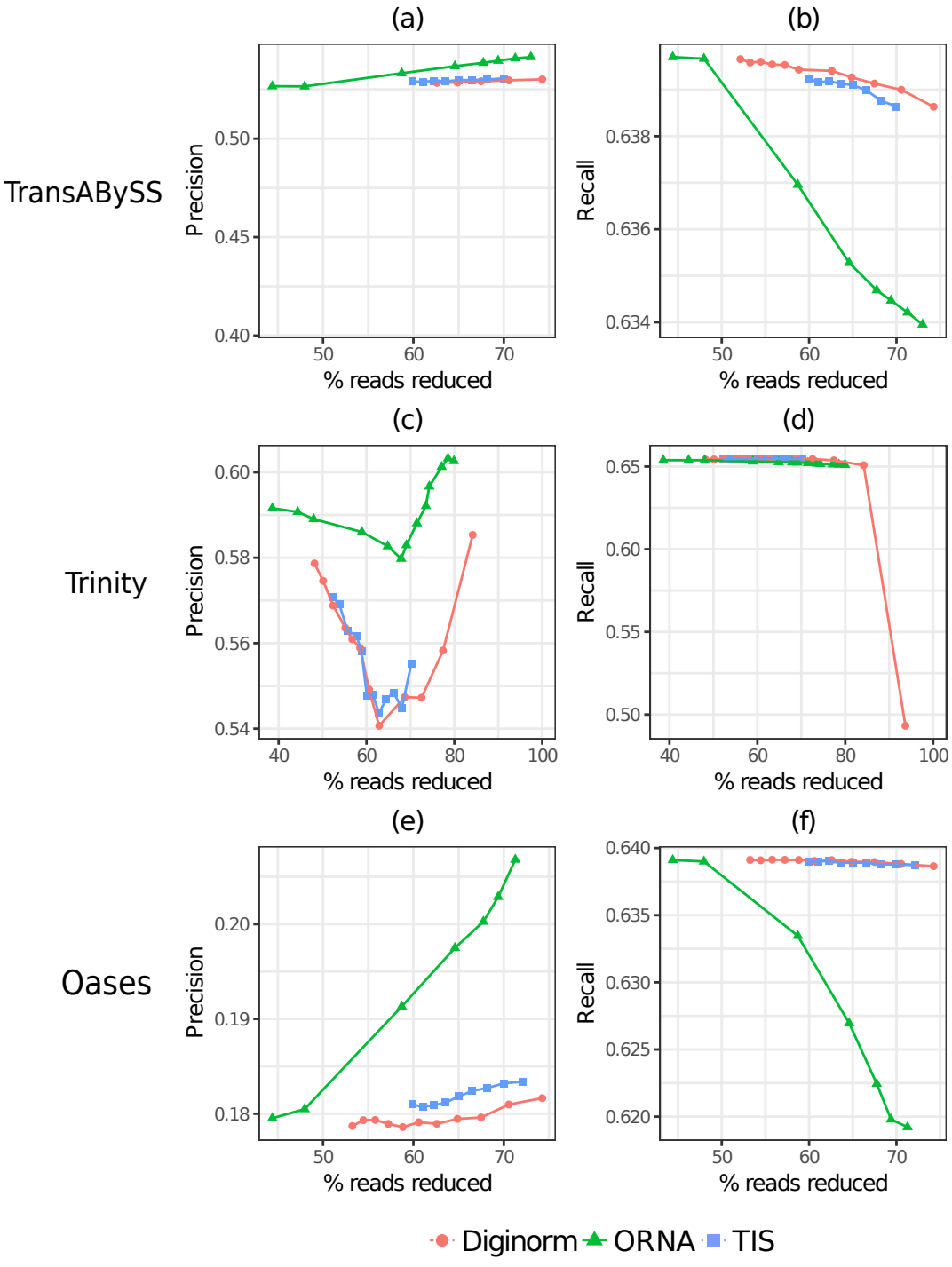

FIGURE 3.5: Comparison of precision (first column) and recall (second column) scores obtained from assemblies of ORNA, Diginorm and TIS reduced Brain dataset. Each point on a line corresponds to a different parametrization of the algorithms. The amount of data reduction (x-axis) is compared against the precision/recall measured from REF-EVAL (y-axis).

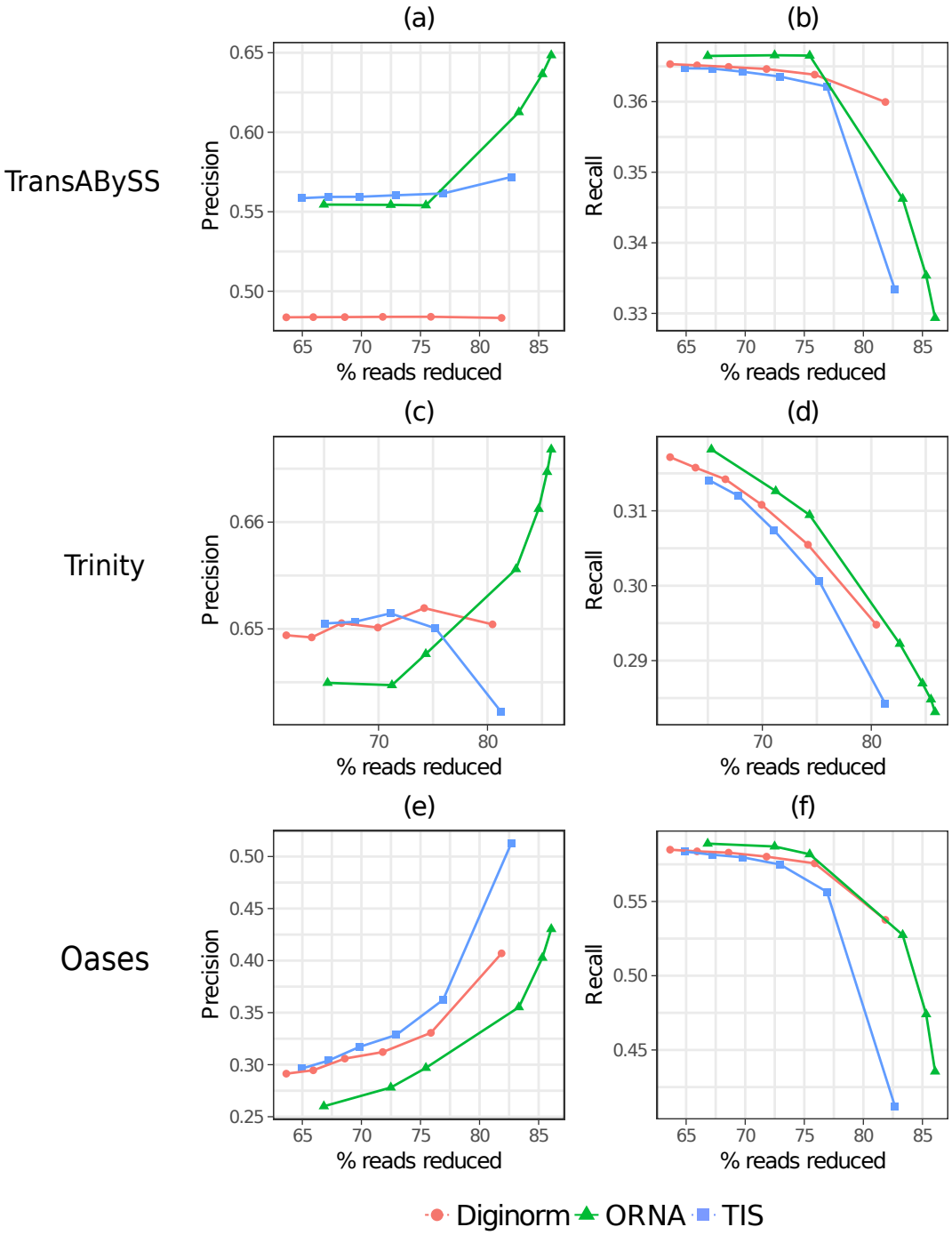

FIGURE 3.6: Comparison of precision (first column) and recall (second column) scores obtained from assemblies of ORNA, Diginorm and TIS reduced hESC datasets. Each point on a line corresponds to a different parametrization of the algorithms. The amount of data reduction (x-axis) is compared against the precision/recall measured from REF-EVAL (y-axis).

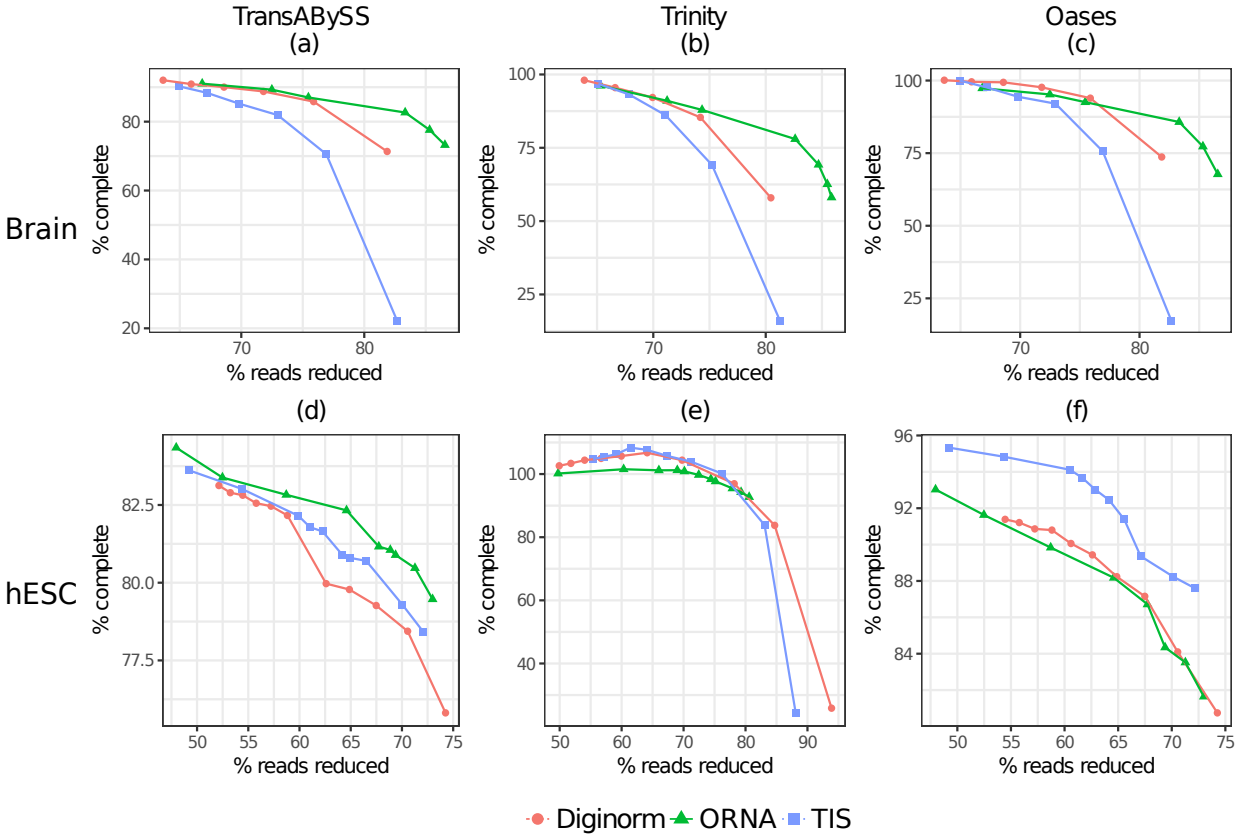

FIGURE 3.7: Comparison of assemblies generated from ORNA, Diginorm and TIS reduced datasets. Each point on a line corresponds to a different parameters of the algorithms. The amount of data reduction (x-axis) is compared against the assembly performance measured as *% of complete* (y-axis, see text). (a) and (d) represent TransABySS assemblies (k=21) applied on normalized Brain and hESC data, respectively. (b) and (e) represent Trinity assemblies (k=25) and (c) and (f) represent Oases multi k-mer assemblies applied on normalized Brain and hESC data, respectively.

behavior in assemblies generated using TransABySS. However, in the case of Trinity assemblies, some of the reduced datasets gave more 100%-hits than the original assemblies with TIS performing slightly better than the rest. For hESC assemblies generated using Oases, we noticed that TIS was performing much better with Diginorm and ORNA performing similar to each other. The difference in performance between Oases, Trinity and TransABySS assemblies from the hESC dataset underlines that data reduction shows varying effects on assembly contiguity depending on the assembler used.

For evaluating ORNA in paired-end mode, we normalized Brain dataset using ORNA, TIS, and Diginorm. We ran all the normalization algorithms in paired-end mode. We assembled the normalized dataset using TransABySS. We noticed a similar behavior to the single-end mode. At a higher percentage of reduction ORNA reduced datasets produced more number of 100%-hit transcripts as compared to TIS and Diginorm reduced datasets (fig. 3.8).

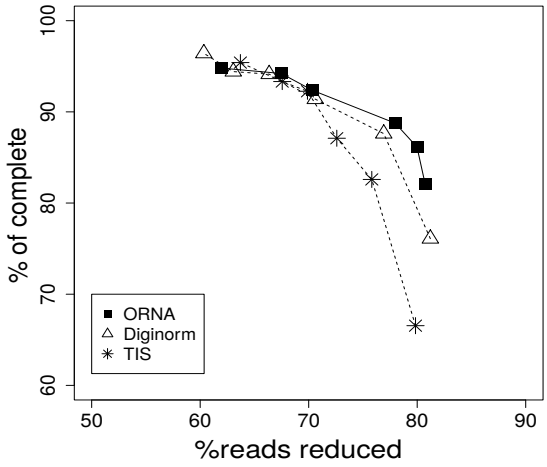

FIGURE 3.8: Comparison of assemblies generated from ORNA, Diginorm and TIS reduced Brain datasets. The normalization algorithms were run in paired-end mode. Each point on a line corresponds to a different parameters of the algorithms. The amount of data reduction (x-axis) is compared against the assembly performance measured as *% of complete* (y-axis, see text).

3.5.3 Finding novel transcripts by joint normalization of large datasets

Detection of novel transcripts from assemblies has been an active area of interest for many researchers. Practitioners generally combine multiple assemblies and either align them to a reference genome or calculate the protein-coding potentials of the assemblies ([Gil13; CJ16; Ven+18]). Here, we propose an alternative method for the detection of novel transcripts where we combine multiple RNA-seq datasets and assemble them. We use five diverse datasets to form an ENCODE dataset of 883M reads (see section 3.3). Assembling the ENCODE dataset required 357GB of RAM and 173hrs of runtime to produce the final assembly. Hence, we first normalized the ENCODE dataset using ORNA in single-end mode and assembled them. The runtime and memory requirements for the process of assembly got reduced by nearly 90%. We obtained the *missed transcripts* (see section 3.4). We aligned the missed transcripts against a reference genome and compared the alignment with annotated Ensembl and GENCODE annotation to obtained the bio-types of the missed transcripts. We were able to successfully assemble 381 protein-coding transcripts which would not have been assembled using the individual datasets. Along with these, the set of missed transcripts consisted of 22 long non-coding RNA's, 49 non-coding RNAs and 15 pseudogenes which could be investigated further.

3.5.4 Adding quality and k-mer abundance information enhances the normalization

As mentioned above, ORNA iterates over datasets in the order created by the sequencer. An important read which is towards the end of the dataset might get skipped if all the labels present in the reads are covered from the reads before. But how can we know if a read is important without assembling the whole dataset first? For this, we came up with two scores of a read r_i namely (a) read quality score which is set by taking the sum of Phred scores of all the bases (3.4) and (b) read abundance score which is set by taking the median of abundances of all labels present in r_i (3.6).

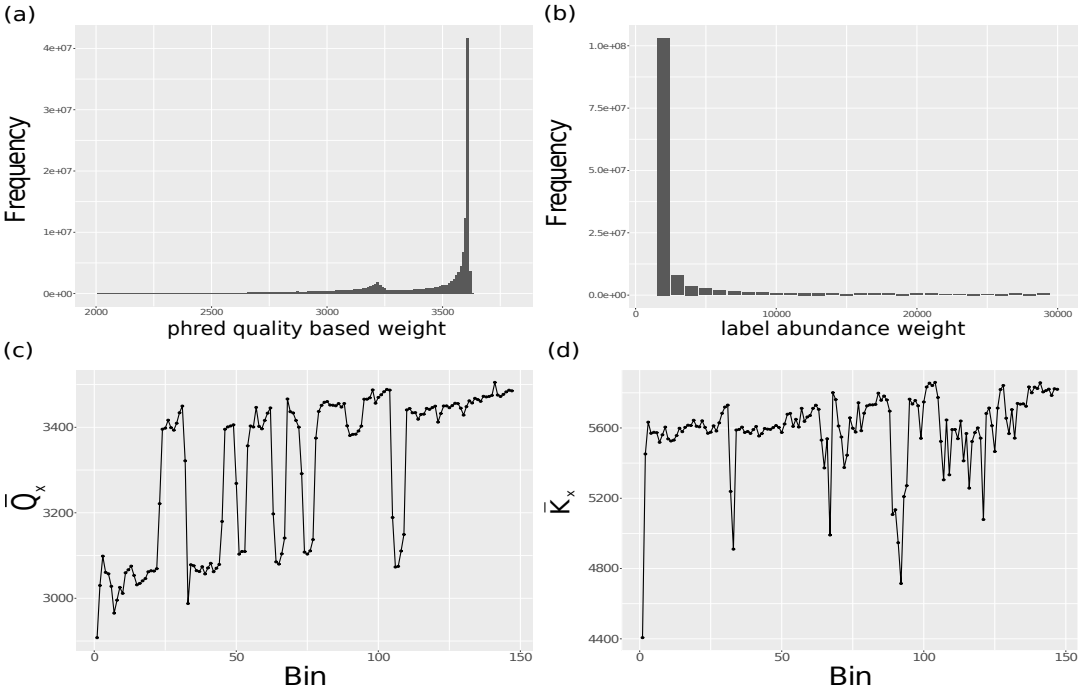

FIGURE 3.9: Average read quality score (a) and average read abundance score (b) distribution in Brain RNA-seq data. The position-wise distribution of average read quality score (c) and average read abundance score (d) in the brain dataset. Reads in both datasets were divided into bins of 1 million (x-axis). These bins were then considered as partial datasets and the scores ($\bar{Q}_x$ and $\bar{K}_x$) was calculated for each bin (y-axis).

We assign a weight to r_i by taking the inverse of the scores (eq. 3.5 and 3.7). Figure 3.9a,b shows the distribution of both types of scores in the uncorrected Brain dataset. We see that there is a large percentage of low quality and low abundant reads in the dataset which might never be used in the process of assembly. Next, we wanted to see at which position do these reads occur in the original dataset. For this, we divided the read set into bins of 1 million according to their position in the data file (in fastq format). For each bin, we calculated the average quality score ($\overline{Q}_x$) and the average read abundance score ($\overline{Q}_x$) where x denotes a bin of size 1 million. Figure 3.9c,d show the position-wise distribution of average read quality scores and average read abundance scores in the Brain RNA-seq dataset. In both cases, we see that there is a decent percentage of high scoring reads towards the end of the dataset which might get skipped during the process of normalization using ORNA. To ensure that high scoring reads are given priority over others in the process of normalization, we propose two extensions of ORNA namely ORNA-Q and ORNA-K. They assign weights to each read which is inversely proportional to the read-quality score or read-abundance score. ORNA-Q/-K retains a minimum number of reads required to cover all k-mers from the original dataset a certain number of times. At the same time, it minimizes the overall weight of the normalized dataset. We achieve this by reordering the input set of reads according to read weights and proceed with the read normalization on the ordered set as input.

Read ordering in ORNA-Q/-K is better than random ordering

To check the effect of ordering on normalization, we randomized the ordering of reads in the Brain dataset to come up with four different orderings (order 1-4). We

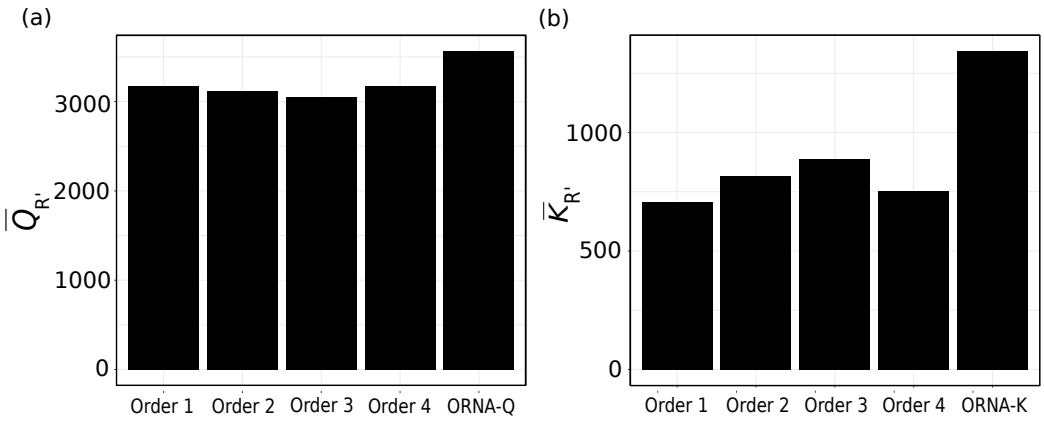

FIGURE 3.10: Comparison of (a) ORNA-Q and (b)ORNA-K applied on original Brain dataset against ORNA applied on four random orders (Order 1-4) of Brain dataset. The y-axis denotes the average read score obtained from normalization of different ordering (x-axis).

applied ORNA, in its default settings, to all the orders. Simultaneously, we applied ORNA-Q and ORNA-K on the original ordering of the Brain dataset. Figure 3.10 compares the application of ORNA on the random orderings against the performance of ORNA-Q/-K in terms of average read score of the normalized dataset. We observed that both ORNA-Q and ORNA-K improve the average read score in the reduced datasets as compared to all other orderings. This suggests the fact that ORNA-Q/-K reorders the original dataset correctly and selects more high-scoring reads as compared to ORNA on the random ordering.

ORNA-Q/-K improves assembly performance

Next, we wanted to check the effect of ORNA-Q/-K on the assembly quality. To obtain various levels of reduction, we normalized the Brain and HeLa dataset using ORNA-Q and ORNA-K with different parameter settings. Simultaneously, we normalized the two datasets using ORNA, Diginorm, and base-quality aware normalization software Bignorm. Again, we used different parameters to achieve different levels of reduction. We assembled the normalized datasets using TransABySS. We used REF-EVAL to evaluate the resultant assemblies and obtained nucleotide precision, recall, and F1 score. Table 3.4 lists the average precision, recall and F1 score across all normalized datasets for the Brain and HeLa dataset. We observe that the F1 score, which is the harmonic mean of nucleotide precision and recall, behave more or less similar and stable across all the algorithms. We see ORNA-Q performing slightly better in the case of Brain and ORNA-K performing slightly better in the case of HeLa. It is also interesting to see that for the datasets, ORNA-Q and ORNA-K perform better than ORNA. This might be because both ORNA-Q/-K retain more high scoring reads which are important for the assembler. We also noticed this in fig 3.10.

To measure assembly contiguity, we obtained the number of 100%-hits from each assembly. We compared the performance of ORNA-Q/-K against the performance of ORNA, Diginorm, and Bignorm. Figure 3.11 compares the number of reads reduced (x-axis) against % of complete (y-axis) for the Brain and HeLa datasets, respectively. We observe that ORNA, ORNA-Q, and ORNA-K perform better than

TABLE 3.4: Comparison of mean F1 scores, nucleotide precision, and nucleotide recall. Brain and HeLa datasets normalized by the five algorithms (ORNA, ORNA-Q/-K, Diginorm, and Bignorm) were assembled using TransABySS. Several normalized datasets were obtained by varying parameters for each algorithm. Each of these datasets was assembled separately. All the assemblies were then evaluated using REF-EVAL. Averages were taken over results obtained from different assemblies. The mean F1, precision and recall scores obtained for the original (unreduced) dataset is shown in the first column. The highest mean obtained by any normalization algorithm is underlined.

Dataset	measure	Unreduced	ORNA-K	ORNA-Q	ORNA	Diginorm	Bignorm
Brain	F_1score	0.442	0.441	<u>0.442</u>	0.438	0.441	0.426
HeLa	F_1score	0.280	<u>0.280</u>	0.279	0.278	0.273	0.272
Brain	Recall	0.347	0.347	<u>0.350</u>	0.345	0.349	0.331
HeLa	Recall	0.354	0.355	0.360	0.359	<u>0.372</u>	0.369
Brain	Precision	0.610	0.608	0.603	0.603	0.598	<u>0.635</u>
HeLa	Precision	0.232	<u>0.232</u>	0.227	0.227	0.214	0.219

Diginorm and Bignorm in general. This is mainly due to the retention of all k-mers from the original dataset which is not ensured in Diginorm and Bignorm. For Brain dataset, at a higher percentage of reduction (>75%) both, ORNA-Q and ORNA-K, outperform ORNA. A reason for this might be the fact that at a higher percentage of reduction the three algorithms require fewer reads to cover all labels from the original dataset. Since ORNA processes read in an order in which it is present in the input file, it selects the low-scoring reads which are in abundant towards the start of the file (fig 3.9). However, both ORNA-Q and ORNA-K reorder the reads according to the read score. Hence, they only select high scoring reads to fulfill the optimization conditions. At the lower percentage of reduction (55% - 75%) all the three algorithms retain similar reads and hence show similar performance. For HeLa, the performance of ORNA-Q and ORNA-K are only slightly better than ORNA. This might be again attributed to the ordering of reads in the original dataset. We found that unlike Brain, there is a significant percentage of high scoring reads at the start of the HeLa dataset. Hence, the event at a higher rate of reduction, ORNA's performance is comparable with the performance of ORNA-Q and ORNA-K. This supports the idea that the position of high-quality reads in the dataset influences in-silico normalization algorithms.

3.5.5 Comparison of resource requirements

As mentioned in section 3.2.3, ORNA retrieves, and stores all the k-mers from the original datasets in bloom filters which makes it runtime and memory efficient. In table 3.5, we show the comparison of memory and runtime required by ORNA against those required by Diginorm and TIS for Brain, hESC and the ENCODE dataset. We chose the normalized datasets which had similar read-counts. All the algorithms were run on a machine with 16GB register memory (RDIMM) and 1.5TB RAM. In general, ORNA requires less than half the memory and runtime as compared to TIS for both single and multi-threaded version. In Diginorm, a user can control the memory consumed by tuning the number of hashes and the size of each hash. A smaller number of hashes would reduce the runtime of Diginorm. However, it would also increase the probability of false-positives k-mers. A higher number of hashes would increase the runtime of Diginorm. For Diginorm, we tested two different parametrizations, denoted as Diginorm^a and Diginorm^b. The first uses less and

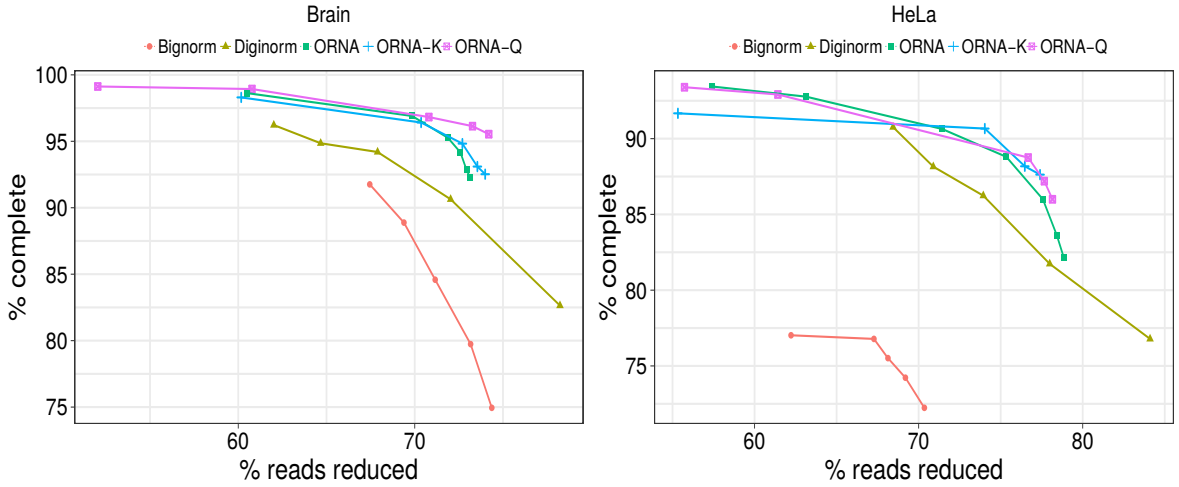

FIGURE 3.11: Comparison of assemblies generated from ORNA-Q, ORNA-K, ORNA, Diginorm and Bignorm reduced datasets. Each point on a line corresponds to a different parameters of the algorithms. The amount of data reduction (x-axis) is compared against the assembly performance measured as *% of complete* (y-axis, see text). (a) and (b) represent TransABySS assemblies (k=21) applied on normalized Brain and HeLa data.

the second a similar amount of memory compared to ORNA. In both cases, we see that ORNA has an advantage of runtime over Diginorm. While Diginorm's memory can be flexibly set, we note that ORNA uses much less space than the assembly of the reduced dataset itself, therefore not restraining the workflow.

TABLE 3.5: Runtime (in minutes) and memory (in GB) required by different normalization algorithms. Time and memory as obtained by running with 10 threads are shown in brackets (if possible). Note that the memory of Diginorm can be set by the user. For comparison it is set such that it uses less or similar memory than ORNA denoted as Diginorm ^{a} and ^{b}, respectively. The percent of reads reduced by each method (% reduced) is shown in the first column for each dataset. The total number of reads (in millions) and the file size (in GB) of the original dataset is shown in brackets next to the dataset.

method	Brain (147M - 20.1GB)			hESC (142M - 13.2GB)			combined (883M - 98.1GB)		
	%reduced	time [min]	mem [GB]	%reduced	time[min]	mem [GB]	%reduced	time[min]	mem [GB]
ORNA	83.3	104 (41)	6.62 (5.57)	63.90	58 (21)	6.16 (6.11)	81.4	740 (314)	32.9 (33.01)
Diginorm^a	81.86	110	3.13	61.63	115	3.13	80.37	760	12.51
Diginorm^b	81.51	135	6.26	62.59	126	6.26	79.31	2158	34.3
TIS	83.92	160 (95)	20.39 (20.19)	62.16	145 (127)	13.54 (13.53)	82.04	1859 (783)	95.19 (96.09)

In case of ORNA-Q and ORNA-K, we encounter an additional runtime and memory requirements for the sort step of the algorithms. Table 3.6 compares the resource requirements of ORNA-Q/-K against the requirements of ORNA, Diginorm and Bignorm for uncorrected Brain and HeLa dataset. Like Diginorm, the memory consumption of Bignorm can be controlled by tuning the algorithm parameters. Hence, we have used two versions of Bignorm (Bignorm^a and Bignorm^b) where Bignorm^a uses the default setting except the k-mer size. and Bignorm^b uses a smaller hash size which limits the memory consumed. We also used two versions of Diginorm namely Diginorm^a and Diginorm^b where Diginorm^a uses the default settings and Diginorm^b sets the number of hashes to 2 to reduce the memory requirements. The parameters for Bignorm^b and Diginorm^b were set in a way so that the memory consumption of these variants were similar to that of ORNA-Q and

ORNA-K. Note that the amount of memory used in Diginorm and Bignorm is directly proportional to the precision of *k*-mer counts. In other words, to obtain a high precision in their normalized datasets, both Diginorm and Bignorm require a large amount of memory.

Since ORNA-Q and ORNA-K calculates and stores read weights, it requires more memory and runtime as compared to ORNA which is validated from the table. However, we see that the increase is not significant and is outweighed by the improvement in the performance. One of the reason for this might be the fact that ORNA-Q/-K store the read weights in disk as files. Also, the count-sort step used to re-order the reads is quite fast and memory efficient. When compared to Diginorm^b and Bignorm^b, ORNA-Q and ORNA-K require more runtime when the memory consumption is similar. But in these settings Diginorm and Bignorm lose out on precision of the *k*-mers. And when default settings are used for Diginorm and Bignorm (Diginorm^a and Bignorm^a), they require more than double the memory required by ORNA-Q/-K

TABLE 3.6: Runtime (in minutes) and memory (in GB) required by ORNA-Q/-K, ORNA, Diginorm and Bignorm for normalizing Brain (147M) and HeLa dataset (216M). *Notes:* The memory required to store the complete dataset in the main memory is indicated in brackets next to the name of the dataset. The column *% reduced* states the percent of reads reduced by each method. Time and memory as obtained by running the algorithm with 10 threads (if possible) are shown in brackets. * Bignorm always runs with 4 cores and fixed memory settings.

method	Brain (147M - 35.1GB)			HeLa (216M - 60.7GB)		
	% reduced	time [min]	mem [GB]	% reduced	time [min]	mem [GB]
ORNA	69.8	112 (42)	6.38 (6.31)	75.31	219 (64)	9.81(9.85)
ORNA-Q	70.65	116 (50)	7.10 (7.13)	72.72	223 (70)	10.01 (10.02)
ORNA-K	70.3	130 (52)	6.41(6.5)	73.86	279 (75)	9.98 (10.01)
Diginorm^a	72.03	135	12.5	72.91	198	12.51
Diginorm^b	70.51	112	6.26	75.10	155	9.76
*Bignorm^a	69.38	(47)	41.94	71.28	(58)	41.94
Bignorm^b	69.39	(41)	5.23	71.26	(55)	5.24

3.6 Discussion

Transcriptome sequencing have been constantly used to study non-model species for which no reference sequence is available. De novo assembly strategy is used to assemble the high coverage data produced from the sequencer. However, it has already been shown that a high percentage of data is redundant and can be removed prior to assembly process ([Yor+16]). Current normalization algorithms do not guarantee the preservation of the underlying de Bruijn graph structure (nodes and connections). In this chapter, we proposed a set multi-cover (SMC) based optimization approach, named ORNA, for normalization of RNA-seq data. ORNA preserves the nodes and edges of the de Bruijn graph by retaining all the connections from the original dataset. It retains the minimum number of reads required to cover all connecting *k*-mers from the original dataset a certain number of times. We termed the connecting *k*-mers as labels. We also set individual demands/threshold of each label allowing us to retain the relative difference in abundances between labels. This is

important as many assemblers use this information to resolve complex graph structures.

Since ORNA retains all the *k*-mers from the original dataset, it is quite natural that any erroneous bases present in the dataset will be retained. Hence, it is always beneficial to error correct the dataset, using RNA-seq specific error correctors like SEECER ([Lee03]), before normalizing them. However, the error correction algorithms are themselves a heuristics and many erroneous bases escape their stringent filter. To reduce the probability of retaining such erroneous bases, we proposed two extensions of ORNA namely ORNA-Q and ORNA-K which were based on the principles of *Weighted Set Multi Cover* (WSMC) optimization. These extensions incorporate base quality (ORNA-Q) or *k*-mer abundance information (ORNA-K) into ORNA's SMC approach. ORNA-Q and ORNA-K score each read based on either the Phred score of bases present in the read or the abundance of *k*-mers in the read. The scoring is done in such a way that the reads have erroneous bases end up with low scores. Both ORNA-Q and ORNA-K retain a minimum number of reads required to cover all labels from the original dataset. But at the same time, they ensure that high-scoring reads are given priority over others during the process of selection. However, any erroneous base which is covered by only one *k*-mer will always be retained.

We compared the performance of ORNA against two state-of-the-art normalization algorithm namely Diginorm and Trinity's In-Silico normalization. Both these algorithms are *k*-mer based like ORNA. However, neither of the two algorithm guarantee the preservation of *k*-mers which form connections in the DBG. We applied the three algorithms and normalized two datasets generated from Brain and hESC. We assembled the normalized datasets using two DBG based assemblers and evaluted the assemblies produced. In terms of F1 score, we noticed that all the algorithms produced stable results with ORNA performing slightly better. Concerning the amount of full-length assemblies (100%-hits), ORNA reduced datasets performed better than Diginorm and TIS reduced datasets on the brain data. For hESC data, the results were not as clear as in Brain and depended highly on the assembler used. The difference in the distribution of expressed mRNAs and their expression in Brain and hESC might be another reason behind this variation. Also, we found that order of the reads within a dataset has an impact on the normalization. This was confirmed by the observation that ORNA-Q and ORNA-K retained better quality and high scoring reads as compared to ORNA. Also for the Brain dataset, where a high percentage of high-scoring reads were towards the end of the dataset, ORNA-Q and ORNA-K were able to retain more number of 100%-hits. We observed that, for both the datasets, Bignorm was performing the worst in terms of number of 100%-hits assembled. Apart from the quality threshold, Bignorm bases its decision on three other parameters namely rarity threshold, contribution threshold and abundance threshold. There effect on the performance of Bignorm were not explored in the manuscript of Bignorm. To study the effect of these parameters, one has to try multiple combinations which would result in a large and complex search space. Hence, we restricted our analysis on the most important parameter which is the quality threshold (-Q). However, proper optimization of all the four parameters may lead to a better performance of Bignorm on the RNA-seq data.

Now that we saw ORNA and its extensions are able to scale down the process of de novo assembly, they can also be used to fulfill other computationally expensive

applications. We proposed one such application we generated a dataset by concatenating multiple ENCODE datasets and normalized it. Upon assembling the normalized dataset, we obtained more than 300 protein coding transcripts along with several novel non-coding RNAs. The sequence information required to assemble these protein coding transcripts and non-coding RNAs could be only obtained when the datasets are combined. And since ORNA retains all *k*-mers, these sequences are also retained. The principles of ORNA and its extensions can also be applied on metagenomics and metatranscriptomics datasets as these kind of datasets also exhibit a non-uniform coverage. Although ORNA is designed keeping DBG based assemblers in mind, it can also prove to be effective for other approaches like overlap-layout-consensus assembly which currently requires a large amount of computational resources ([Li+12]). We noticed from the results that there is a variability in the performance of ORNA and its extension when different parameters are used. This underlines the fact that ORNA is still a heuristic that can be tuned further to achieve better results. Overall, ORNA's reduction of data and its impact of the assembly performance is promising and thereis scope of integrating it into a sequence assembler. We feel it will encourage researchers in developing complex heuristics to the assembly problem without worrying about the resource requirements. This would then be step forward in achieving the goal of generating an accurate and complete transcriptome of a species from short read data.

3.7 Availability

ORNA and ORNA-Q/K are open-source software developed in C++11 language and compiled using GCC version 4.7. ORNA can be downloaded via the code hosting platform GitHub (github.com/SchulzLab/ORNA) and should be executed in a linux based system. ORNA-Q/K are extensions of ORNA and can be executed by setting the corresponding parameter in ORNA. The input to the program is a fasta or fastq read file along with a *k*-mer size, and a value of the base of the logarithm function (for threshold calculation). Using the *k*-mer size and the base value, ORNA calculates the threshold *t* for each label present in the dataset and retains minimum number of reads required to cover each label at least *t* times.

Chapter 4

SOS: Streamlined de novo transcriptome assembly

The previous chapters describe algorithms for normalizing the coverage of RNA-seq data in an informed way. This chapter introduces a pipeline named SOS which streamlines *de*-novo transcriptome assembly by adding a tuple of read-error correction and normalization over the existing layer of sequence assembly. In the results, we show that using SOS enhances the quality of the assemblies produced by generating more full-length transcripts as compared to running the assemblers in their default mode.

4.1 Motivation

Existing pipelines such as DRAP ([Cab+17]) and the approach used for the re-assembly of Marine Microbial Eukaryote Transcriptome Sequencing Project (MMETSP) data ([Kee+14; JAB18]) offer insight towards using a streamlined workflow for de novo transcriptome assembly. These pipelines pre-process their data by trimming adapters and removing low-quality regions from the input reads followed by normalization using Diginorm. Studies have shown that these pre-processing steps run into a major risk of losing important k-mer information and hence might cause sub-optimal assemblies ([DS18; Le+13]). Moreover, DRAP uses only two assemblers namely Oases and Trinity whereas the MMETSP pipeline uses only Trinity. Since assembly algorithms are mostly heuristics, we believe that using one or two assemblers does not guarantee a high-quality assembly for multiple datasets. Also, due to the recent advances in second generation sequencing technologies high coverage data from non-model species are being constantly sequenced. This has resulted in development of modern and sophisticated assembly procedure. We believe that this trend would continue for a while. So in future, pipelines such as DRAP might become obsolete unless since they use only a limited set of tools for analysis. So there is a need for streamlined pipeline which could incorporate multiple pre-processing, assembly and post-processing tools which can be easily updates to future changes.

In this chapter, we introduce SOS which combines tools for error correction, normalization, assembly, and transcript quantification. The advantage of SOS lies twofold 1) SOS improves the quality of the assembly produced as it incorporates necessary pre-processing steps like error correction and normalization and 2) it is quite flexible in its assembly implementation. In other words, assembler used in the pipeline can be easily replaced with another assembler by changing just a few lines of codes in the source file.

4.2 Method

4.2.1 SOS at a glance

SOS pipeline consists of three major steps: 1) pre-processing, 2) *de*-novo transcriptome assembly and 3) expression estimation. Figure 4.1 depicts the major steps in the pipeline. We begin by removing sequencing errors from the dataset. This is done using RNA-specific error correction algorithm named SEECER. We then proceed by normalizing the data using ORNA. This reduces the coverage of the dataset without losing any sequence information from the original dataset. However, this step is optional and can be skipped if the coverage of the original data is already low. The error correction and normalization steps constitute the pre-processing stage of the pipeline. The resultant data from the pre-processing step is assembled using a multi *k*-mer based assembler incorporated with the parameter estimator algorithm KREATION. The final step of the pipeline is to quantify the transcripts produced from the assemblers using Salmon. In further sections, we explain all the above procedure in detail. All the tools and evaluation metrics used in SOS have been explained in detail in chapter 2.

4.2.2 Pre-processing

Error correction is considered as one of the most important steps in any pre-processing workflow. However, the non-uniform coverage of RNA-seq data posses a major challenge towards this. Since DNA-specific error correctors rely on the coverage of the input dataset, they cannot be applied directly to RNA-seq data. RNA-seq specific error correction methods such as *HAMMER* ([Med+11]), *BayesHammer* ([NKA13]), and RCorrector ([SF15]) error correct the data using a combination of *k*-mer based hamming graph and probabilistic error models. They identify groups of "solid *k*-mers" based on the abundance information. The erroneous or "weak" *k*-mers in the dataset are corrected based on the solid *k*-mers using probabilistic models. In our work, we use SEECER which has been shown to perform accurately on substitution as well as indel errors ([BM+12]). For analysis, we use default parameters except for the *k*-mer size which was set to 17. We direct the readers to chapter 2 for more details on SEECER.

Error corrected reads are then normalized using ORNA. As described in the previous chapter, ORNA retains the minimum number of reads required to retain all *k*-mers from the original dataset a certain number of times. This ensures that the structure of the de Bruijn graph is preserved and the quality of the assemblies produced is not compromised. Again, we direct the readers to chapter 3 for details of ORNA.

4.2.3 Transcriptome assembly

Reference free transcriptome assembly marks the next step of SOS. Currently, SOS is tested on three de Bruijn graph-based assemblers namely - Oases ([Sch+12b]), Trans-ABySS ([Rob+10]) and SOAPdenovo-trans ([Xie+14]) and one splice graph-based assembler namely TransLiG ([Liu+19]). Assemblies generated using multiple *k*-mer sizes outperform assemblies using a single *k*-mer size. Hence, we generate assemblies using multiple *k*-mer values and merge them to form a single non-redundant assembly. However, selecting a range of *k*-mer sizes for obtaining a good quality assembly is a challenge. Random selection of *k*-mer range comes with a risk of

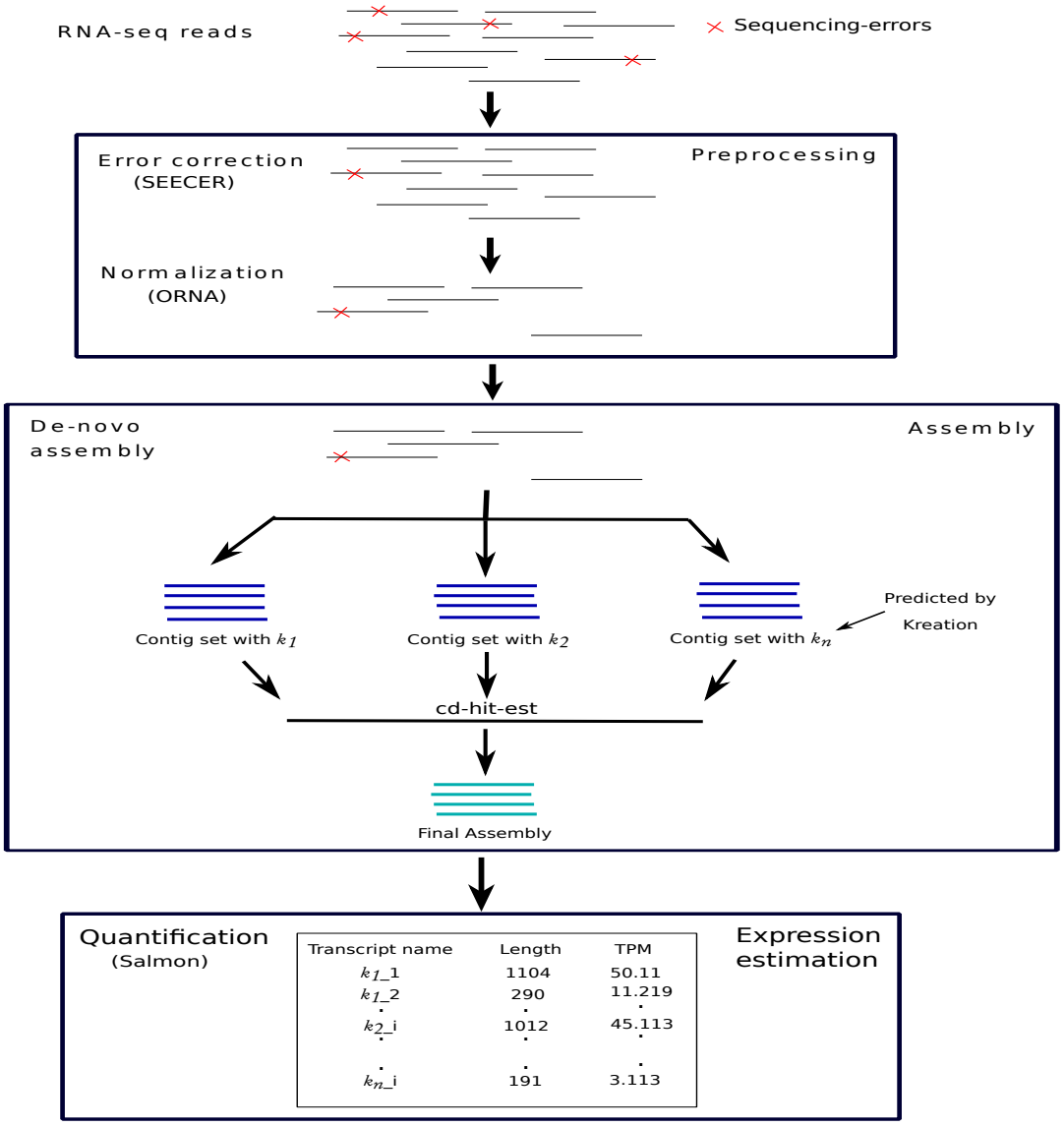

FIGURE 4.1: Workflow of SOS. Reads are error corrected using HMM model based algorithm SEECER. The error corrected are normalized using ORNA. The normalized reads are assembled using a multi *k*-mer based assemblers. The assembler is integrated with KREATION which is a *k*-mer parameter selection. The final assembly is quantified using salmon to achieve a transcript level expression estimation.

losing out on important *k*-mers and hence lowering the overall sensitivity of the final assembly. Default *k*-mer ranges provided with the assembly algorithms are not guaranteed to produce optimal assemblies as they are data-dependant. Running the assembler over the entire range of the read length generally results in a lengthy assembly process. Parameter prediction algorithms such as "velvetadvisor" and *k*merGenie ([CM14]) are not effective as they are genomic read specific. Also, none of the above methods estimates a *k*-mer size range which could be used for a multi *k*-mer based assembly procedure. The RNA-specific method KREATION algorithm only estimates the upper bound of *k*-mer range. This came closet to our requirements. In SOS, we set the minimum *k*-mer size of the range to one-third of the read length and let KREATION estimate the stopping *k*-mer for the assembler.

4.2.4 Transcript quantification

The assembly procedure is followed by a transcript quantification step. Current tools for transcripts expression quantification can be classified into two sub-types: 1) alignment-based and 2) alignment-free. In the alignment-based mode, reads are aligned back to the assembled transcripts using aligners such as Bowtie2 ([Lan+09]). Given the alignment, tools such as eXpress ([RP13]) and cuffllinks ([Tra+10]) use expectation-maximization (EM) procedures to estimate transcript abundance. TIGAR2 ([Nar+14]) uses Bayesian inference to quantify transcript expression levels. However, alignment-based methods are slow and memory intensive. Also, reads, if not long enough, can map to multiple transcripts thus making the quantification procedure inaccurate. Recently developed algorithms such as Sailfish ([PMK14]), Kallisto ([Bra+16]) and Salmon ([Pat+17b]) do not rely on accurate base-to-base mapping and only wish to know the region from which the read might have originated. This speeds up the quantification process. The sophisticated error models used in these modern quantification algorithms improves the accuracy of the expression estimation. For SOS, we use salmon for quantifying transcripts in alignment-free mode. We use default parameters for building the index from the assembled sequences and quantifying the expression estimates.

4.3 Data retrieval and assembly evaluation

The pipeline was applied on three Illumina paired-end datasets - 147M paired-end reads from Brain (SRR332171, [BM+12]), 344M reads from A549 cells (SRX085304) and 250M reads from Blood (SRX984188, [Zha+15]). All the datasets were downloaded from the SRA run browser.

For evaluating the assembled transcripts, we aligned transcripts against the reference genome using Blat ([Ken02]) and compared it against annotated ENSEMBL transcripts ([Zer+18], version 78). We then used standard metrics of 100%-hits, nucleotide sensitivity, and nucleotide specificity. Further we compared our assemblies against benchmarking sets of universal single-copy orthologs (BUSCOs) compiled by [Wat+13]. The sets are identified using OrthoDB database from Metazoan, Vertebrate, Arthropod and Fungal lineages. BUSCO sets consists of orthologous genes which exists as single-copy in at least 90% species of the lineage. The tool BUSCO ([Sim+15], v4.0, [HM19]) detects these orthologous candidate genes and obtains the abundance of single-copy orthologs. To avoid confusion between the tool BUSCO and set BUSCO, we will from hereon use the term BUSO-T for refererring

to the tool. The obtained orthologs or BUSCOs are generally divided into four categories namely i) complete and single-copy, ii) complete and duplicated, iii) fragmented, and iv) missing. If more than one copy of a *Complete* gene is recovered, then it is classified as *Complete and Duplicated*. Similarly, if the gene is recovered only in fragments, then it comes under the category of *Fragmented*. We term the genes which are not covered by any of the transcripts as *Missing*. For our analysis, we only compared our assemblies from 255 single-copy orthologs from eukaryotes.

4.4 Results

4.4.1 Error correction improves assembly quality

Most of the current assemblers have their error correction step integrated into them. The basic idea behind their error correction procedure is the difference in abundance between true *k*-mers and the erroneous *k*-mers generated from allelic differences, repeat sequences with minor variations and base-calling read errors. As mentioned in the previous chapters, these erroneous *k*-mers have a low abundances as compared to their neighbouring *k*-mers and form bubbles and tips in the graph. An assemblers generally pops the bubbles and tips by removing the lower coverage variants from the graph. However, in case of RNA-seq low *k*-mer abundance does not always imply the *k*-mer being erroneous. As a result, the assemblers in-built error correction step is not always effective in case of RNA-seq data.

In SOS, we added a tuple of error correction using SEECER over the assembly step. We studied the impact of error correction on the quality of the assemblies. We assembled an uncorrected and error corrected Brain dataset separately. Note, that there is normalization is not executed for this analysis. For assembly, we used a single *k*-mer based assembler(TransABySS) and a multi *k*-mer based assembler(Oases-M) with default parameters. Table 4.1 compares the assemblies generated using both the strategies. We see that using error corrected data, there is an improvement in the nucleotide sensitivity and nucleotide specificity for both the assemblers. We are also able to assemble at least 15% more full-length transcripts (100%-hits). We took these additional 15% full length hits and checked its existence in the assembly done using uncorrected data. As can be seen from fig 4.2, these transcripts were present in fragments in their respective assemblies. Many of the transcripts were assembled 50%-80% of their total length. Due to the sequencing errors, they lacked important *k*-mer information to connect regions in the graph and complete their sequence. It is interesting to note that many of these transcripts get assembled up to 99% of their length. The reads, forming these transcript, had very few errors towards the end. Hence, the assembler were short of few correct bases needed to complete the sequence. Error-correcting the reads transforms the erroneous bases into true bases and thus completing the transcript.

Assembler	Strategy	#Transcripts	100%-hits	Sensitivity	Specificity
TransABySS	Assembly	142037	4435	28.15	76.8
	EC-Assembly	149290	5129	29.25	79.11
Oases	Assembly	704807	5790	39.97	76.2
	EC-Assembly	529766	6795	44.15	80.81

TABLE 4.1: Impact of error correction on the quality of the assembly produced from Brain dataset.

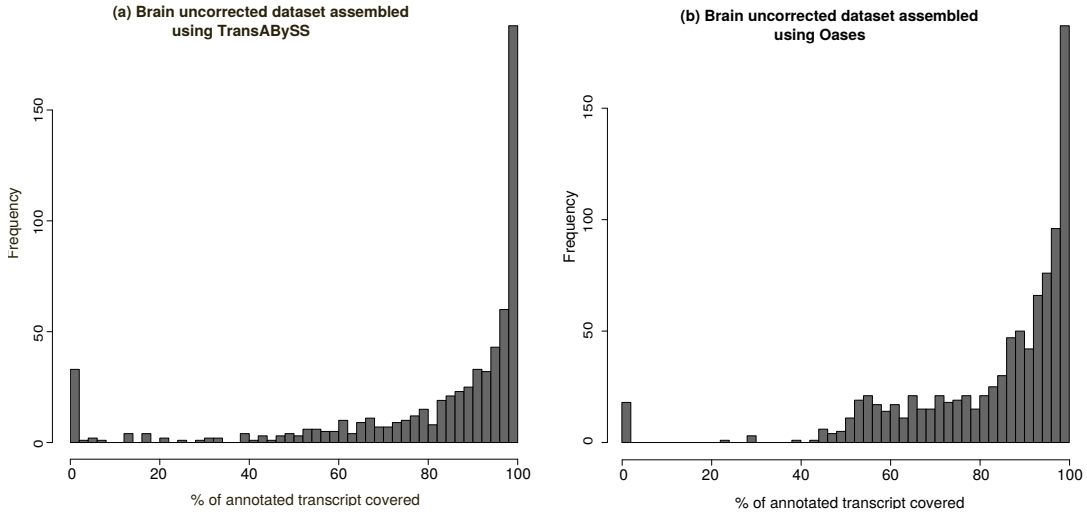

FIGURE 4.2: Distribution of annotated transcripts which were assembled completely when error corrected data was used but were assembled only in fragments when uncorrected data is used for the assembly.

4.4.2 Combined error correction and normalization generates fast and accurate de novo assemblies

The process of assembly generally requires a lot of computational memory and runtime especially for bigger datasets. Hence, to reduce the memory and runtime requirements, we use ORNA which has been shown to normalize the data upto 70% without a significant impact on the assembly quality ([DS18]). As ORNA retains all the *k*-mers from the original dataset, the normalized dataset also contains erroneous *k*-mers. The error-correction step before normalization procedure limits the number of errors retained. We tested the impact of error correction and normalization on the assemblies produced from Brain datasets. First, we normalized the error corrected and uncorrected Brain dataset separately to different levels. This is done by varying the base(b) parameter of ORNA. We assembled these datasets using Oases with a *k*-mer size range of 21-49 and a step size of 2 (read length of Brain dataset was 50bps). As seen in fig 4.3a, running ORNA on the corrected data reduces more reads compared to the uncorrected data. This is expected as erroneous *k*-mers are converted into true *k*-mers reducing the total number of unique *k*-mers in the original datasets. As a result, the size of universe *k*-mer set is reduced and ORNA requires smaller number of reads to cover the universe set. We also observe error correction leads to more number of 100%-hits as due to error correction the assemblers traverse the correct path in the DBG.

But what about the computational resource requirement for the additional step of error correction and normalization? So far, we saw that assembly can be performed on –(i) unreduced and uncorrected dataset, (ii) unreduced but error corrected dataset, (iii) error corrected and normalized dataset. Figure 4.3b shows the maximum memory required for assembling the brain dataset using the above three strategies. Erroneous *k*-mers complicate the de Bruijn graph by adding extra nodes and extra paths. This is reflected in the higher memory requirement for the uncorrected data (red dot in fig. 4.3a). By error-correcting the data, the memory is reduced

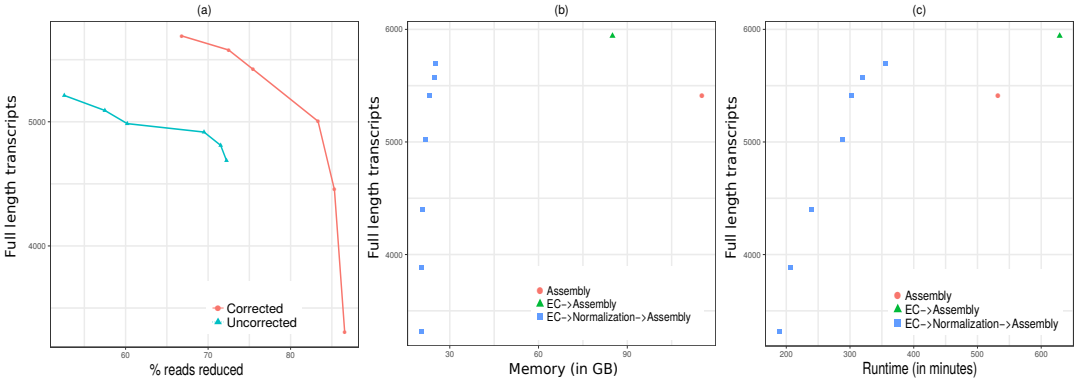

FIGURE 4.3: Impact of error correction on assembly performance. Corrected and uncorrected brain dataset were normalized to different levels by varying the base parameter of ORNA (a) Number of full-length hits (y-axis) obtained at different levels of reduction (y-axis). Each point in the graph represents the base value used for reduction.(b and c) Comparison of the number of full-length transcripts (y-axis) assembled using error corrected (EC, depicted as green triangle), EC and normalized (depicted as blue square), and uncorrected data (depicted as red cirlce) against the memory (b) and runtime(c) required to assemble using the mentioned strategies (x-axis).

by nearly 20% (green triangle in fig. 4.3a). It is further reduced when the error corrected data is first normalized and then assembled. The users might be worried with the increase in runtime due to the additional pre-processing steps. But this increase outweighed by the improvement in the assembly quality. Moreover, the runtime can be reduced by the normalization step. As shown in fig. 4.3c that for three ORNA parameters (b=1.3, 1.5, 1.7) the assembly quality is better than the assembly of uncorrected data with much lower time and memory consumption. For more stringent parameters (b>1.7) the quality degrades due to a high % of reduction.

4.4.3 KREATION avoids unnecessary runs of assembly

It has been previously shown that uninformed parameter selection, especially k-mer size range, for the process of assembly results in a sub-optimal assembly ([DS16]). In this work we used KREATION which can be integrated into any multi k-mer based assembler. Starting from an assembly using a user-defined k-mer size, KREATION evaluates the novel sequences added in each k-mer iteration and stop the assembly process when no new sequence information is added to the assembly pool. In the manuscript of KREATION, it was shown that the algorithm is able to accurately predict the stopping k-mer for multiple combinations of datasets and assemblers. KREATION also resulted in the reduction of misassemblies produced and the runtime required by the assembler.

However, the performance of KREATION on the assembly of normalized dataset has never been studied. To test this, we ran Oases and TransABySS on Brain dataset using the k-mer range which spanned over the entire read-length. We used a series of increasing kmer values $K = (k_1, \ldots, k_n)$, where $0 < k_1 < k_2, \ldots, < k_n \leq$ *read-length* and $k_i = k_{(i-1)} + 2$. We set $k_1 = 21$ and ran the assembly till $k_n = 49$. We term the number of 100%-hits obtained as *optimal*. With this notion, we measure the performance of any $k_i \in K$ by calculating the number of 100%-hits obtained by merging assemblies from k_1 to k_i and denoting the number as *% of optimal*.

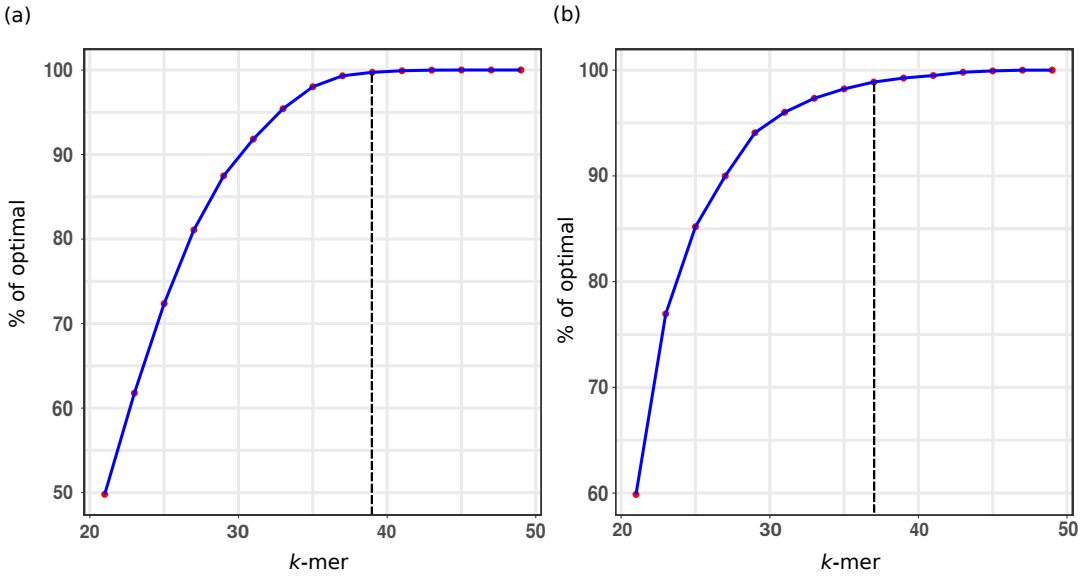

FIGURE 4.4: Performance of KREATION on error-corrected and reduced Brain dataset using Oases(a) and TransABySS(b) assembler. The stopping k-mer estimated by KREATION is depicted by dashed line on the x-axis and the percentage of optimal reached till the stopping k-mer is depicted on the y-axis. The number of 100-% hits achieved when the assembler is run using a k-mer range covering the entire read-length is considered as *optimal*

As seen in fig 4.4, *% of optimal* increases with the increase in k-mer size. KREATION estimated a stopping k-mer of $k = 39$ for Oases (fig 4.4a) and $k = 37$ for TransABySS (fig 4.4b) denoted by a dashed line in the figure. As expected, the assembly performance starts getting saturated after these k-mers. In other words, very few 100%-hits are assembled in the iterations after the KREATION estimated stopping point. We list the complete numbers in tab 4.2. As the table shows, nucleotide sensitivity and specificity are not affected when KREATION is used in the assembler. The number of k-mer assemblies not computed and the time saved due to KREATION is also shown in the table.

Assembler	Strategy	100%-hits	Sensitivity	Specificity	Runs saved	%runtime reduced
Oases	Full	5851	32.01	70.31	-	-
	KREATION	5835	32.12	70.59	5	10
TransABySS	Full	7459	32.41	66.32	-	-
	KREATION	7417	32.57	66.62	7	12

TABLE 4.2: Impact of KREATION on the quality of the assembly produced from Brain dataset in terms of 100%-hits, nucleotide sensitivity and specificity. We also report the number of assembly runs saved and % runtime reduced due to KREATION.

4.4.4 SOS shows improvement over running assemblers in their default version

An important feature of SOS is its flexibility to accommodate multiple assemblers with different settings. Here we tested SOS with three de Bruijn graph based assemblers namely TransABySS, Oases, SOAPdenovo-trans, and one splice graph based

Dataset	Strategy	TransABySS			Oases		
		100%-hits	Sensitivity	Specificity	100%-hits	Sensitivity	Specificity
A549	Default	9350	34.8	15.2	-	-	-
	SOS	12620	37.3	13.49	11702	39.06	15.54
Brain	Default	4435	28.1	76.8	5790	39.97	76.21
	SOS	7417	31.9	69.56	5842	32.21	70.64
Blood	Default	16367	38.04	33.76	-	-	-
	SOS	22399	41.86	35.62	22026	46.66	50.58
Dataset	**Strategy**	**SOAPdenovo trans**			**TransLig**		
		100%-hits	**Sensitivity**	**Specificity**	**100%-hits**	**Sensitivity**	**Specificity**
A549	Default	3287	30.8	11.8	12220	20.66	28.9
	SOS	5586	32.3	9.4	14296	34.7	41.7
Brain	Default	2669	28.5	66.0	4796	24.01	77.98
	SOS	4380	29.8	66.0	7807	30.5	85.2
Blood	Default	6232	32.88	30.7	22712	35.21	64.13
	SOS	12281	35.01	23.91	20579	33.72	62.95

TABLE 4.3: Comparison of assemblies generated using default settings (depicted as Default in the table) and assemblies generated using SOS (depicted as SOS in the table) in terms of 100%-hits, nucleotide sensitivity and specificity.

assembler TransLiG. All the assemblers were run with multiple *k*-mer sizes (where ever possible) and the assemblies generated from each *k*-mer size were merged to form a single non-redundant assembly. We started from a *k*-mer equivalent to one-third of the read-length. However, we couldn't follow this criteria for assembly of Blood dataset (read length of 101bps) using TransLiG as the assembler does not support *k*-mer size>31 and hence we used only a single *k*-mer size of 31. We generated assemblies till a stopping *k*-mer size predicted by KREATION. To compare our results, we also applied the four assemblers in their default settings. Table **??** shows the metric obtained using the above assembly strategies. As previously mentioned, SOS resulted in a consistent improvement of *100%-hits* and nucleotide sensitivity except for Blood assembled with TransLiG. SOS could only generate a single assembly from a reduced dataset due to the limitation of the assembler. The drop in assembly performance for TransLiG assembled Blood dataset is mainly due to the high percentage of reduction in the dataset.

Interestingly, we also see a drop in nucleotide specificity in the case of TransABySS and SOAPdenovo-Trans. Using multiple *k*-mer sizes results in the generation of more transcripts. These extra transcripts are either unannotated in the original ENSEMBL annotation or are misassemblies. Since nucleotide specificity depends on the number of assembled bases from the original annotation, multi-*k*-mer based assembly might show a drop in nucleotide specificity. However, for TransLiG there is an improvement in all the measures. This underlines the fact that assembly algorithms are heuristics and a single assembler cannot guarantee high-quality assembly all the time.

Further, to test the performance of assemblies in terms of gene content, we integrated SOS with Oases and TransABySS assemblers and tested in on Brain, A549 and Blood datasets. Simultaneously, we ran the assemblers individually with their default settings. For Blood and A549 datasets, Oases (in its default settings) required a huge amount of runtime and computational memory. Hence, the assemblies had

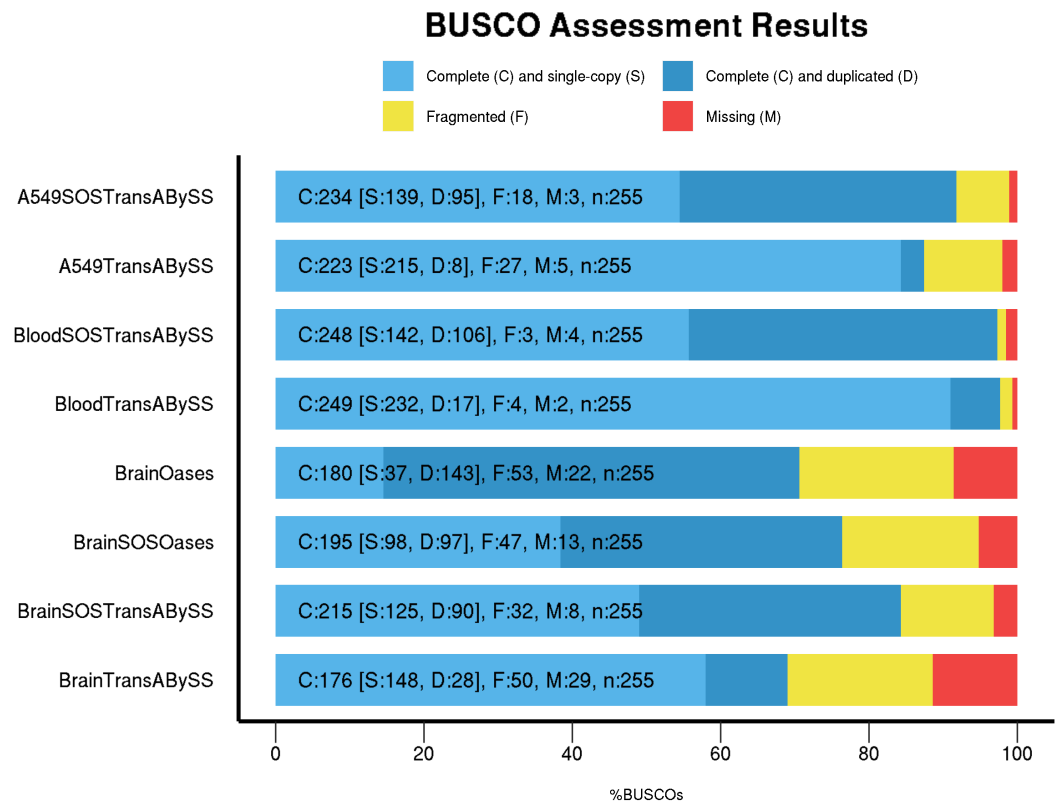

FIGURE 4.5: Identification of BUSCOs from the assemblies generated using SOS and from the assemblies generated using assemblers in their default version.

to be interrupted after nearly 27 days of execution. From the remaining assembled transcripts, we obtained the number of single-copy orthologs using BUSCO-T. These single-copy orthologs were termed as BUSCOs. As seen from fig 4.5, SOS consistently assembles a higher number of "Complete" BUSCOs except for the Blood dataset where the numbers are similar. It is interesting to note that there is a significant increase in the number of *Complete and Duplicated* BUSCOs. This might either be due to assembly of multiples copies of a single transcript or due to assembling of alternate isoforms of a gene which results in the gene getting covered multiple times.

4.5 Discussion

The progress in de novo transcriptome assembly has given an impetus to the study of non-model species. It has also shown the potential to detect novel transcripts that are not present in the reference genome assembly. However, only a few practitioners quality control and error correct their data before assembling them and hence miss out on important transcripts. There is a need for a streamlined approach towards transcriptome assembly which would guarantee a high-quality assembly all the time. With this thought, we introduce an automated pipeline named SOS which error corrects, normalizes and assembles short-reads from RNA-seq technologies. We integrated three de Bruijn graph-based assemblers (TransABySS, SOAPdenovo-trans, and Oases) and one splice-graph based assembler (TransLiG) to the pipeline.

We ran the assemblers using multiple *k*-mer sizes and merged the final assemblies to obtain a single non-redundant assembly. However, running assemblers in multi *k*-mer mode usually results in high memory and runtime consumption. We addressed this issue by adding two other tools namely i) ORNA which reduces the redundancy in the input dataset without affecting assembly sensitivity and ii) KREATION which estimates appropriate *k*-mer parameter for the assembler and hence avoids unnecessary assembly iterations. For evaluation, we used reference-based approaches such as detecting single-copy orthologs using BUSCO-T and obtaining of the number of full-length transcripts. We saw that the mere addition of error correction step and running the assemblers with multiple *k*-mers showed a drastic improvement in the number of full-length annotated transcripts being assembled. We also noticed that normalization using ORNA and parameter estimation using KREATION makes the assembly of large datasets feasible. But in general, we observed that no tool was dominant for all datasets.

We feel that SOS is an important step towards improving the transcriptome assembly procedure. However, this study was limited to only a few tools for pre-processing and assembly. There are many other tools which can be integrated into SOS and tested for its effect. We tried making SOS flexible such that these new tools can be incorporated into the pipeline. For instance, the performance of SEECER is found to drop when the sequencing depth is moderate to low ([ME13]). Hence, an alternative to SEECER when using low coverage data can be tested. Similarly we saw in the previous chapter that in some cases, Trinity's in-silico normalization works better than ORNA. The selection of the best assembler and pre-processing tools based on appropriate metric remains a challenge and needs to be investigated in the future.

It has been a constant endeavor of researchers around the globe to understand the complexity of the transcriptome. RNA-seq data has always been a reliable method to perform this task. However, there are certain gaps in the analysis that cannot be filled using RNA-seq data alone. The combination of RNA-seq data and other data types can resolve certain voids and hence being constantly applied. For instance, RNA-seq data can be combined with DNA sequencing data for RNA-editing analysis or Single Nucleotide Polymorphism (SNP) discovery ([Con+16; Gaf13]). Similarly, a combination of RNA-seq data with chromatin immunoprecipitation sequencing (ChIP-seq) data can throw light on the activating or repressive effect of transcription regulators on its target genes ([Wan+13; Cha+18]). RNA-seq data combined with proteomics can be effectively used for accurate isoform discovery. Moreover, practitioners can study post-translation regulatory effects by combining mass spectrometry data with RNA-seq data ([SSK08]). All these possibilities are been actively researched upon and tools incorporating these are constantly being developed. In future, SOS can be developed into a sophisticated pipeline which could incorporate all such tools. The final goal of SOS or any such pipeline should be to make a transcriptome which is close to the true sequence which would enable an accurate expression analysis.

4.6 Availability

SOS is an open source pipeline which is managed using workflow management tool SnakeMake. It can be downloaded via GitHub (github.com/SchulzLab/SOS). We

executed the pipeline in Linux and Mac environment. However, the requirements of SOS is not constant as it depends upon the requirements of the individual tools integrated in the pipeline.

Chapter 5

Transcript quantification and gene-fusion detection using long reads

In previous chapters, we used data from short-read or second-generation sequencing technologies. For a long period, sequencers such as Illumina's HiSeq, NextSeq, and MiSeq have been in favor due to its cost-effectiveness and accuracy. As a result, many tools and algorithms were developed for mining information from the reads generated from such sequencers. However, second-generation sequencing technologies suffer from some major drawbacks. For instance, mapping or assembling reads originating from complex regions of the genome, such as repeat regions and regions with high GC-content, is a challenge ([MKH19]). Also, short reads do not have enough coverage information for accurate estimation of alternate splice forms.

Long reads technologies such as sequencers from Pacific Biosciences and Oxford Nanopore have made significant progress in terms of sequencing output per run at a reduced cost. They cover a large percentage of transcript (in many cases the complete transcript) which makes them a candidate for accurate transcript quantification and gene-fusion detection. In this chapter, we propose an algorithm for long-read specific transcript quantification and gene-fusion detection. This work was done in collaboration with Prof. Dr. Tobias Marschall and Mikko Rautiainen from Saarland University and Prof. Dr. Jonathan Göke from Genome Institute of Singapore, Singapore. The work on fusion detection was done by Mikko Rautiainen.

5.1 Motivation

Numerous tools are available for quantification of transcripts using short reads. As mentioned in the previous chapter, modern bioinformatics algorithms such as Cufflinks ([Tra+10]), Kallisto ([Bra+16]) and Salmon ([Pat+17b]) rely on mapping reads on a reference transcriptome and estimate abundance. Read mapping is also the first step for fusion detection approaches such as TopHat-Fusion ([KS11]), SOAPfuse ([Jia+13]) and MapSplice ([Wan+10]). These algorithms align reads to a reference transcriptome using a "splice-aware" aligners. They detect fusion events by considering reads which span two different genes ([Kum+16]). Although efficient, the above methods deal with specifics related to short-read RNA-seq protocol. In other words, they treat biases inherent with short-read protocols and hence are not efficient with long reads. There are few tools that deal with transcript quantification using long reads. They are not able to assign the exact exon to which a read belong

and hence fall short of mapping an exon to a transcript. In regards to fusion detection, the read-length is not large enough to span the exonic junctions and hence accurate fusion-event detection is a challenge. To our knowledge, no work was done regarding long-read specific fusion detection.

5.2 Related work

As mentioned above, there are limited number of algorithms which deal with transcript quantification using long reads. Tools such as ToFU ([Gor+15]) and isONclust ([SM19]) cluster reads based on similar sequences. The idea behind this approach is that reads originating from the same gene or transcript would cluster together. Practitioners can quantify transcripts by counting the number of reads in a cluster. However, many genes have regions of similar sequences. Reads originating from these regions might get clustered together. This would produce sub-optimal quantification. Also, finding similarity between reads, especially pairwise, would be time and memory intensive. Algorithm SQANTI ([Tar+18]) rely on the output of ToFU for its abundance estimation, annotation, and quantification. However, it is designed only for reads generated from PacBio sequencers and does not support reads from other technologies like Oxford Nanopore sequencers

An alternative approach is to align long reads against a reference sequence and count the number of reads aligned to a transcript. However, the alignment of long reads is challenging due to the high error rate from the sequencers (ranging from 10-20%, [FWA19]). Hence, special alignment packages such as BLAT ([Ken02]), BLASR([CT12]) and Minimap ([Li16]) were developed to deal with the high error rate. Recently developed long-read specific methods such as TALON ([Wym+19]) and Mandalorian ([Byr+17]) use Minimap2 for aligning reads to reference transcriptome and base their quantification on the alignment output. Similarly, a study by [Son+19] gave Minimap2 input to Salmon and ran its quantification.

We observe that most of the above algorithms rely on alignment produced by the Minimap2 package. Given an input read dataset and a reference sequence, Minimap2 finds a seed in the reference sequence and extends the seed to find an optimal alignment. The quantification tools then assign the read to its corresponding aligned transcript. However, in the case of multiple alignments, the aligner output the alignment which has the maximum score. This makes the assignment of a read to a transcript biased towards how the primary alignment is selected. In other words, if a read maps to multiple highly-similar transcripts, the assignment becomes ambiguous. Hence, there is a scope of improvement in finding the correct alignment which would enable accurate quantification and gene-fusion detection.

The concept of splicing graph as a representation of the transcriptome has been used in many research works ([Heb+02b; Pat+17a; Gar+18]). Tools for variant calling, genome assembly and short tandem repeat analysers use graphs with nucleotide sequences as nodes and edges representing the adjancencies ([Pat+17a; Ant+16; Wic+17]). Furthermore, it has been studied that alternate splicing event can be detected using the splicing graph ([Den+18]). Hence, in this work we use alignments of reads to a splice-graph generated from reference transcriptome. Based on the alignments, we quantify transcripts and detect gene-fusion events in the concerned specie.

5.3 Method

In this work, we introduce Aeron, a long-read specific pipeline for transcript quantification and gene-fusion event detection. Aeron consists of three steps namely 1) index construction, 2) alignment of reads to the graph and 3) quantification and/or gene-fusion event detection.

5.3.1 Index construction

Index for the quantification and gene-fusion step consists of (a) a splice graph generated from reference transcriptome and (b) paths followed by annotated transcripts in the splice graph. Here, we discuss the steps followed to generate the splice graph and obtain the paths followed by transcripts in the graph.

Graph construction

The idea of splice-graph as a representation of a transcriptome was first explored by [Heb+02a]. Briefly, a splice-graph is a Directed Acyclic Graph (DAG) where nodes represents the splicing sites of a given gene and edges represents exons and introns between the sites. For our work, we wanted to represent all the possible alternative splicing events. Hence, we modify the splicing graph construction and term it as *gene-exon* graph. Figure 5.1 summarizes the graph construction process.

We define *DNA* sequence as a string consisting of characters from the set $\Sigma = \{A, C, T, G\}$. We define a gene g as a section of the DNA sequence which has protein coding regions. A gene g consists of multiple exonic and intronic regions. Different combinations of exonic regions (along with intronic regions in some cases) constitute different transcripts of the gene (see chapter 2 for more details). We term a base within g as exonic-base if it is overlapped by at-least one exon belonging to g (blue characters in fig.5.1). Otherwise, the base is termed as *intronic base* (black characters in fig. 5.1). A genomic position in g serves as a *border* position if it is the boundary of any exon belonging to g. 5′ boundary of an exon is considered as 5′ border (depicted as 5′ in fig.5.1) and the 3′ boundary of an exon is termed as the 3′ border (depicted as 3′ in fig.5.1). Hence, it can deduced from the above definitions that each exon of g is represented by a pair of the border positions $x = (a, b)$ where a and b are 5′ and 3′ borders respectively. We refer to a list of all 5′ borders and 3′ borders as *border list* ρ. This list can further be sub-divided into *acceptor-list* α containing all the 5′ borders and donor list δ containing all the 3′ borders. We order the members in each list incrementally.

Each exon of g represents a node in our *gene-exon* graph. If an exon goes through alternative splicing event, i.e, it has an alternate donor or an alternate acceptor site, then is split into sub-nodes. The split happens at the position of the alternate site. We formalize the split in the following way. We consider each node v_{ij} in the graph as sub-string $g[i...j]$ between two consecutive border positions $i \in \rho(g)$ and $j \in \rho(g)$. The node v_{ij} is created using the following formula:

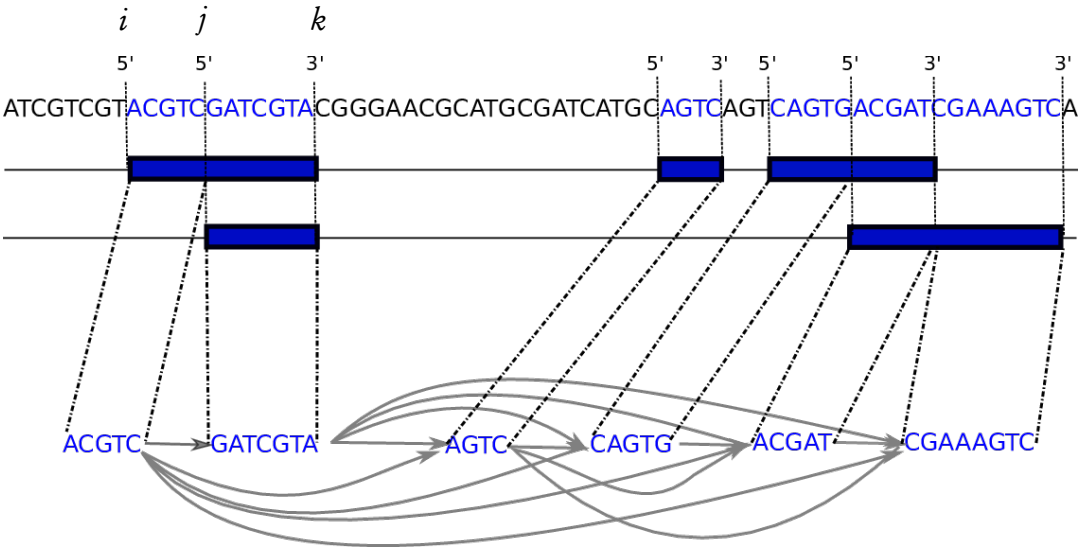

FIGURE 5.1: Gene-exon graph construction. Each exon sequence of a gene is considered as a node (blue boxes) of the graph. If the exon has an alternate donor or an alternate acceptor site, then the corresponding node is divided into subnodes. A node is connected to all the nodes downstream of it.

$$v_{ij} = \begin{cases} g[i \ldots j-1] & \text{if } i \in \alpha(g) \text{ and } j \in \alpha(g), \\ g[i \ldots j] & \text{if } i \in \alpha(g) \text{ and } j \in \delta(g), \\ g[i+1 \ldots j] & \text{if } i \in \delta(g) \text{ and } j \in \delta(g), \\ g[i+1 \ldots j-1] & \text{if } i \in \delta(g) \text{ and } j \in \alpha(g) \text{ and} \\ & g[i+1 \ldots j-1] \text{ is composed of exonic bases} \\ \phi & \text{if } i \in \delta(g) \text{ and } j \in \alpha(g) \text{ and} \\ & g[i+1 \ldots j-1] \text{ are intronic bases} \end{cases} \tag{5.1}$$

where $v_{ij} = \phi$ represents an null node.

For instance in figure 5.1, position i, j and k represents three border positions where i and j are 5′ borders and k is a 3′ border. The position i and j satisfy the condition 1 of equation 5.1. Hence, we take the sub-string $g[i...j-1]$ for the node between i and j which is equal to $v_{ij} = ACGTC$. Similarly, position j and k satisfy the condition 2 of equation 5.1. As a result, the node between j and k would be the sub-string $g[i...j]$ which is $v_{jk} = GATCGTA$.

As mentioned above, the gene-exon graph should take into account all possible alternative splicing events in a gene. Hence, there should always be a path in the graph that connects all exons and its sub-exons (sub-nodes). For instance, if there are three nodes v_{12},v_{34} and v_{56} corresponding to three exons x_{12}, x_{34} and x_{56} respectively. If a transcript is formed by splicing out x_{34} and using only x_{12} and x_{56}, then this event should be represented in the graph as a path passing through nodes v_{12} and v_{56}. Hence, in the graph, given two non-null nodes $v_{ii'}$ and $v_{jj'}$, an edge $e_{i'j}$ is created between all vertices with $i' < j$.

We construct a separate *gene-exon* graph for each and every gene belonging to a

specie. As a result, we end up with a set of graphs $\mathcal{G} = \{G_1, G_2, ..., , G_m\}$ where m is the number of genes present in a specie.

Sequence-to-graph alignment

An important step of the index construction is aligning all the transcripts to all the *gene-exon* graphs and obtaining the path traversed by each transcript. This is achieved by the recently developed GraphAligner ([RM19]). As defined in chapter 2, a path in a graph $G = (V, E)$ is a list of nodes $p = v_1, v_2, ..., v_n$ where $v_i \in V$ and edges $(v_i, v_{i+1}) \in E$. We define the *path sequence* as a sequence of nucleotide bases which is formed by concatenating the labels of all the nodes in p. The goal of our alignment is to find, for each transcript, the path in the graph with the smallest edit distance between the path sequence and the transcript sequence. GraphAligner is a seed-and-extend based aligner which finds the maximal exact matches between the query sequence and the node sequences and extends them with a bit-parallel dynamic programming algorithm ([RMM19]). The similarity between sequences is measured in terms of E-value which is calculated by the following formula:

$$E = Kmne^{-\lambda S} \tag{5.2}$$

where E is the expected number of spurious hits, K, and λ are parameters that depend on the scoring scheme, S is the alignment score, m is the database size in base pairs and n is the query size in base pairs. We use the number of base pairs in the graph as the database size. The K and λ were chosen to correspond to a scoring scheme with match score +1 and mismatch cost -2. The final alignment results contain a path, edit distance between the query sequence and the graph and start and end position of the alignment in the query sequence.

Alignment of transcripts to the graphs

Once all the gene-exon graphs are constructed, we align all the transcripts present in species to every gene-exon graph in $\mathcal{G}$. We obtain the path followed by each transcript in a graph. For each transcript t, we obtain the path which has the minimal edit distance to the transcript. Ideally, the path should come from the gene-exon graph of the gene from which t originates. Hence, each transcript will only have one path associated with it. We collect all such paths in set $\mathcal{P} = \{p_1, p_2, ..., p_m\}$ and name the set as *transcript-path* set

The gene-exon graph set $\mathcal{G}$ and transcripts-path set $\mathcal{P}$ constitute the index of Aeron. The index can be used for multiple samples of the same species.

5.3.2 Alignment of reads to the graph

Similar to the alignment of the transcripts to the graph, we align all reads in a dataset R against all the graphs in $\mathcal{G}$. We only consider alignments with an *E-value* below 1. In case there are multiple alignments with E-value below 1, we select the longest alignment. For each read, we then obtain path-sequence followed by the reads. Let $q_r = \{v_1\}.\{v_2\}....\{v_k\}$ be the path sequence of read r where $v_i \in V$ is the node sequence. We then compare q_r against every path sequence belonging $\mathcal{P}$. We align q_r to each and every path in $p_i \in \mathcal{P}$, which belongs to transcript t_i, using the Needleman-Wunsch algorithm ([NW70]). We discard all the secondary alignment and retain only those alignments which span end-to-end of a read. We define *overlap-score* of a

read r as the fraction of r which matches a given path-sequence of a transcript t_i. If the overlap score is above 20%, the read r is said to be aligned to t_i.

5.3.3 Quantification of transcripts

We assign the read to the transcript to which it aligns. A read may get aligned to multiple transcripts. In such cases, we assign the read to the transcripts with which it has the highest overlap score. In some cases, a read can get aligned to multiple transcripts with the same score. In this scenario, we assign the read to the transcript whose 3' end is closest to the 3' end of the read. We quantify a transcript by counting the number of reads assigned to it and converting these counts to Transcripts Per Millio (TPM). TPM for a transcript t_i measures the expected number of copies of t_i in one million transcripts. We calculated this by first taking the raw count of reads assigned to a transcript and dividing the number by the transcript-length. This gave us the transcript-level expression. We then summed up all the transcript level expressions and divided the resultant by 1,000,000 which gave us a scaling factor. TPM of t_i was then calculated by dividing the transcript-level expression of t_i by the scaling factor. To obtain the gene-level expression estimate, we summed the TPM values of all the transcripts belonging to a gene.

Evaluation

We tested the quantification step of Aeron on multiple datasets and compared the results generated against the expression estimates from Minimap. We used the widely known concept of Transcripts Per Million (TPM) for our evaluation. In addition to this, we also calculated the Mean Absolute Relative Difference (MARD) metric which was previously used for transcript expression comparison by [Pat+17b]. Briefly, we first calculate the *Absolute Relative Difference* using the following formula:

$$ARD_i = \begin{cases} 0, & \text{if } x_i = y_i = 0 \\ \frac{|x_i - y_i|}{x_i + y_i}, & \text{otherwise}, \end{cases} \tag{5.3}$$

where x_i and y_i are the true value and estimated value respectively for transcript i. For real datasets, we used the TPM values obtained using short reads (from salmon) as the true value and the TPM value obtained using long reads (from Aeron and Minimap2) as the estimated value. We also used simulated data generated using NanoSim ([Yan+17]) for our analysis. For calculating the ARD score for simulated datasets, we used the actual number of reads originating from a transcript i as true value and the number of reads assigned to i by Aeron as the estimated value. One can deduce from equation 5.3 that when the estimated expression is close to the true value, the ARD value tends to move towards zero. Hence, to score the overall performance of an algorithm, we can take an average of ARD values of all the transcripts as shown in equation 5.4.

$$MARD = \frac{1}{M} \sum_{i=1}^{M} ARD_i \,. \tag{5.4}$$

5.3.4 Fusion detection

One of the major applications of Aeron is to detect gene-fusion events using long-reads generated from third-generation technologies. The algorithm for fusion detection was developed and implemented by Mikko Rautiainen. Similar to the quantification step, we align the reads to gene-exon graphs. However, this time we retain all the secondary alignments of a read. We might obtain partial alignment of a read to multiple gene-exon graphs. Whenever a read get partially aligned to two different genes such that the end-points of the alignment are within 20bps with the read, then the read is said to support a *tentative-fusion* between the two genes.

We collect the genes which are part of tentative fusions and generate a fusion graph from it. A fusion graph is a combination of gene-exon graphs of the two genes involved in the tentative fusion and depends on which part of the read is aligned to which gene. For instance, if the first half of a read aligns to gene A and the second half aligns to gene B, we combine the gene-exon graphs of A and B. We place a *cross-over* node between the graphs of A and B and connect each base of A to the cross-over node by a directed edge. We also connect the cross-over node to each base of B by a directed edge. This way, an alignment can start from any position in A and end up in any position in B. We then align the reads to their fusion graphs keeping a condition that the alignment must span the entire read length.

A read getting aligned end-to-end against a fusion graph might just be by chance and is not an indication that the read supports the fusion event. To further analyze the predicted fusion event, we align the reads to the gene-exon graphs of tentative-fusion genes. Again, the alignment must span the entire read length. We calculate the difference between the lowest edit distance between the read sequence and the gene-exon graph and the lowest alignment edit distance between the read sequence and the fusion graph to obtain a *fusion score*. At each point, a read can support only one fusion event. We decide this by keeping the alignment of a read with a fusion graph which has the lowest edit distance. Read alignment whose fusion score is below a user-defined threshold is discarded. Similarly, a read alignment whose alignment error rate to the fusion graph is above 20% is discarded. We then consider the paths of the remaining fusion alignments as predicted fusion transcripts. When multiple reads align to the same fusion graph and cross over at the same exon, we consider them as the same fusion event. We then select one of them is arbitrary as the fusion transcript. Similarly, if multiple reads align to the same fusion graph but cross over at different exons, we consider them as separate events. We output the list of predicted fusion transcripts as path-sequence.

5.3.5 Data retrieval

For analyzing Aeron, we used a novel dataset with 2.7M reads from K562 cells with an average read length of 750bps. The dataset was generated in the Genome Institute of Singapore, Singapore. Apart from these, we used 25M reads from the NA12878 dataset ([Jai+17b]) with an average read length of 1030bps downloaded from SRA run browser. For the alignment step, we used the chromosomes and transcripts from GRCh38.p12 ([Sch+16]). We also aligned the datasets against the human reference genome using Minimap2. We filtered out all the secondary alignments from the resultant SAM file. For each transcript, we counted the number of reads aligned to it.

We then converted the count values to Transcripts Per Million (TPM). As in the case of Aeron, to achieve gene-level expression estimates, we summed the TPM values of all transcripts belonging to a gene.

To compare the expression estimates obtained from long reads (using Aeron and Minimap2) against estimates obtained using short reads, we downloaded two short-read datasets: one for K562 cell which consisting of 113M paired-end reads (SRX498124, [Wan+19]) and second for NA12878 which consisted of 188M paired-end reads (ERX329308, [Kil+13]). We then calculated the expression for both the datasets using Salmon (v0.11.12) with default parameters except for *k*-mer parameter which was set to 17.

5.3.6 Parameter optimization

As mentioned above, Aeron using GraphAligner for aligning reads against a sequence-graph generated from the reference genome. Two of the most important parameters for GraphAligner are the *seed-length* and the (number of seeds) (NOS) which are used for the seeding step of the aligner. Seeds hits are exact matches between portions of the reads and parts of the node sequence. Hence, the higher the seed-length lesser than the number of seeds found and alignments achieved. Similarly, the number of seeds denotes how many available seeds are used to compute an alignment. Hence, a high number of seeds and a small seed-length are required to achieve an accurate assembly. However, a very small seed-length might result in spurious alignment. Similarly, a very high number of seeds would increase the runtime of the pipeline drastically. We wanted that the default values of these parameters should give us a good accuracy in a reasonable runtime. For this, we first simulated 1M oxford nanopore reads using NanoSim (v2.5.0, [Yan+17]) in transcriptome mode. We gave the novel K562 ONT data as the reference to NanoSim to create the training read profile.

We performed several runs of Aeron on the simulated dataset. Each run consisted of a different combination of seed-length and the number of seeds. We measured the accuracy of each run using the MARD score (see section 5.3.3). Figure 5.2 shows the effect of varying seed length and the number of seeds on the runtime (x-axis) and the MARD score (y-axis). The curves in the graph represent a single NOS parameters and the points on the curve represent the different seed length. We see an improvement in accuracy when we increase the number of seeds. Note that a MARD score is inversely proportional to accuracy. Hence, a lower MARD score depicts an accurate expression estimation. However, we observe that setting the NOS parameter too high results in an increase in runtime. We also observe that with the increase in seed-length, the accuracy also goes down. We selected a combination of parameters (NOS=15 and Seed-length=17) which seemed to be a good trade-off between the accuracy and the run-time. However, please note that this combination might be data-specific. Different datasets might result in different combinations.

5.4 Results

In this chapter, we present the Aeron pipeline which quantifies transcripts and predicts gene-fusion events using alignments of long-reads against sequence-graphs.

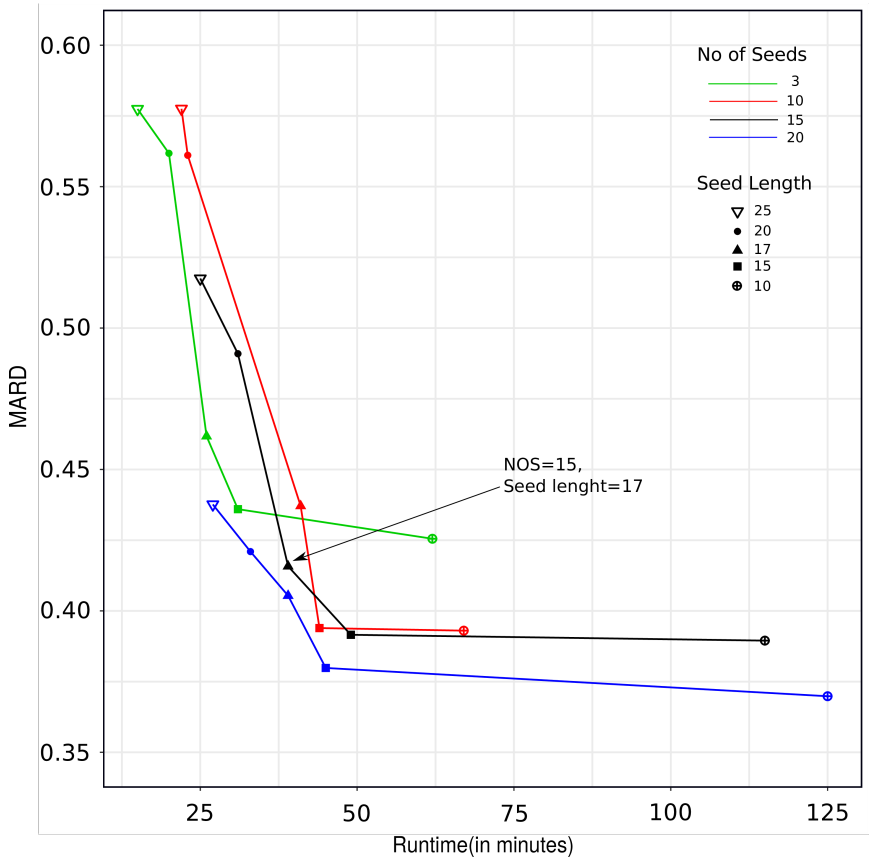

FIGURE 5.2: Parameter selection for Aeron. The grah represents different runs of Aeron having different combinations of seed length and number of seeds (NOS) parameters. Each curve in the graph represents a single NOS parameter and each point on a curve represents a single seed length parameters.

Figure 5.3 depicts the workflow of the quantification step of Aeron. We begin by constructing a gene-exon graph for each gene in the genome. In the gene-exon graph, each exon belonging to the gene constitutes a node. If an exon has an alternate donor/acceptor site, we divide the corresponding node into sub-nodes where the split occurs at the alternate donor/acceptor site. We connect the 3′ end of a node to the 5′ end of all the nodes downstream of it (figure. 5.3a). This way, any alternate splicing event can be mapped to the graph. We represent a transcriptome as the set of all gene-exon graphs. We proceed by aligning all the transcripts against all the gene-exon graphs are record the path followed by each transcript in the graph (figure 5.3b). The above steps mark the indexing phase of Aeron. We can use the same index for multiple datasets belonging to the same species.

In the next phase, we align long reads against all the gene-exon graphs (figure. 5.3c). We compare the alignment of each read against the set of paths followed by transcripts in the gene-exon graph (figure. 5.3d). We consider a transcript t as expressed if there is a read r such that: 1) at-least 20% of the path covered by r is contained in the path traversed by t and 2) the E-value of the alignment between r and the gene-exon graph is less than 1.0. We assign read r to transcript t with a score s, where s is the percentage of the path of r contained in t. If a read aligns to two different transcripts with the same score, we assign the read to the candidate transcript whose 3′ end is closest to the 3′ end of the read. This idea stems from the concept that long-read sequencers generate sequences from the 3′ to the 5′ end. Hence, there is a certain bias towards the 3′ end of the transcripts ([Dep+19]). We then perform the final quantification of a transcript t by counting the number of reads assigned to t and converting the number into Transcripts Per Million (TPM).

5.4.1 Performance of Aeron on different sequencing protocols

In this work, we tested Aeron on reads generated from Oxford Nanopore (ONT) sequencing technologies. Like most of the short read technologies, oxford nanopore technology synthesizes complementary DNA (cDNA) and amplifies it using PCR. An alternate protocol is to directly synthesize the single stranded RNA (DirectRNA) ([Dep+19]). This reduces the amplification biases introduced during the PCR site. We downloaded two ONT datasets from NA12878 cell line generated from the above two protocols. The first one contained 15M sequences obtained using the cDNA protocol and the second 10M sequences obtained using the DirectRNA protocol. We ran Aeron with default parameters on both the datasets and measured the performance in terms of the TPM values. Figure 5.4 shows the distribution of reads generated from both the protocols along the transcript length. As expected, we see most of the reads aligned towards the 3′ end of the transcripts. We see a stronger bias in the alignments of Direct RNA reads as compared to the reads from the cDNA protocol.

We then computed the expression levels of the know ENSEMBL ([Zer+18], v92) genes and transcripts for both the datasets. Out of 58,336 annotated genes and 203,675 annotated transcripts, we obtained expression estimates of 28,584 genes and 102,748 annotated transcripts in the case of cDNA data and 28,021 genes and 107,030 transcripts in case of Direct RNA data. We then compared the gene and transcript level TPM values for both the protocols. We calculated the Spearman correlation between the expression estimates from cDNA data and the expression estimates from Direct RNA data. Figure 5.6 compares the expression estimates of NA12878 data

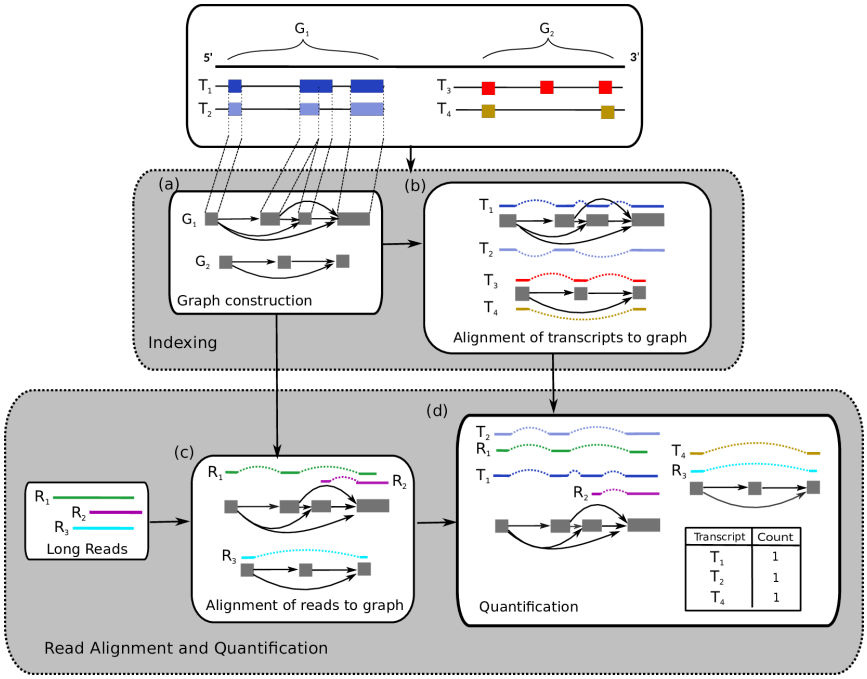

FIGURE 5.3: Workflow of Aeron. (a) Each gene in the genome is represented as a gene-exon graph where exons are represented as nodes. The 3′ end of each node is connected 5′ end of all the nodes downstream of the node. (b) Transcripts are aligned against the gene-exon graph and path traversed of the transcripts are recorded. (c) Long RNA reads are aligned against all the gene-exon graphs. (d) The paths followed by the reads are compared against the paths followed by the transcripts. A read is assigned to a transcript if the atleast 20% of the path followed by the read is contained in the transcripts.

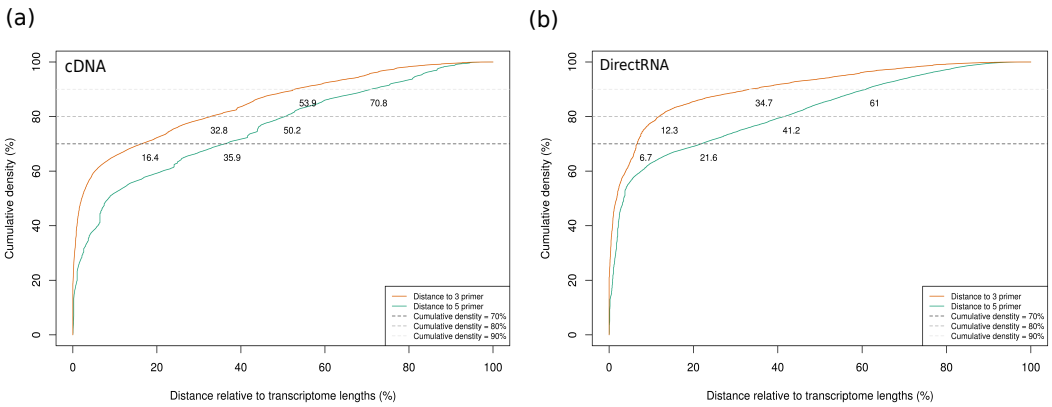

FIGURE 5.4: Distribution of reads along the transcript in NA12878 datasets generated from (a)cDNA protocol (b) Direct RNA protocol

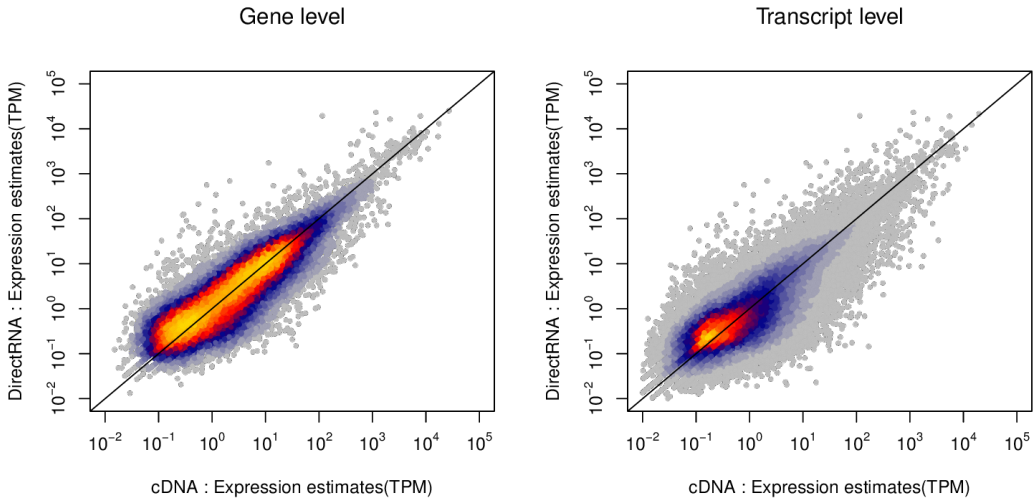

FIGURE 5.5: Comparison of expression estimates obtained from NA12878 dataset generated from cDNA and Direct RNA protocol.(a) Comparison of gene-level expression estimates. (b) Comparison of transcript level expression estimate.

generated from the two protocols. We observe that at gene-level (figure 5.6a), the expression estimates are highly correlated with the spearman coefficient of 0.90. However at transcript level, the correlation drops to 0.68 (figure 5.6a). A reason for this might be the position of transcript sequences within a gene. Within a gene, many transcripts may overlap with each other. A high percentage of these overlapping transcripts have the same 3' end position. In cases, if the read aligns to the overlapping transcripts with the same score, we assign the read randomly to one of the transcripts. Hence, we may encounter discrepancy in transcript level expression estimates.

To check whether Aeron generates reproducible quantification, we combined the cDNA and Direct RNA datasets and divided the combined datasets into three subsets. We ran Aeron separately on the three subsets using the default parameters. We calculated the transcript-level expression for the three subsets. We found the expression estimates to be highly correlated with each other asserting the fact that Aeron generates reproducible expression estimates.

5.4.2 Comparison of Aeron against expression estimates from Minimap2

To test the performance of Aeron over multiple datasets, we ran Aeron on 2M novel K562 data and 25M Na12878 data. We aligned and quantified both the datasets against the annotated human Ensembl transcriptome. As mentioned in section 5.2, current long read specific algorithms depend upon alignments from Minimap2 for expression estimation. Hence, we also aligned the NA12878 and novel K562 against a reference genome using Minimap2 and filtered out all the secondary alignments. We assigned a read to a transcript if it aligns with the transcript. We then counted the number of reads assigned to a transcript and converted the number into TPM values.

In the case of Aeron, we achieved more than 80% alignment rate for K562 and around 96% alignment rate for NA12878. We looked into the read length and the

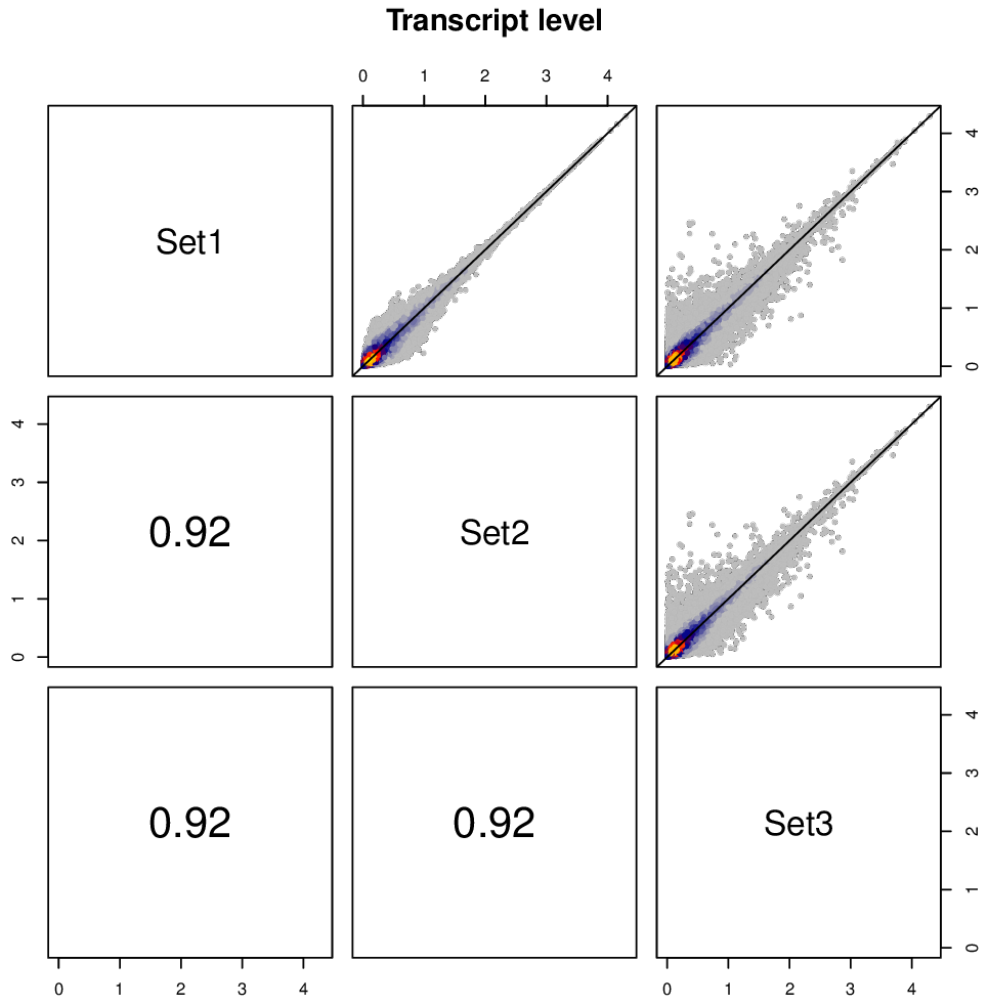

FIGURE 5.6: Comparison of transcript level expression estimates of Aeron using three different subsets of NA12878 data.

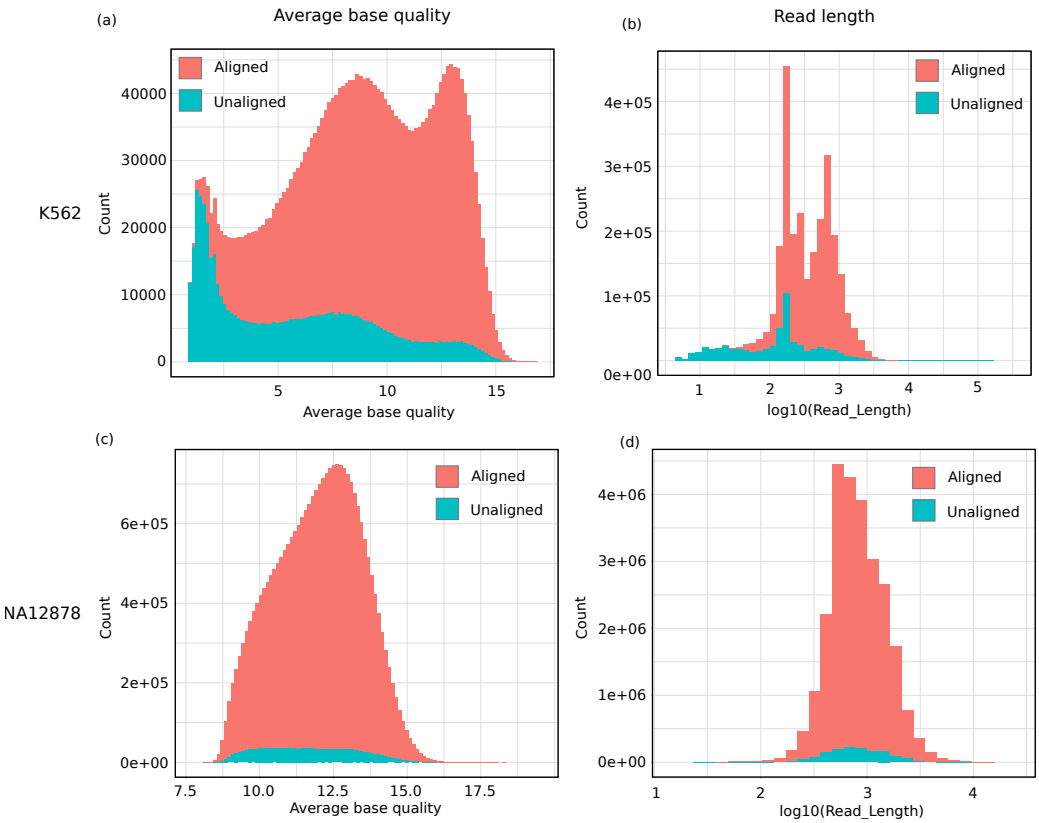

FIGURE 5.7: Average base quality distribution (a and c) and read-length distribution (b and d) within K562 (first row) and NA12878 (second row) dataset.

base quality of the reads which were left unaligned in both the datasets. We averaged the quality scores of individual bases in the read and termed this as *average base quality* of the read. Figure 5.7 shows the properties of the reads which were left unaligned in both the datasets. As expected, many of the unaligned reads (blue bars in the plot) were either had either a low average base quality (5.7a and c) or were shorter in length (5.7b and d). In Aeron, shorter reads are mostly overlapped by longer reads in the graph. Hence, the aligner considers these reads as secondary alignment and filters them out. In the case of low-quality bases, these are generally sequencing errors. Hence, when a read has low average base quality, it is difficult to determine its origin correctly. The K562 dataset was a combination of data produced by three different sequencing runs (three different peaks in (5.7a)). The first sequencing run generated a lot of low-quality bases and the aligner was not able to align most of the reads from this run.

We compared the performance of Aeron against estimates from Minimap2 at both gene and transcript levels. Table 5.1 summarizes the results obtained from the comparison. We see that in K562, Aeron can assign 50% more reads to transcripts as compared to Minimap2. In the case of the NA12878 dataset, Aeron can map 3% more reads to transcripts. To check whether these additional assignments are correct, we generated expression estimates from short-read data using Salmon. Taking the salmon estimates as true value, we calculated the MARD scores for Aeron and Minimap2. We also calculated the Spearman correlation between the estimates from the

Gene level						
Dataset	**Aeron**			**Minimap2**		
	#reads mapped	**Correlation**	**MARD**	**#reads mapped**	**Correlation**	**MARD**
K562 (2.7M)	2,167,286	**0.833**	**0.350**	1,074,411	0.704	0.428
NA12878 (25M)	23,902,112	**0.822**	**0.349**	23,211,716	0.778	0.37
Transcript level						
Dataset	**Aeron**			**Minimap2**		
	#reads mapped	**Correlation**	**MARD**	**#reads mapped**	**Correlation**	**MARD**
K562 (2.7M)	2,167,286	**0.297**	**0.635**	1074411	0.209	0.659
NA12878 (25M)	23,902,112	**0.270**	0.664	23211716	0.207	**0.637**

TABLE 5.1: Spearman correlation and MARD between Transcripts Per Million (TPM) at gene level obtained from Aeron/Minimap2 using Oxford Nanopore Sequencing (ONT) data and TPM at gene and transcript level obtained from Salmon using Illumina data. The size of the dataset is depicted in brackets next to the name.

long reads (using Aeron and Minimap2) and the estimates from the short reads (using Salmon). In these comparisons, we considered all genes and transcripts present in the human genome according to Ensembl annotation (v92). We found the Aeron estimates to be closer to the expression estimates from short reads evident by the higher correlation value and lower MARD scores at both gene and transcript levels. An exception to this is in th case of the MARD score of NA12878 where Minimap2 performs better than Aeron. Note that, MARD score is based on the difference between the estimated value and the true value. So if the estimated value to closer to the true value, the MARD score would move closer to 0.

We observe from the table 5.1 that the correlation between the expression estimates from long reads and the expression estimates from Salmon at transcript level is constantly low. We also a similar behavior in MARD score where it is constantly high for both the datasets. As mentioned in section 5.4.1, presence of overlapping transcripts might be one plausible reason for this. Another reason might be the presence of high percentage of short length transcripts in the transciptome. Error-prone reads generated from these transcripts might not have enough length to get aligned to the correct transcript. Also, if the shorter transcript is completely overlapped by a longer transcripts, a read may get aligned to the longer transcript instead of the shorter one. Hence, we might not be able to quantify such short-length transcripts. To test this hypothesis, we applied a length threshold on the transcripts. We removed all the transcripts, whose length is below the threshold, from the transcript pool and recalculated the correlation value. Figure 5.8 shows the effect of transcript length cutoff on the expression estimation. As expected, when we remove transcripts of smaller length, the correlation with Salmon estimates increases. For a range of transcript length cutoff of 0-10000bps, the correlation improved from 0.29 to 0.64 for K562 cell line and 0.27 to 0.74 for NA12878 dataset. This asserted our hypothesis that the presence of short transcripts affects the quantification at transcript-level.

One of the drawbacks of long read based quantification is the inability to quantify low expressed regions. We wanted to check whether Aeron can map and quantify low expressed regions. Figure 5.9 shows a scatter plot of gene expression estimates of Aeron (first column) and Minimap2 (second column) applied on the novel K562 data and the NA12878 data. For this plot, we only take the genes which have non-zero expression estimate by either Aeron or Minimap2. Again, we take the expression estimates from Salmon as the true value and depict that on the y-axis of

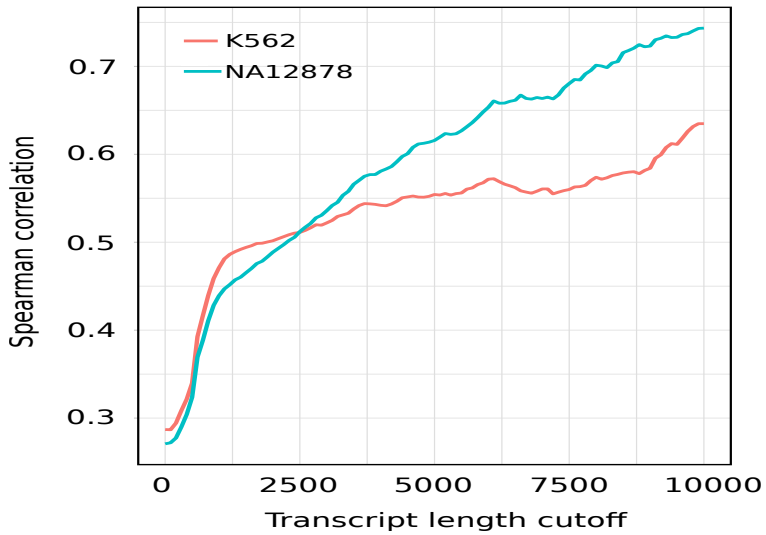

FIGURE 5.8: Effect of transcript length on the spearman correlation. The x-axis denote the transcript length cut-off applied Aeron. All transcripts whose length is below the cut-off length were removed from the reference transcript pool. The y-axis denotes the spearman correlation with expression estimates using short reads when the length cut-off ia applied.

each plot. We observe that Aeron can achieve estimates closer to the true value resulting in the majority of the points lying close to the diagonal. We observe that Aeron is able to detect more genes at lower range of expression (1-5 TPM) as compared to Minimap2. A plausible reason for this behaviour might be that few reads originate from these low-expressed regions. Since these reads are error-prone, minimap2 is not able to map the reads to the correct region of origin. The conditions of mapping is less stringent in case of Aeron where only 20% of the path covered by a read should be contained in the transcripts. Hence, the reads originating from low expressed transcripts are successfully mapped back to the correct genomic region.

Next, we wanted to check whether the alignments produced by Minimap2 could be processed differently to produce better quantification. Recently, [Son+19] proposed an alternate approach for quantifying transcripts by using Salmon on the BAM file produced by Minimap2. Simultaneously, Soneson et al also ran Salmon in quasi-mapping mode with the index generated from ENSEMBL cDNA reference fasta file. We compared these two approaches with the quantification produced from Aeron at gene and transcript level. Table 5.2 summarizes the results obtained. We found that Aeron was still able to perform better than the approaches suggested by Soneson ey al. As mentioned in section 5.2, Salmon is designed with the specifics and the error-model of short-read sequencing technology. Since the error model for long-read technologies is different, Salmon is not effective when it comes to quantification with long reads.

5.4.3 Gene-fusion detection

Another major objective of Aeron is to detect gene-fusion events using alignments of long reads against gene-exon graphs. The gene-fusion detection algorithm was developed and implemented by Mikko Rautiainen from the *Algorithms for Computational Genomics* group at Saarland University, Germany. Briefly, we align the reads

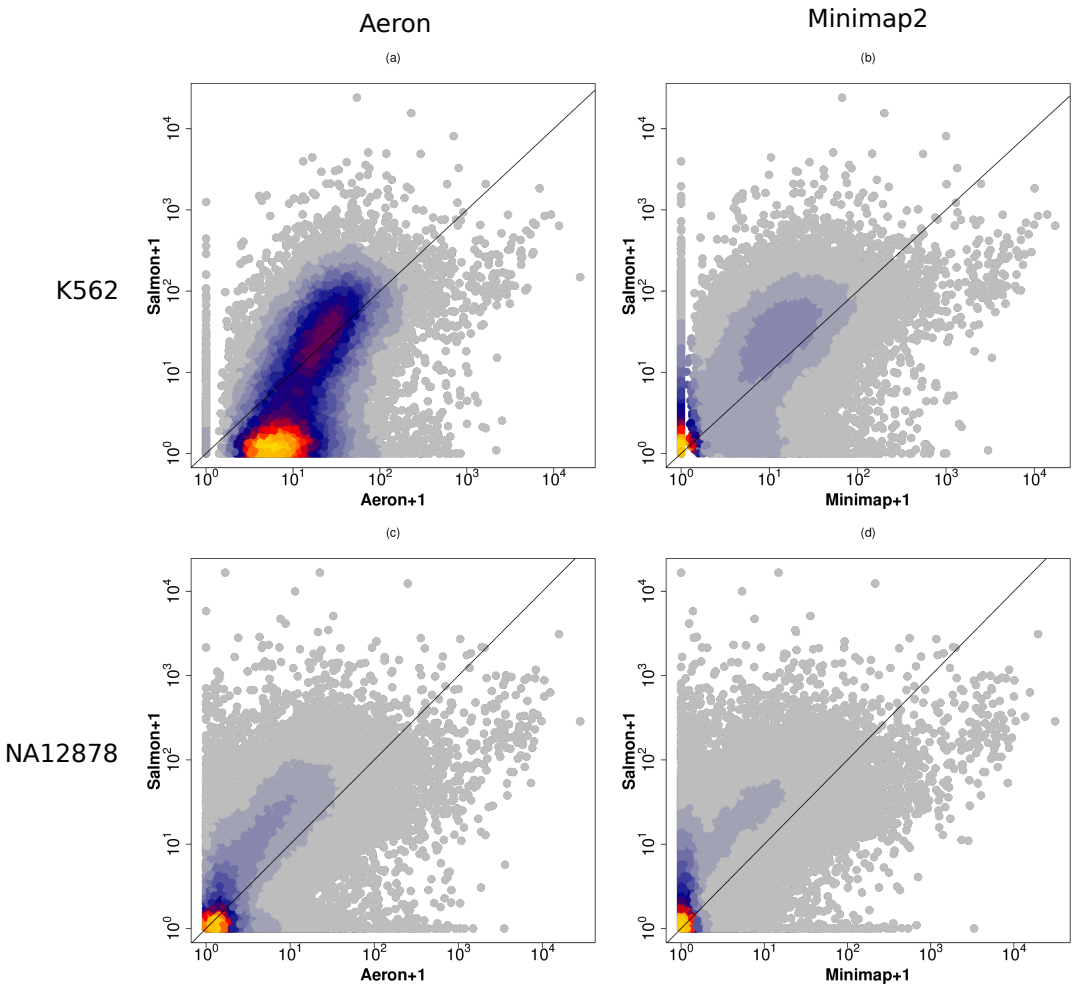

FIGURE 5.9: Comparison of gene level expression estimates obtained from Aeron (a and c) against the expression estimates obtained from Minimap2 (b and d) for the two datasets - K562 (a and b) and NA12878 (c and d). The expression estimated using long reads are the shown in the x-axis and the estimates obtained using short reads are shown in the y-axis. For clarity, expression estimates have been log transformed with a base 10.

Gene Level				
Dataset	Aeron	Minimap2	Salmon (Quasi-Mapping)	Salmon (Alignment-based)
K562 (2.7M)	**0.836**	0.704	0.706	0.700
NA12878 (25M)	**0.823**	0.778	0.759	0.772
Transcript Level				
Dataset	Aeron	Minimap2	Salmon (Quasi-Mapping)	Salmon (Alignment-based)
K562 (2.7M)	**0.297**	0.209	0.233	0.215
NA12878 (25M)	**0.276**	0.233	0.257	0.221

TABLE 5.2: Alternate approaches for quantifying transcripts using long reads. The table shows the spearman correlation between the expression estimates obtained using short reads (using salon) against expression estimates obtained using Aeron(column 2), only minimap2 output(column 3), salmon in quasi mapping mode with long reads as input (column 4) and salmon in alignment-based mode on output generated from minimap2 (column 5).

against the *gene-exon* graph. We consider reads which get partially aligned to different genes. We define a pair of genes to support *tentative-fusion* if: 1) a read gets partially aligned to both the genes and 2) the alignment end-point of the first part of the read and the alignment starting point of the second part of the read is within 20bps of each other. The fused sequence of both the genes forms the tentative-fusion sequence. A read may support multiple tentative-fusions. We then take the gene-exon graphs of genes involved in a tentative-fusion and combine them to form a fusion-graph. We add a temporary node between the two gene-exon graphs. We connect all the bases of the first graph with this temporary node. Similarly, we connect the temporary node is to all the bases of the second gene-exon graph. We generate a fusion graph for each pair of genes participating in a tentative fusion. We then perform an end-to-end alignment of reads against the fusion graphs. If reads are not able to align end-to-end to the fusion graph, then the fusion-graph and the corresponding tentative-fusion are discarded. We term this step as the *graph filtering* step. We then align the reads to the individual gene-exon graph of genes which are part of the remaining tentative fusion. We define *fusion score* as the difference between the scores obtained from fusion graph alignment and the gene-exon graph alignment. We then filter out alignments whose fusion score is below a user-defined threshold. A read can support only one tentative fusion. Hence when we remove an alignment, we also remove the corresponding fusion graph. We term this step as *fusion filtering step*. We then take the sequence of the alignment along the remaining fusion graph as predicted fusion event.

Performance of fusion detection on simulated data

To test the performance of gene-fusion step of Aeron, we simulated fusion transcripts of different lengths. Briefly, we randomly selected two different genes carved out a sequence section of the length l from both the genes. We concatenated the two sections to form a fusion gene of length $2l$. We then took a random substring from transripts belonging to each gene and the substrings were concatenated to form a fusion transcripts. We then simulated reads from these fusion transcripts using NanoSim and ran the fusion detection pipeline using the simulated reads.

We show the performance of the fusion detection step of Aeron in figure 5.10. We can observe from the precision-recall curve (figure. 5.10a) that fusion of smaller length are difficult to detect as the recall saturates at 15%. This might be due to the high error rate of ONT technologies and short read-length generated from the shorter fusion transcripts. This makes it difficult to map the entire reads back to the transcripts. As the fusion length increases, we see an improvement in the recall rate. We achieve a maximum recall rate of 95% for the fusion lengths between 700-1000bps.

Next we wanted to check the number of simulated fusions we are able to recapture using Aeron. For this we first simulated 450 fusion events for different sizes and divided them into size ranges from 100bs-1000bps such that each size range has 50 simulated fusions. We then ran the Aeron pipeline on the simulated dataset. First we considered all the tentative-fusions which were obtained after first step, i.e, after the partial alignment of reads to different genes (depicted as tentative in figure 5.10a). We see that this is able to predict more than 90% of the fusion events from size range 600-100bps. However, we also obtained around 28,696 false positives. We then filtered out fusion transcripts which couldn't surpass the graph filtering step. We observed a similar curve to the tentative fusion. However, the number of false

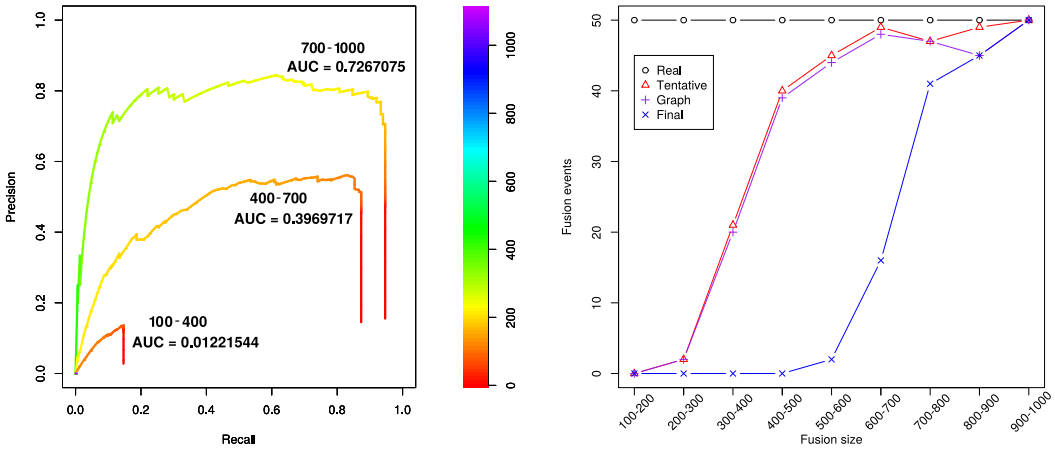

FIGURE 5.10: Performance of Aeron's fusion detection algorithm on simulated data. (a) precision-recall curve of fusion detection with simulated for fusions where length l ranges from 100-400bps(bottom curve), 400-700 bps(middle curve) and 700-100bps(top curve). Note that the total length of the fusion transcript is $2l$. (b) number of detected true fusion events per fusion size. The curve depicts the number of simulated fusions (black line) and the fusion detected by Aeron (blue, red and purple line).

positives was reduced to 49. Further, we filtered out all the fusion transcripts which couldn't pass the fusion score filter. Here, we could only recreate fusions which were greater than 600bps in length. However, the number of false postives was drastically reduced to 20 in the final step. We see that shorter fusions are not detected even in the tentative fusion phase; this is most likely due to the high error rate preventing the reads from getting aligned to the correct genes.

Performance of fusion detection on real data

So far, we saw the performance of Aeron's fusion detection on a simulated dataset. However, the simulated dataset does not capture some of the complexities of a real gene-fusion event. Hence, we tested Aeron on the novel K562 and the NA12878 datasets. K562 is an important cancer cell line with some known fusion events. We used the NA12878 dataset as a control since no known fusion event is present in this cell line. The initial run of the pipeline resulted in 25 events in K562 and 24 events in NA12878. However, these events involved mitochondrial genes and gene fused with its pseudogenes. We also removed fusion events where alignment to one gene was worse than alignment to the second gene. So far we were only aligning against the Ensembl annotated transcripts. However, there might be a transcript t not annotated in Ensembl which is similar to the fusion transcript. The read might originate from t. Hence, we aligned the fusion transcripts against NCBI's annotation (release 109) and removed all those predicted fusion transcripts aligned to a transcript from NCBI's annotation. Finally, we were left with 8 events for K562 and 2 events for NA12878.

Table 5.3 summarizes the fusion events obtained from Aeron. The predicted events included the well know BCT-ABL event ([Kur+03]). Other events included the HGB2-HGB1 event which has a high read support ([Lee+10]). The predicted

K562						
Ensembl ID	**Gene**	**Chromosome**	**Ensembl ID**	**Gene**	**Chromosome**	**Support**
ENSG00000196565	HBG2	chr11	ENSG00000213934	HBG1	chr11	89 reads
ENSG00000186716	BCR	chr22	ENSG00000097007	ABL	chr9	2 reads
ENSG00000257949	TEN1	chr17	ENSG00000250506	CDK3	chr17	2 reads
ENSG00000204177	BMS1P1	chr10	ENSG00000188234	AGAP4	chr10	2 reads
ENSG00000241553	ARPC4	chr3	ENSG00000214021	TTLL3	chr3	4 reads
ENSG00000198056	PRIM1	chr12	ENSG00000196531	NACA	chr12	3 reads
ENSG00000198056	PRIM1	chr12	ENSG00000196531	NACA	chr12	3 reads
ENSG00000164867	NOS3	chr7	ENSG00000185345	PRKN	chr6	2 reads
NA12878						
Ensembl ID	**Gene**	**Chromosome**	**Ensembl ID**	**Gene**	**Chromosome**	**Support**
ENSG00000223361	FTH1P10	chr5	ENSG00000162734	PEA15	chr1	3 reads
ENSG00000172493	AFF1	chr4	ENSG00000162244	RPL29	chr3	2 reads

TABLE 5.3: Predicted fusion events for K562.The first 6 columns describe the two genes involved in the fusion. The column "Support" counts the number of reads whose primary alignment covers the fusion breakpoint and 150bp from both sides of it.

TEN1-CDK3 and BMS1P1-AGAP4 were read-through events reported earlier ([Pra+10; Str+02]).

5.5 Discussion

Sequencing technologies like Oxford Nanopore and PacBio overcome several shortcomings of short-read technologies. They are producing reads which almost spans the entire transcript length. Hence, there is a new interest in transcript quantification and gene-fusion detection using only long reads. In this work, we propose Aeron, a novel pipeline that quantifies transcripts and predicts gene-fusion events based on an alignment of reads against a sequence graph. We used GraphAligner ([RMM19]) to align reads to the transcriptome and assigned reads to a transcript based on the position of the alignment within the transcripts. We tested Aeron on two different datasets and compared the results to expression estimates derived from alignments using Minimap2 and found Aeron to be performing better. We found a high correlation between estimates from Aeron and expression estimates from short-read data (using salmon). The assignment of a read to a transcript was based on Minimap2 outputs was highly biased towards the primary alignments. As a result, the quantification produced based on Minimap2 output tends to be sub-optimal. We also show that Aeron does not depend on the sequencing protocol used to generate the reads. This was evident from the high correlation between the Aeron runs on reads generated from cDNA and the reads generated by native RNA.

However, the quantification step of Aeron still suffers from some drawbacks. Mainly, the transcript level quantification is still not as accurate as of the gene-level quantification. In other words, although we show improvement over existing methods, the assignment of many reads to the correct transcript of origin is still a challenge for Aeron. There might be many reasons for this behavior. We show in results that the short transcripts hinder the performance of Aeron. Many transcripts in the reference sequence were a shorter version of a longer transcript. If a read originating from the shorter transcripts, is not long enough to cover the transcript, it becomes difficult to map and assign the read back to the transcript. We mitigate this problem at gene-level where we add up the expression estimates of all transcripts belonging

to a gene. The short length of a read posses another hindrance towards accurate transcript-level expression estimation. If an error-prone read is short, then it is difficult to map the read back to the transcriptome.

This brings our attention to the high error-rate in long-read sequencing technologies. We can encounter an error rate of 5%-7% in Oxford Nanopore which makes aligning of reads extremely difficult. Hence, one can error correct the reads before aligning and quantifying transcripts using them. Currently, there is no long-read specific error correction method for transcriptomic sequences. Recently, [Lim+19] studies the effect of using genomic long-read specific error correction methods on transcriptomic data. They found that genomic methods are effective to a certain extent. However, they affect the gene and isoform diversity which would harm the quantification process. [RMM19] proposed an error correction application using GraphAligner to align long reads against the de Bruijn graph generated from short reads. However, this method has only been tested on genomic data. Also, this is a hybrid method which requires short reads for its implementation.

We also propose a fusion detection algorithm using the alignments of long reads against the sequence graph. We tested the approach on simulated data and the K562 dataset. We observed that the efficiency of the fusion detection approach depends highly on the length of the fusion transcripts. Detecting the shorter fusion transcript is still a challenge for Aeron which can be further looked into. The application of Aeron on the K562 data resulted in the detection of known and novel gene-fusion events. These events included the well known BCR-ABL1 and HGB2-HGB1 gene-fusions. We realized that the gene-fusion approach still needed some curations to weed out false positives which would be worked upon in the future.

In conclusion, we feel that Aeron would be of high interest amongst practitioners since long read transcriptomic technologies are catching fuel. With the improvement in third-generation sequencing, the accuracy of our tool would also increase. For instance, an improvement in base-calling would result in reduced error-rate. This would improve the alignment of reads to the transcriptome resulting in better quantification and gene-fusion detection. An accurate transcript quantification and gene-fusion event detection can open a pandora of transcriptomic information that could be used further to gain novel insights into the genomic landscape of a species.

5.5.1 Availability and Implementation

Aeron is an open source pipeline which is managed using SnakeMake. The modules are written in python and the assembler is written in C++. It can be downloaded via GitHub ((github.com/SchulzLab/Aeron). We tested the algorithm in Linux environment.

Aeron runs in three steps. In step 1, a reference transcriptome is given and Aeron generates a gene-exon graph. The transcripts are mapped to the graph and the paths traversed by each transcript are recorded. This step has to be performed only once and the output can be reused for multiple read datasets. Step 2 consists of alignement of reads against the graph. Aeron then performs the quantification based on the overlap between the paths followed by the read in the graph and the paths followed by transcripts. In step 3, the alignment of reads against the graph is reanalysed to predict possible gene fusion events.

We direct the user to go through the readme file given with the tool to get a detailed picture of the functioning of the algorithm.

Chapter 6

Summary and Outlook

In this chapter, we revisit all the software and algorithms developed in the scope of thesis. We try to address the limitations of each software and discuss the possible improvements to enhance their functionality.

6.1 Read normalization (ORNA and ORNA-Q/K)

Optimized Read Normalization Algorithm (ORNA) is a software tool to normalize the high coverage RNA-seq dataset. The goal of ORNA is to remove redundant reads in an informed way without compromising on the *k*-mer information contained in them. The software is open source and can be accessed via GitHub (https://github.com/SchulzLab/ORNA).

Most of the de novo assemblers use *k*-mers from the reads to build a de Bruijn graph. The assemblers consider *k*-mers as the nodes of the graph and connect them based on the overlapping of sequences between two *k*-mers. They then generate assemblies by traversing various paths in the graph. Hence, while normalizing the data, it is important to preserve all *k*-mers from the original dataset. ORNA considers read normalization as a Set Multi-Cover problem. The set of all *k*-mers from the original dataset is considered as the universe. A minimum set of reads required to cover each element of the universe a certain number of times (say t) is obtained and termed as the normalized dataset. The assemblers rely on the relative difference of abundance between neighboring node *k*-mers to perform graph simplification. Hence, ORNA sets a different value of t for each *k*-mer based on the *k*-mers abundance. Thus, it also retains the relative differences of abundances between neighboring node *k*-mers.

ORNA can be applied de novo to any RNA-seq dataset. It is computationally convenient with linear run time which depends on the size of the dataset and the read-length. We utilize the *k*-mer counting algorithm from the GATB library which makes it memory and runtime efficient. In our evaluation, we show that ORNA retains all *k*-mers from the original dataset. We compared the assembly performance of ORNA with other normalization algorithms namely Diginorm and the inbuilt in silico normalization step of Trinity assembler (TIS). The effect of the retaining all the *k*-mers is seen here the assemblies generated from ORNA reduced datasets have more number of annotated transcripts as compared to the assemblies generated from Diginorm and TIS reduced datasets. Also, the memory and runtime required by ORNA are comparable to that of Diginorm and TIS. We also noticed that error correcting the reads before normalization resulted in the removal of more number of reads. It

also improved the quality of the assembly produced.

However, there is still scope for a major improvement here. The goal of any normalization algorithm should be to achieve the same quality of assembly which is obtained from an unreduced dataset. Although we noticed that assemblies from ORNA reduced datasets are the closest to the assembly from unreduced data, it is still unable to assemble some transcripts especially at a higher percentage of reduction. We hypothesized that a possible reason for this behavior might be the ordering of reads in the input dataset. ORNA traverses the reads in the order in which it is generated from the sequencer. Hence, a high-quality read which is generated towards the end of a sequencing run might get discarded by ORNA. We implemented this idea in the form of ORNA-Q/K which is given as an extension to ORNA. In this approach, we score each read based on Phred scores of the bases in the read and the abundance of *k*-mers present in the read. During the read selection procedure, we give priority to reads having a high score. We aimed at maximizing the overall score of the normalized dataset.

Indeed, this modification enabled us to capture some additional transcripts which were missed earlier. But, at a higher percentage of reduction, the assembly quality was still lower than the assembly produced from the unreduced dataset. We realized that the assembly quality is highly dependent on the heuristics of the assembly algorithm. Although most of the *de*-Novo assemblers begin by constructing a *de*-Bruijn, they follow different approaches to generate assemblies from them. For instance, Trinity uses only a single *k*-mer size to generate the assembly whereas algorithms, such as Oases and TransABySS, generate multiple assemblies using different *k*-mer sizes and merge them to obtain the final assembly. We felt that generalizing the different heuristics followed by the assemblers and incorporating it into our normalization algorithm is beyond the scope of this thesis.

One of the major hindrance for any transcriptomic analysis pipeline is the length of the reads generated from the sequencers. The reads generated are not long enough to cover the entire transcript. Hence to cover all the bases in a trancsriptome, practitioners generally prefer to sequence deep. We feel that ORNA and ORNA-Q/K, when integrated into an assembler, can enable assembly algorithms to efficient assemble these high coverage data. This would pave the way for many scientific studies which would not have been possible earlier due to limited computational resources. In our work, we propose one such application where we assemble a combined RNA-seq dataset to detect biologically relevant long non-coding and protein coding RNAs. ORNA and ORNA-Q/K can also be applied on metagenomics and metatranscriptomic dataset. However, we have not tested this application as this is beyond the scope of this thesis.

6.2 Streamlined de novo transcriptomic assembly

During the course of our analysis, we realised that there are many confounders which make the RNA-seq analysis less straightforward. Such confounders included sequencing errors, redundancy in the read dataset, repetitive sequences and variants. We understood the importance of error correction and normalizing the reads before assembling them. These two steps drastically affect the quality of the final assembly produced. Unfortunately, not many practitioners perform pre-processing

of data and hence end up with sub-optimal assembly. We developed an automated pipeline named SOS which performs read error correction (using SEECER) and normalization (using ORNA) before assembling them using a de Bruijn graph based assembler. The assembler used in the pipeline were run with multiple *k*-mer values and the resulting assemblies were merged together to form a single non-redundant assembly. We used the *k*-mer selection algorithm KREATION to predict the *k*-mer parameter for the assembler. The pipeline is open source and is available in GitHub (https://github.com/SchulzLab/SOS). As mentioned before, an assembly algorithm is a heuristic and is highly data specific. A single assembler cannot guarantee a high quality assembly for multiple datasets. Hence, the user can replace the assembler by just changing a few lines of codes in the pipeline. The pipeline ends with estimating transcript level expression from the final assembly.

We tested the pipeline on three datasets of varying read length and coverage. We ran the pipeline using four different state-of-art assemblers. In all the cases, we found running the assembler using SOS resulted in better quality assembly as compared to the default settings of the assembler. In terms of the computational resource requirements, the memory and runtime required by SOS is larger than the assemblers in their default settings. This is due to the additional step of error correction and normalization. The resource requirement, precisely computational memory and runtime, also increases due the assemblers being executed using multiple *k*-mers. However, the improvement in the results far outweighs the additional resource requirements.

The SOS pipeline can be treated as a base structure and multiple other tools can be integrated into the pipeline. Algorithms such as DESeq and CuffDiff can be considered to perform differential expression analysis. Similarly, algorithms such as TrinityFusion and STAR-Fusion can be integrated to predict fusion events de-novo. Although, it must be noted that the results obtained from such tools might not agree with each other. Also, the parameters settings and the characteristics of the datasets used might highly influence the results especially for genes and transcripts which are expressed at low levels. Proper parameter selection of the algorithms based on data characteristics is still a black box which can be explored in the future.

6.3 Long read specific transcript quantification and gene fusion detection

As mentioned earlier, one of the major limitations of short-read sequencing is its inability to accurately reconstruct full length transcripts. This problem proves a major roadblock especially when studying complex species where a large percentage of genes undergo alternate splicing. Technologies such as Oxford Nanopore and Pacific Biosciences (PacBio), which were earlier used for genomic sequencing, are now providing transcriptomic long reads which cover the entire transcript.

We developed a tool, AERON, to quantify transcript expression using reads from Oxford Nanopore Technology (ONT). In the core, we generate a sequence graph from the reference transcriptome and align the reads against the graph. Based on the alignments, we estimate the expression of annotated transcripts. We also developed the first long read specific gene-fusion detection pipeline which is also based

on the alignment of reads against the sequence graph. The pipeline is open source and can be accessed via github (https://github.com/SchulzLab/AERON).

We tested the quantification pipeline of AERON on two ONT datasets of varying coverage and read-length and the compared the estimates against expression estimated obtained from the Minimap2 aligner. In both the cases, AERON is able to accurately predict the expression at gene-level. However at transcript level, the performance of AERON declines. This might be due to the high level of overlap between transcripts belonging to the same gene. If the reads originate from the overlapping regions, then it becomes difficult to assign them to the correct transcript. Also, the reads might not get aligned to the correct transcript due to the high error rate generated from the sequencing technologies. However, the transcript level expression estimates from AERON are still better than the expression estimates obtained from the Minimap2 alignment. ONT sequencing can be performed on the complementary DNA strand as well as directly on the RNA strand. The later procedure removes the dependence on the amplification process and hence the biases produced in the abundance of cDNA due to PCR amplification is reduced. We tested AERON on the datasets generated by both the procedures and found that the results correlate with each other at gene as well as at transcript level. Hence, AERON is independent of the procedure used for sequencing. Regarding the gene-fusion detection, we tested the pipeline on the K562 dataset and predicted some experimentally verified gene-fusion events. We also predicted some novel events which can be investigated further.

A major drawback of long-read technology, which also proved a hindrance in our work, is the high error rate produced by the sequencers. Practitioners sometimes augment the long-read analysis with short reads generated from the same sample to reduce the effect of sequencing errors. Updates in sequencing technologies are producing even longer reads which pass multiple times over a cDNA molecule which in turn reduces the sequencing errors. These two updates would eventually improve the accuracy of our quantification and gene-fusion detection. The sequence graph used by AERON covers all the possible alternative splicing events in the species. Hence in future, AERON can also be updated to assemble unannotated transcripts containing annotated exons.

6.4 Outlook

Updates in sequencing technologies are revealing certain unknown complexities of the transcriptome. Illuminas TrueSeq synthetic long-read technologies are one such interesting advance, where the library preparation is multiplexed and restricted to certain DNA molecules ([Til+15; Con+16]). These molecules are barcoded and pooled back and bulk sequenced. The barcoding of molecules makes the assembly procedure efficient and accurate. Earlier, we discussed the generation of long reads from technologies such as Oxford Nanopore and PacBio. These technologies have enabled amplification-free, single-molecule sequencing of cDNA to recover full-length transcripts without the necessity of the intermediate assembly step. However, certain isoform detection pipelines detect unknown isoforms which needs validation and classification. Scalability of long-read data analysis tools is another major challenge faced by the scientific community ([Ama+20]). Tools which results

in accurate analysis often take a lot of computational runtime which makes parameter optimization a cumbersome process. The scalability is also affected by the data generation and its storage. Oxford Nanopore sequencing has the fastest turnaround time. However, the results generated by the sequencers consume a lot of memory which increases the IT costs for large projects ([WJH19]).

Besides, Single-cell RNA-seq (scRNA-seq) is becoming one of the most active fields. Sequencers from companies such as 10x genomics can generate sequences from a small amount of starting material generally coming from a single cell. With the rapid progress in the field, more than 10,000 cells per sample can be sequenced and analyzed. It is estimated that within the next five to ten years, transcripts from more than 30 million to 100 million would be sequenced and analyzed ([HLD18]). This would pave the way to classify and identify novel cell types which would further expand our understanding of the heterogeneity of cell system. Currently, scRNA-seq data are mapped to a reference genome and analyzed. 10x genomics have their own in-house data analysis pipeline named CellRanger which aligns reads, generates feature-barcode matrix and performs gene-expression analysis ([Zhe+17]). Depending on the sequencing protocols, various other tools are also applied to scRNA-seq data. Tools such as TraCeR and BraCer and used to reconstruct TCR and BCR sequences respectively ([Stu+16; Lin+18]). Packages such as Seurat, SC3 and clusterExperiment are used for identification of highly variable genes, dimensionality reduction and clustering of scRNA data ([But+18; Ris+18; Kis+17]). Certain bulk RNA-seq analysis algorithms and pipelines, with tweaks, can also be used at different stages of the scRNA-seq analysis ([STM15]). Read mappers such as TopHat and STAR aligners are constantly applied for generating gene-count matrices. Packages such as FastQC and Kraken are used for quality control. Softwares like RSEM and DESeq2 are generally used for differential expression analysis. As the scRNA-seq analysis mainly depends on the reference sequence, there is little scope for new transcript discovery using the current methodologies. It is expected that new methods would be developed in the future opening a pandora box of information on cell physiology to systems biology.

Interesting times are ahead...

Bibliography

[AB12] Mariam Ayub and Hagan Bayley. "Individual RNA base recognition in immobilized oligonucleotides using a protein nanopore". In: *Nano Letters* 12.11 (2012), pp. 5637–5643. ISSN: 15306984. DOI: 10.1021/nl3027873.

[Alb+02] B. Alberts et al. *Molecular Biology of the Cell*. 4th. Garland, 2002.

[Alt+97] Stephen F. Altschul et al. *Gapped BLAST and PSI-BLAST: A new generation of protein database search programs*. 1997. DOI: 10.1093/nar/25.17.3389.

[Alv+15] Jose R. Alvarez et al. "DNA/RNA transverse current sequencing: Intrinsic structural noise from neighboring bases". eng. In: *Frontiers in Genetics* 6.JUN (2015), p. 213. ISSN: 16648021. DOI: 10.3389/fgene.2015.00213.

[Ama+20] Shanika L Amarasinghe et al. "Opportunities and challenges in long-read sequencing data analysis". In: *BMC genome biology* 21 (Feb. 2020), p. 30. DOI: https://doi.org/10.1186/s13059-020-1935-5.

[And+15] Simon Andrews et al. *FastQC. A quality control tool for high throughput sequence data. Babraham Bioinformatics*. 2015. URL: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/{\%}0Ahttp://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/.

[Ant+16] Dmitry Antipov et al. "HybridSPAdes: An algorithm for hybrid assembly of short and long reads". In: *Bioinformatics* 32.7 (2016), pp. 1009–1015. ISSN: 14602059. DOI: 10.1093/bioinformatics/btv688.

[Ard+18] Simon Ardui et al. *Single molecule real-time (SMRT) sequencing comes of age: Applications and utilities for medical diagnostics*. 2018. DOI: 10.1093/nar/gky066.

[Au+13] Kin Fai Au et al. "Characterization of the human ESC transcriptome by hybrid sequencing". In: *Proceedings of the National Academy of Sciences of the United States of America* 110.50 (2013). ISSN: 00278424. DOI: 10.1073/pnas.1320101110.

[Bia+13] Eva Bianconi et al. "An estimation of the number of cells in the human body". In: *Annals of Human Biology* (2013). ISSN: 03014460. DOI: 10.3109/03014460.2013.807878.

[Bla03] Douglas L. Black. "Mechanisms of Alternative Pre-Messenger RNA Splicing". In: *Annual Review of Biochemistry* 72.1 (2003), pp. 291–336. ISSN: 0066-4154. DOI: 10.1146/annurev.biochem.72.121801.161720.

[BLU14] Anthony M. Bolger, Marc Lohse, and Bjoern Usadel. "Trimmomatic: A flexible trimmer for Illumina sequence data". In: *Bioinformatics* 30.15 (2014), pp. 2114–2120. ISSN: 14602059. DOI: 10.1093/bioinformatics/btu170.

[BM+12] Nuno L. Barbosa-Morais et al. "The evolutionary landscape of alternative splicing in vertebrate species". In: *Science* 338.6114 (2012), pp. 1587–1593. ISSN: 10959203. DOI: 10.1126/science.1230612.

[BMS77] S. M. Berget, C. Moore, and P. A. Sharp. "Spliced segments at the 5′ terminus of adenovirus 2 late mRNA". In: *Proceedings of the National Academy of Sciences of the United States of America* 74.8 (1977), pp. 3171–3175. ISSN: 00278424. DOI: 10.1073/pnas.74.8.3171.

[BO16] Timo Beller and Enno Ohlebusch. "A representation of a compressed de Bruijn graph for pan-genome analysis that enables search". In: *Algorithms for Molecular Biology* 11.1 (2016). ISSN: 17487188. DOI: 10.1186/s13015-016-0083-7. arXiv: 1602.03333.

[Bra+16] Nicolas L. Bray et al. "Near-optimal probabilistic RNA-seq quantification". In: *Nature Biotechnology* 34.5 (2016), pp. 525–527. ISSN: 15461696. DOI: 10.1038/nbt.3519.

[Bro+12] C. Titus Brown et al. "A Reference-Free Algorithm for Computational Normalization of Shotgun Sequencing Data". In: *arXiv* 1203 (2012). arXiv: 1203.4802. URL: http://arxiv.org/abs/1203.4802.

[Bro97] Andrei Z. Broder. "On the resemblance and containment of documents". In: *Proceedings of the International Conference on Compression and Complexity of Sequences*. 1997. DOI: 10.1109/sequen.1997.666900.

[Bru46] N.G. Bruijn, de. "A combinatorial problem". English. In: *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam* 49.7 (1946), pp. 758–764. ISSN: 0370-0348.

[Bum13] Roger Bumgarner. "Overview of dna microarrays: Types, applications, and their future". In: *Current Protocols in Molecular Biology* SUPPL.101 (2013). ISSN: 19343639. DOI: 10.1002/0471142727.mb2201s101.

[Bus+16] Elena Bushmanova et al. "RnaQUAST: A quality assessment tool for de novo transcriptome assemblies". In: *Bioinformatics* 32.14 (2016), pp. 2210–2212. ISSN: 14602059. DOI: 10.1093/bioinformatics/btw218.

[But+18] Andrew Butler et al. "Integrating single-cell transcriptomic data across different conditions, technologies, and species". In: *Nature Biotechnology* 36.5 (2018), pp. 411–420. ISSN: 1546-1696. DOI: 10.1038/nbt.4096. URL: https://doi.org/10.1038/nbt.4096.

[Byr+17] Ashley Byrne et al. "Nanopore long-read RNAseq reveals widespread transcriptional variation among the surface receptors of individual B cells". In: *Nature Communications* 8.1 (2017), p. 16027. ISSN: 2041-1723. DOI: 10.1038/ncomms16027. URL: https://doi.org/10.1038/ncomms16027.

[Cab+17] Cédric Cabau et al. "Compacting and correcting Trinity and Oases RNA-Seq de novo assemblies". In: *PeerJ* 2017.2 (2017), e2988. ISSN: 21678359. DOI: 10.7717/peerj.2988. URL: https://doi.org/10.7717/peerj.2988.

[Cah+12] V. Cahais et al. "Reference-free transcriptome assembly in non-model animals from next-generation sequencing data". In: *Molecular Ecology Resources* 12.5 (2012), pp. 834–845. DOI: 10.1111/j.1755-0998.2012.03148.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1755-0998.2012.03148.x.

[CCS09] Chandra Chekuri, Kenneth L. Clarkson, and Har Peled Sariel. "On the set multi-cover problem in geometric settings". In: *Proceedings of the Annual Symposium on Computational Geometry*. 2009, pp. 341–350. ISBN: 9781605585017. DOI: 10.1145/1542362.1542421.

[Cha+15] Zheng Chang et al. "Bridger: A new framework for de novo transcriptome assembly using RNA-seq data". In: *Genome Biology* 16.1 (2015). ISSN: 1474760X. DOI: 10.1186/s13059-015-0596-2.

[Cha+18] Aniruddha Chatterjee et al. "A Guide for Designing and Analyzing RNA-Seq Data". In: *Gene Expression Analysis: Methods and Protocols*. Ed. by Nalini Raghavachari and Natàlia Garcia-Reyero. New York, NY: Springer New York, 2018, pp. 35–80. ISBN: 978-1-4939-7834-2. DOI: 10.1007/978-1-4939-7834-2_3. URL: https://doi.org/10.1007/978-1-4939-7834-2_3.

[Che+17] Shifu Chen et al. "AfterQC: Automatic filtering, trimming, error removing and quality control for fastq data". In: *BMC Bioinformatics* 18 (2017). ISSN: 14712105. DOI: 10.1186/s12859-017-1469-3.

[Chv79] V. Chvatal. "A Greedy Heuristic for the Set-Covering Problem". In: *Mathematics of Operations Research* 4.3 (1979), pp. 233–235. ISSN: 0364-765X. DOI: 10.1287/moor.4.3.233.

[CJ16] Nicolas Cerveau and Daniel J. Jackson. "Combining independent de novo assemblies optimizes the coding transcriptome for nonconventional model eukaryotic organisms". In: *BMC Bioinformatics* 17.1 (2016). ISSN: 14712105. DOI: 10.1186/s12859-016-1406-x.

[CLM16] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. "Compacting de Bruijn graphs from sequencing data quickly and in low memory". In: *Bioinformatics* 32.12 (2016), pp. i201–i208. ISSN: 14602059. DOI: 10.1093/bioinformatics/btw279.

[CM05] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: The count-min sketch and its applications". In: *Journal of Algorithms* 55.1 (2005), pp. 58–75. ISSN: 01966774. DOI: 10.1016/j.jalgor.2003.12.001.

[CM09] Mo Chen and James L. Manley. *Mechanisms of alternative splicing regulation: Insights from molecular and genomics approaches*. 2009. DOI: 10.1038/nrm2777.

[CM14] Rayan Chikhi and Paul Medvedev. "Informed and automated k-mer size selection for genome assembly". In: *Bioinformatics* 30.1 (2014), pp. 31–37. ISSN: 13674803. DOI: 10.1093/bioinformatics/btt310. arXiv: 1304.5665.

[Con+05] Ana Conesa et al. "Blast2GO: A universal tool for annotation, visualization and analysis in functional genomics research". In: *Bioinformatics* 21.18 (2005), pp. 3674–3676. ISSN: 13674803. DOI: 10.1093/bioinformatics/bti610.

[Con+16] Ana Conesa et al. *A survey of best practices for RNA-seq data analysis*. 2016. DOI: 10.1186/s13059-016-0881-8.

[CPT11] Phillip E.C. Compeau, Pavel A. Pevzner, and Glenn Tesler. "How to apply de Bruijn graphs to genome assembly". In: *Nature Biotechnology* 29.11 (2011), pp. 987–991. ISSN: 10870156. DOI: 10.1038/nbt.2023.

[CR13] Rayan Chikhi and Guillaume Rizk. "Space-efficient and exact de Bruijn graph representation based on a Bloom filter". In: *Algorithms for Molecular Biology* 8.1 (2013). ISSN: 17487188. DOI: 10.1186/1748-7188-8-22.

[CT12] Mark J. Chaisson and Glenn Tesler. "Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): Application and theory". In: *BMC Bioinformatics* 13.1 (2012). ISSN: 14712105. DOI: 10.1186/1471-2105-13-238.

[CT93] Rita S. Cha and William G. Thilly. "Specificity, efficiency, and fidelity of PCR". In: *Genome Research* 3.3 (1993), S18–29. ISSN: 10889051. DOI: 10.1101/gr.3.3.S18.

[Cun+15] Fiona Cunningham et al. "Ensembl 2015". In: *Nucleic Acids Research* 43.D1 (2015), pp. D662–D669. ISSN: 13624962. DOI: 10.1093/nar/gku1010.

[Dai+10] Manhong Dai et al. "NGSQC: Cross-platform quality analysis pipeline for deep sequencing data". In: *BMC Genomics* 11.SUPPL. 4 (2010). ISSN: 14712164. DOI: 10.1186/1471-2164-11-S4-S7.

[DB02] David W. Deamer and Daniel Branton. "Characterization of nucleic acids by nanopore analysis". In: *Accounts of Chemical Research* 35.10 (2002), pp. 817–825. ISSN: 00014842. DOI: 10.1021/ar000138m.

[Del02] A. L. Delcher. "Fast algorithms for large-scale genome alignment and comparison". In: *Nucleic Acids Research* 30.11 (2002), pp. 2478–2483. ISSN: 0305-1048. DOI: 10.1093/nar/30.11.2478.

[Den+18] Luca Denti et al. "ASGAL: Aligning RNA-Seq data to a splicing graph to detect novel alternative splicing events". In: *BMC Bioinformatics* 19.1 (2018). ISSN: 14712105. DOI: 10.1186/s12859-018-2436-3.

[Dep+19] Daniel P. Depledge et al. "Direct RNA sequencing on nanopore arrays redefines the transcriptional complexity of a viral pathogen". In: *Nature Communications* 10.1 (2019). ISSN: 20411723. DOI: 10.1038/s41467-019-08734-9.

[Des+16] Stéphane Deschamps et al. "Characterization, correction and de novo assembly of an Oxford Nanopore genomic dataset from Agrobacterium tumefaciens". In: *Scientific Reports* 6 (2016). ISSN: 20452322. DOI: 10.1038/srep28625.

[Dör+08] Andreas Döring et al. "SeqAn an efficient, generic C++ library for sequence analysis". In: *BMC Bioinformatics* 9 (2008). ISSN: 14712105. DOI: 10.1186/1471-2105-9-11.

[Dre+14] Erwan Drezen et al. "GATB: Genome Assembly & Analysis Tool Box". In: *Bioinformatics (Oxford, England)* 30.20 (2014), pp. 2959–2961. ISSN: 13674811. DOI: 10.1093/bioinformatics/btu406.

[DS16] Dilip A. Durai and Marcel H. Schulz. "Informed kmer selection for de novo transcriptome assembly". In: *Bioinformatics* 32.11 (2016), pp. 1670–1677. ISSN: 14602059. DOI: 10.1093/bioinformatics/btw217.

[DS18] Dilip A. Durai and Marcel H. Schulz. "In silico read normalization using set multi-cover optimization". In: *Bioinformatics* 34.19 (2018), pp. 3273–3280. ISSN: 14602059. DOI: 10.1093/bioinformatics/bty307.

[Eid+09] John Eid et al. "Real-time DNA sequencing from single polymerase molecules". In: *Science* 323.5910 (2009), pp. 133–138. ISSN: 00368075. DOI: 10.1126/science.1162986.

[Fit70] Walter M. Fitch. "Distinguishing homologous from analogous proteins". In: *Systematic Zoology* 19.2 (1970), pp. 99–113. ISSN: 00397989. DOI: 10.2307/2412448.

[FS81] Molly Fitzgerald and Thomas Shenk. "The sequence 5-AAUAAA-3 forms part of the recognition site for polyadenylation of late SV40 mRNAs". In: *Cell* 24.1 (1981), pp. 251–260. ISSN: 00928674. DOI: 10.1016/0092-8674(81)90521-3.

[Fu+12] Limin Fu et al. "CD-HIT: Accelerated for clustering the next-generation sequencing data". In: *Bioinformatics* 28.23 (2012), pp. 3150–3152. ISSN: 13674803. DOI: 10.1093/bioinformatics/bts565.

[Fu+18] Yu Fu et al. "Elimination of PCR duplicates in RNA-seq and small RNA-seq using unique molecular identifiers". In: *BMC Genomics* 19.1 (2018), p. 531. ISSN: 14712164. DOI: 10.1186/s12864-018-4933-1.

[FWA19] Shuhua Fu, Anqi Wang, and Kin Fai Au. "A comparative evaluation of hybrid error correction methods for error-prone long reads". In: *Genome Biology* 20.1 (2019). ISSN: 1474760X. DOI: 10.1186/s13059-018-1605-z.

[Gaf13] Daniel J. Gaffney. "Global Properties and Functional Complexity of Human Gene Regulatory Variation". In: *PLOS Genetics* 9.5 (May 2013), pp. 1–8. DOI: 10.1371/journal.pgen.1003501.

[Gar+18] Erik Garrison et al. *Variation graph toolkit improves read mapping by representing genetic variation in the reference*. 2018. DOI: 10.1038/nbt.4227.

[GHG14] A Gordon, G J Hannon, and Gordon. *FASTX-Toolkit*. 2014.

[Gil13] Donald G Gilbert. "Gene-omes built from mRNA-seq not genome DNA". In: *7th Annual Arthropod Genomics Symposium* (2013), p. 47405. DOI: 10.7490/f1000research.1112594.1.

[Gil78] Walter Gilbert. *Why genes in pieces?* 1978. DOI: 10.1038/271501a0.

[Gne+11] Sante Gnerre et al. "High-quality draft assemblies of mammalian genomes from massively parallel sequence data". In: *Proceedings of the National Academy of Sciences of the United States of America* 108.4 (2011), pp. 1513–1518. ISSN: 00278424. DOI: 10.1073/pnas.1017351108.

[Gor+15] Sean P. Gordon et al. "Widespread polycistronic transcripts in fungi revealed by single-molecule mRNA sequencing". In: *PLoS ONE* 10.7 (2015). ISSN: 19326203. DOI: 10.1371/journal.pone.0132628.

[Gra+11] Manfred G. Grabherr et al. "Full-length transcriptome assembly from RNA-Seq data without a reference genome". In: *Nature Biotechnology* 29.7 (2011), pp. 644–652. ISSN: 10870156. DOI: 10.1038/nbt.1883.

[Gra+13] Manfred G.; Grabherr et al. "Trinity: reconstructing a full-length transcriptome without a genome from RNA-Seq data". In: *Nature Biotechnology* 29.7 (2013), pp. 644–652. ISSN: 1546-1696. DOI: 10.1038/nbt.1883.Trinity. arXiv: 1512.00567.

[Gut+10] Mitchell Guttman et al. "Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs". In: *Nature Biotechnology* 28.5 (2010), pp. 503–510. ISSN: 10870156. DOI: 10.1038/nbt.1633.

[Har+00] Lodish Harvey et al. *Molecular Cell Biology. 4th edition*. 2000. DOI: 10.1016/j.jasms.2009.08.001.

[Har+12] Jennifer Harrow et al. "GENCODE: The reference human genome annotation for the ENCODE project". In: *Genome Research* 22.9 (2012), pp. 1760–1774. ISSN: 10889051. DOI: 10.1101/gr.135350.111.

[Heb+02a] Steffen Heber et al. "Splicing graphs and EST assembly problem". In: *Bioinformatics* 18 (2002), S181–S188. ISSN: 13674803. DOI: 10.1093/bioinformatics/18.suppl_1.S181.

[Heb+02b] Steffen Heber et al. "Splicing graphs and EST assembly problem". en. In: *Bioinformatics* 18 Suppl 1 (2002), S181–8.

[HLD18] Byungjin Hwang, Ji Hyun Lee, and Bang Duhee. "Single-cell RNA sequencing technologies and bioinformatics pipelines". In: *Experimental Molecular Medicine volume* (Aug. 2018). ISSN: 96. DOI: https://doi.org/10.1038/s12276-018-0071-8.

[HM19] Martin Hölzer and Manja Marz. "De novo transcriptome assembly: A comprehensive cross-species comparison of short-read RNA-Seq assemblers". In: *GigaScience* 8.5 (2019). ISSN: 2047217X. DOI: 10.1093/gigascience/giz039.

[IFI11] Lucian Ilie, Farideh Fazayeli, and Silvana Ilie. "HiTEC: Accurate error correction in high-throughput sequencing data". In: *Bioinformatics* 27.3 (2011), pp. 295–302. ISSN: 13674803. DOI: 10.1093/bioinformatics/btq653.

[JAB18] Lisa K Johnson, Harriet Alexander, and C Titus Brown. "Re-assembly, quality evaluation, and annotation of 678 microbial eukaryotic reference transcriptomes". In: *GigaScience* 8.4 (Dec. 2018). ISSN: 2047-217X.

[Jai+17a] Chirag Jain et al. "A Fast Approximate Algorithm for Mapping Long Reads to Large Reference Databases". In: *A Fast Approximate Algorithm for Mapping Long Reads to Large Reference Databases* (2017), p. 103812. DOI: 10.1101/103812.

[Jai+17b] M Jain et al. "Nanopore sequencing and assembly of a human genome with ultra-long reads". In: *bioRxiv* (2017). DOI: 10.1101/128835.

[Jia+13] Wenlong Jia et al. "SOAPfuse: An algorithm for identifying fusion transcripts from paired-end RNA-Seq data". In: *Genome Biology* 14.2 (2013). ISSN: 1474760X. DOI: 10.1186/gb-2013-14-2-r12.

[JKP13] Prachi Jain, Neeraja Krishnan, and Binay Panda. "Augmenting transcriptome assembly by combining de novo and genome-guided tools". In: *PeerJ* 1 (Aug. 2013), e133. DOI: 10.7717/peerj.133.

[Kar72] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[KB10] Sujai Kumar and Mark L. Blaxter. "Comparing de novo assemblers for 454 transcriptome data". In: *BMC Genomics* 11.1 (2010). ISSN: 14712164. DOI: 10.1186/1471-2164-11-571.

[Kee+14] Patrick J. Keeling et al. "The Marine Microbial Eukaryote Transcriptome Sequencing Project (MMETSP): Illuminating the Functional Diversity of Eukaryotic Life in the Oceans through Transcriptome Sequencing". In: *PLoS Biology* 12.6 (2014). ISSN: 15457885. DOI: 10.1371/journal.pbio.1001889.

[Ken02] W. J. Kent. "BLAT—The BLAST-Like Alignment Tool". In: *Genome Research* 12.4 (2002), pp. 656–664. ISSN: 1088-9051. DOI: 10.1101/gr.229202.

[Kil+13] Helena Kilpinen et al. "Coordinated effects of sequence variation on DNA binding, chromatin structure, and transcription". In: *Science* 342.6159 (2013), pp. 744–747. ISSN: 10959203. DOI: 10.1126/science.1242463.

[Kis+17] Vladimir Yu Kiselev et al. "SC3: consensus clustering of single-cell RNA-seq data". In: *Nature Methods* 14.5 (2017), pp. 483–486. ISSN: 1548-7105. DOI: 10.1038/nmeth.4236.

[KS11] Daehwan Kim and Steven L. Salzberg. "TopHat-Fusion: An algorithm for discovery of novel fusion transcripts". In: *Genome Biology* 12.8 (2011). ISSN: 14747596. DOI: 10.1186/gb-2011-12-8-r72.

[KSS10] David R. Kelley, Michael C. Schatz, and Steven L. Salzberg. "Quake: Quality-aware detection and correction of sequencing errors". In: *Genome Biology* 11.11 (2010). ISSN: 14747596. DOI: 10.1186/gb-2010-11-11-r116.

[KTT13] Tien Chueh Kuo, Tze Feng Tian, and Yufeng J. Tseng. "3Omics: A web-based systems biology tool for analysis, integration and visualization of human transcriptomic, proteomic and metabolomic data". In: *BMC Systems Biology* 7 (2013), p. 64. ISSN: 17520509. DOI: 10.1186/1752-0509-7-64.

[Kum+16] Shailesh Kumar et al. "Comparative assessment of methods for the fusion transcripts detection from RNA-Seq data". In: *Scientific Reports* 6 (2016). ISSN: 20452322. DOI: 10.1038/srep21597.

[Kur+03] Razelle Kurzrock et al. *Philadelphia Chromosome-Positive Leukemias: From Basic Mechanisms to Molecular Therapeutics*. 2003. DOI: 10.7326/0003-4819-138-10-200305200-00010.

[Lan+09] Ben Langmead et al. "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome". In: *Genome Biology* 10.3 (2009). ISSN: 14747596. DOI: 10.1186/gb-2009-10-3-r25.

[Lav+15] T. Laver et al. "Assessing the performance of the Oxford Nanopore Technologies MinION". In: *Biomolecular Detection and Quantification* 3 (2015), pp. 1–8. ISSN: 22147535. DOI: 10.1016/j.bdq.2015.02.001.

[LD11] Bo Li and Colin N. Dewey. "RSEM: Accurate transcript quantification from RNA-Seq data with or without a reference genome". In: *BMC Bioinformatics* 12 (2011). ISSN: 14712105. DOI: 10.1186/1471-2105-12-323.

[Le+13] Hai Son Le et al. "Probabilistic error correction for RNA sequencing". In: *Nucleic Acids Research* 41.10 (2013), e109. ISSN: 03051048. DOI: 10.1093/nar/gkt215.

[Lee03] Christopher Lee. "Generating consensus sequences from partial order multiple sequence alignment graphs". In: *Bioinformatics* 19.8 (2003), pp. 999–1008. ISSN: 13674803. DOI: 10.1093/bioinformatics/btg109.

[Lee+10] Seung-Tae Lee et al. "Multiplex ligation-dependent probe amplification screening of isolated increased HbF levels revealed three cases of novel rearrangements/deletions in the beta-globin gene cluster". en. In: *British Journal of Haematology* 148.1 (Jan. 2010), pp. 154–160.

[LFJ11] Wei Li, Jianxing Feng, and Tao Jiang. "IsoLasso: A LASSO regression approach to RNA-Seq based transcriptome assembly". In: *Journal of Computational Biology* 18.11 (2011), pp. 1693–1707. ISSN: 10665277. DOI: 10.1089/cmb.2011.0171.

[Li+12] Zhenyu Li et al. "Comparison of the two major classes of assembly algorithms: Overlap-layout-consensus and de-bruijn-graph". In: *Briefings in Functional Genomics* 11.1 (2012), pp. 25–37. ISSN: 20412649. DOI: 10.1093/bfgp/elr035.

[Li+14] Bo Li et al. "Evaluation of de novo transcriptome assemblies from RNA-Seq data". In: *Genome Biology* 15.12 (2014). ISSN: 1474760X. DOI: 10.1186/s13059-014-0553-5.

[Li16] Heng Li. "Minimap and miniasm: Fast mapping and de novo assembly for noisy long sequences". In: *Bioinformatics* 32.14 (2016), pp. 2103–2110. ISSN: 14602059. DOI: 10.1093/bioinformatics/btw152. arXiv: 1512.01801.

[Lim+19] Leandro Lima et al. "Comparative assessment of long-read error correction software applied to Nanopore RNA-sequencing data". In: *Briefings in Bioinformatics* (June 2019). bbz058. ISSN: 1477-4054. DOI: 10.1093/bib/bbz058. URL: https://doi.org/10.1093/bib/bbz058.

[Lin+18] Ida Lindeman et al. "BraCeR: B-cell-receptor reconstruction and clonality inference from single-cell RNA-seq". In: *Nature Methods* 15.8 (2018), pp. 563–565. ISSN: 1548-7105. DOI: 10.1038/s41592-018-0082-3. URL: https://doi.org/10.1038/s41592-018-0082-3.

[Liu+12] Binghang Liu et al. "COPE: An accurate k-mer-based pair-end reads connection tool to facilitate genome assembly". In: *Bioinformatics* 28.22 (2012), pp. 2870–2874. ISSN: 13674803. DOI: 10.1093/bioinformatics/bts563.

[Liu+16] Juntao Liu et al. "BinPacker: Packing-Based De Novo Transcriptome Assembly from RNA-seq Data". In: *PLoS Computational Biology* 12.2 (2016). ISSN: 15537358. DOI: 10.1371/journal.pcbi.1004772.

[Liu+19] Juntao Liu et al. "TransLiG: A de novo transcriptome assembler that uses line graph iteration". In: *Genome Biology* 20.1 (2019), p. 81. ISSN: 1474760X. DOI: 10.1186/s13059-019-1690-7.

[Luo+12] Ruibang Luo et al. *SOAPdenovo2: An empirically improved memory-efficient short-read de novo assembler*. 2012. DOI: 10.1186/2047-217X-1-18.

[LZS13] Bing Xin Lu, Zhen Bing Zeng, and Tie Liu Shi. "Comparative study of de novo assembly and genome-guided assembly strategies for transcriptome reconstruction based on RNA-Seq". In: *Science China Life Sciences* 56.2 (2013), pp. 143–155. ISSN: 16747305. DOI: 10.1007/s11427-013-4442-z.

[Mac15] Matthew D MacManes. *An opinionated guide to the proper care and feeding of your transcriptome*. Tech. rep. 2015, p. 035642. DOI: 10.1101/035642. URL: http://biorxiv.org/content/early/2015/12/30/035642.abstract.

[Mas+12] Andre P. Masella et al. "PANDAseq: Paired-end assembler for illumina sequences". In: *BMC Bioinformatics* 13.1 (2012). ISSN: 14712105. DOI: 10.1186/1471-2105-13-31.

[McC+14] Jamison M. McCorrison et al. "NeatFreq: Reference-free data reduction and coverage normalization for sequence assembly". In: *BMC Bioinformatics* 15.1 (2014). ISSN: 14712105. DOI: 10.1186/s12859-014-0357-3.

[MCS05] Arianne J. Matlin, Francis Clark, and Christopher W.J. Smith. *Understanding alternative splicing: Towards a cellular code*. 2005. DOI: 10.1038/nrm1645.

[ME13] Matthew D. MacManes and Michael B. Eisen. "Improving transcriptome assembly through error correction of high-throughput sequence reads". In: *PeerJ* 2013.1 (2013). ISSN: 21678359. DOI: 10.7717/peerj.113. arXiv: 1304.0817.

[Med+11] Paul Medvedev et al. "Error correction of high-throughput sequencing datasets with non-uniform coverage". In: *Bioinformatics* 27.13 (2011). ISSN: 13674803. DOI: 10.1093/bioinformatics/btr208.

[MK11] Guillaume Marçais and Carl Kingsford. "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers". In: *Bioinformatics* 27.6 (2011), pp. 764–770. ISSN: 13674803. DOI: 10.1093/bioinformatics/btr011.

[MKH19] Tuomo Mantere, Simone Kersten, and Alexander Hoischen. *Long-read sequencing emerging in medical genetics*. 2019. DOI: 10.3389/fgene.2019.00426.

[MQS93] MJ Moore, CC Query, and PA Sharp. "Splicing of precursors to mRNA by the spliceosome. In RNA World (ed. Gesteland RF, Atkins JF)". In: (1993), 303–357.

[MW11] Jeffrey A. Martin and Zhong Wang. *Next-generation transcriptome assembly*. 2011. DOI: 10.1038/nrg3068.

[Nar+14] Naoki Nariai et al. "TIGAR2: Sensitive and accurate estimation of transcript isoform expression with longer RNA-Seq reads". In: *BMC Genomics* 15 (2014). ISSN: 14712164. DOI: 10.1186/1471-2164-15-S10-S5.

[NKA13] Sergey I. Nikolenko, Anton I. Korobeynikov, and Max A. Alekseyev. "BayesHammer: Bayesian clustering for error correction in single-cell sequencing". In: *BMC Genomics* 14 (2013). ISSN: 14712164. DOI: 10.1186/1471-2164-14-S1-S7. arXiv: 1211.2756.

[NP13] Niranjan Nagarajan and Mihai Pop. *Sequence assembly demystified*. 2013. DOI: 10.1038/nrg3367.

[NW70] Saul B. Needleman and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of Molecular Biology* 48.3 (1970), pp. 443–453. ISSN: 00222836. DOI: 10.1016/0022-2836(70)90057-4.

[OE13] Shawn T. O'Neil and Scott J. Emrich. "Assessing De Novo transcriptome assembly metrics for consistency and utility". In: *BMC Genomics* 14.1 (2013). ISSN: 14712164. DOI: 10.1186/1471-2164-14-465.

[OWD13] Christopher R. O'Donnell, Hongyun Wang, and William B. Dunbar. "Error analysis of idealized nanopore sequencing". In: *Electrophoresis* 34.15 (2013), pp. 2137–2144. ISSN: 01730835. DOI: 10.1002/elps.201300174.

[Pat+17a] Benedict Paten et al. "Genome graphs and the evolution of genome inference". In: *Genome research* 27.5 (2017), pp. 665–676.

[Pat+17b] Rob Patro et al. "Salmon provides fast and bias-aware quantification of transcript expression". In: *Nature Methods* 14.4 (2017), pp. 417–419. ISSN: 15487105. DOI: 10.1038/nmeth.4197.

[Pel+12] Jason Pell et al. "Scaling metagenome sequence assembly with probabilistic de Bruijn graphs". In: *Proceedings of the National Academy of Sciences of the United States of America* 109.33 (2012), pp. 13272–13277. ISSN: 00278424. DOI: 10.1073/pnas.1121464109. arXiv: 1112.4193.

[Per+15] Mihaela Pertea et al. "StringTie enables improved reconstruction of a transcriptome from RNA-seq reads". In: *Nature Biotechnology* 33.3 (2015), pp. 290–295. ISSN: 15461696. DOI: 10.1038/nbt.3122.

[PMK14] Rob Patro, Stephen M. Mount, and Carl Kingsford. "Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms". In: *Nature Biotechnology* 32.5 (2014), pp. 462–464. ISSN: 15461696. DOI: 10.1038/nbt.2862. arXiv: 1308.3700.

[Pra+10] Tulika Prakash et al. "Expression of conjoined genes: Another mechanism for gene regulation in eukaryotes". In: *PLoS ONE* 5.10 (2010). ISSN: 19326203. DOI: 10.1371/journal.pone.0013284.

[PS+16] C. Palma-Silva et al. "De novo assembly and characterization of leaf and floral transcriptomes of the hybridizing bromeliad species (Pitcairnia spp.) adapted to Neotropical Inselbergs". In: *Molecular ecology resources* 16.4 (2016), pp. 1012–1022. ISSN: 17550998. DOI: 10.1111/1755-0998.12504.

[PTW01] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. "An Eulerian path approach to DNA fragment assembly". In: *Proceedings of the National Academy of Sciences of the United States of America* 98.17 (2001), pp. 9748–9753. ISSN: 00278424. DOI: 10.1073/pnas.171285098.

[Ris+18] Davide Risso et al. "clusterExperiment and RSEC: A Bioconductor package and framework for clustering of single-cell and other large gene expression datasets". In: *PLOS Computational Biology* 14.9 (Sept. 2018), pp. 1–16. DOI: 10.1371/journal.pcbi.1006378.

[RLC13] Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. "DSK: K-mer counting with very low memory usage". In: *Bioinformatics* 29.5 (2013), pp. 652–653. ISSN: 13674803. DOI: 10.1093/bioinformatics/btt020.

[RM19] Mikko Rautiainen and Tobias Marschall. "GraphAligner: Rapid and Versatile Sequence-to-Graph Alignment". In: *bioRxiv* (2019), p. 810812. DOI: 10.1101/810812. URL: https://www.biorxiv.org/content/10.1101/810812v1.

[RMM19] Mikko Rautiainen, Veli Mäkinen, and Tobias Marschall. "Bit-parallel sequence-to-graph alignment". In: *Bioinformatics* 35.19 (2019), pp. 3599–3607. ISSN: 14602059. DOI: 10.1093/bioinformatics/btz162.

[Rob+10] Gordon Robertson et al. "De novo assembly and analysis of RNA-seq data". In: *Nature Methods* 7.11 (2010), pp. 909–912. ISSN: 15487091. DOI: 10.1038/nmeth.1517.

[Rog+12] Mark F. Rogers et al. "SpliceGrapher: Detecting patterns of alternative splicing from RNA-Seq data in the context of gene models and EST data". In: *Genome Biology* 13.1 (2012). ISSN: 14747596. DOI: 10.1186/gb-2012-13-1-r4.

[RP13] Adam Roberts and Lior Pachter. "Streaming fragment assignment for real-time analysis of sequencing experiments". In: *Nature Methods* 10.1 (2013), pp. 71–73. ISSN: 15487091. DOI: 10.1038/nmeth.2251.

[Sal+93] Miguel Salazar et al. "The DNA Strand in DNA·RNA Hybrid Duplexes is Neither B-form nor A-form in Solution". In: *Biochemistry* 32.16 (1993), pp. 4207–4215. ISSN: 15204995. DOI: 10.1021/bi00067a007.

[SBJZ17] MH Schulz and Bar-Joseph-Z. "Algorithms for Next-Generation Sequencing data". In: Springer, 2017. Chap. Probabilistic models for error-correction of non-uniform sequencing data.

[SC54] Harold H Seward and Mr.Jay W Forrester Charles Wo Adams. "Information Sorting in the Application of Electronic Digital Computers To Business Operations". PhD thesis. 1954, p. 68.

[Sch+09] Jan Schröder et al. "SHREC: A short-read error correction method". In: *Bioinformatics* 25.17 (2009), pp. 2157–2163. ISSN: 13674803. DOI: 10.1093/bioinformatics/btp379.

[Sch+12a] Simon Schliesky et al. *RNA-seq assembly - Are we there yet?* 2012. DOI: 10.3389/fpls.2012.00220.

[Sch+12b] Marcel H. Schulz et al. "Oases: Robust de novo RNA-seq assembly across the dynamic range of expression levels". In: *Bioinformatics* 28.8 (2012), pp. 1086–1092. ISSN: 13674803. DOI: 10.1093/bioinformatics/bts094.

[Sch+16] V A Schneider et al. "Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly". In: *bioRxiv* (2016). DOI: 10.1101/072116. eprint: https://www.biorxiv.org/content/early/2016/08/30/072116.full.pdf. URL: https://www.biorxiv.org/content/early/2016/08/30/072116.

[Sen18] Supriya Sen. "Aberrant pre-mRNA splicing regulation in the development of hepatocellular carcinoma". In: *Hepatoma Research* 4.7 (2018), p. 37. ISSN: 2394-5079. DOI: 10.20517/2394-5079.2018.39.

[SF15] Li Song and Liliana Florea. "Rcorrector: Efficient and accurate error correction for Illumina RNA-seq reads". In: *GigaScience* 4.1 (2015). ISSN: 2047217X. DOI: 10.1186/s13742-015-0089-y.

[SFG08] Michael Sammeth, Sylvain Foissac, and Roderic Guigó. "A general definition and nomenclature for alternative splicing events". In: *PLoS Computational Biology* 4.8 (2008). ISSN: 1553734X. DOI: 10.1371/journal.pcbi.1000147.

[Sib+15] Christopher R. Sibley et al. "Recursive splicing in long vertebrate genes". In: *Nature* 521.7552 (2015), pp. 371–375. ISSN: 14764687. DOI: 10.1038/nature14466.

[Sim+09] Jared T. Simpson et al. "ABySS: A parallel assembler for short read sequence data". In: *Genome Research* 19.6 (2009), pp. 1117–1123. ISSN: 10889051. DOI: 10.1101/gr.089532.108.

[Sim+15] Felipe A. Simão et al. "BUSCO: Assessing genome assembly and annotation completeness with single-copy orthologs". In: *Bioinformatics* 31.19 (2015), pp. 3210–3212. ISSN: 14602059. DOI: 10.1093/bioinformatics/btv351.

[SK17] Mingfu Shao and Carl Kingsford. "Accurate Assembly of Transcripts Through Phase-Preserving Graph Decomposition". In: *Nature Biotechnology* 35.12 (2017), pp. 1167–1169. ISSN: 15461696. DOI: 10.1038/nbt.4020.

[SM19] Kristoffer Sahlin and Paul Medvedev. "De Novo Clustering of Long-Read Transcriptome Data Using a Greedy, Quality-Value Based Algorithm". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11467 LNBI. 2019, pp. 227–242. ISBN: 9783030170820. DOI: 10.1007/978-3-030-17083-7_14.

[Son+19] Charlotte Soneson et al. "A comprehensive examination of Nanopore native RNA sequencing for characterization of complex transcriptomes". In: *Nature Communications* 10.1 (2019). ISSN: 20411723. DOI: 10.1038/s41467-019-11272-z.

[Sov+16] Ivan Sović et al. "Evaluation of hybrid and non-hybrid methods for de novo assembly of nanopore reads". In: *Bioinformatics*. Vol. 32. 17. 2016, pp. 2582–2589. DOI: 10.1093/bioinformatics/btw237.

[SSK08] Karsten Suhre and Philippe Schmitt-Kopplin. "MassTRIX: mass translator into pathways". In: *Nucleic Acids Research* 36 (July 2008), W481–W484. DOI: 10.1093/nar/gkn194.

[STM15] Oliver Stegle, Sarah A. Teichmann, and John C. Marioni. "Computational and analytical challenges in single-cell transcriptomics". In: *Nature Reviews Genetics* 16.3 (2015), pp. 133–145. ISSN: 1471-0064. DOI: 10.1038/nrg3833.

[Sto+09] David Stoddart et al. "Single-nucleotide discrimination in immobilized DNA oligonucleotides with a biological nanopore". In: *Proceedings of the National Academy of Sciences of the United States of America* 106.19 (2009), pp. 7702–7707. ISSN: 00278424. DOI: 10.1073/pnas.0901054106.

[Str+02] Robert L. Strausberg et al. "Generation and initial analysis of more than 15,000 full-length human and mouse cDNA sequences". In: *Proceedings of the National Academy of Sciences of the United States of America* 99.26 (2002), pp. 16899–16903. ISSN: 00278424. DOI: 10.1073/pnas.242603899.

[Stu+16] Michael J. T. Stubbington et al. "T cell fate and clonality inference from single-cell transcriptomes". In: *Nature methods* 13.4 (2016). 26950746[pmid], pp. 329–332. ISSN: 1548-7105. DOI: 10.1038/nmeth.3800.

[SU+16] Richard Smith-Unna et al. "TransRate: Reference-free quality assessment of de novo transcriptome assemblies". In: *Genome Research* 26.8 (2016), pp. 1134–1144. ISSN: 15495469. DOI: 10.1101/gr.196469.115.

[SW81] T. F. Smith and M. S. Waterman. "Identification of common molecular subsequences". In: *Journal of Molecular Biology* (1981). ISSN: 00222836. DOI: 10.1016/0022-2836(81)90087-5.

[Sze+17] Sing Hoi Sze et al. "A scalable and memory-efficient algorithm for de novo transcriptome assembly of non-model organisms". In: *BMC Genomics* 18 (2017). ISSN: 14712164. DOI: 10.1186/s12864-017-3735-1.

[Tar+18] Manuel Tardaguila et al. "SQANTI: Extensive characterization of long-read transcript sequences for quality control in full-length transcriptome identification and quantification". In: *Genome Research* 28.3 (2018), pp. 396–411. ISSN: 15495469. DOI: 10.1101/gr.222976.117.

[Tib96] Robert Tibshirani. "Regression Shrinkage and Selection Via the Lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288. ISSN: 0035-9246. DOI: 10.1111/j.2517-6161.1996.tb02080.x.

[Til+15] hagen Tilgner et al. "Comprehensive transcriptome analysis using synthetic long-read sequencing reveals molecular co-association of distant splicing events". In: *Nature Biotechnology* 33 (May 2015), 736–742. DOI: https://doi.org/10.1038/nbt.3242.

[TPS09] Cole Trapnell, Lior Pachter, and Steven L. Salzberg. "TopHat: Discovering splice junctions with RNA-Seq". In: *Bioinformatics* 25.9 (2009), pp. 1105–1111. ISSN: 13674803. DOI: 10.1093/bioinformatics/btp120.

[Tra+10] Cole Trapnell et al. "Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation". In: *Nature Biotechnology* 28.5 (2010), pp. 511–515. ISSN: 10870156. DOI: 10.1038/nbt.1621.

[Tur+18] Isaac Turner et al. "Integrating long-range connectivity information into de Bruijn graphs". In: *Bioinformatics* 34.15 (2018), pp. 2556–2565. ISSN: 14602059. DOI: 10.1093/bioinformatics/bty157.

[Ung+17] Arnaud Ungaro et al. "Challenges and advances for transcriptome assembly in non-model species". In: *PloS one* 12.9 (2017). 28931057[pmid], e0185020–e0185020. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0185020.

[Var10] Kasturi Varadarajan. "Weighted geometric set cover via quasi-uniform sampling". In: *Proceedings of the Annual ACM Symposium on Theory of Computing*. 2010, pp. 641–647. ISBN: 9781605588179. DOI: 10.1145/1806689.1806777.

[Ven+18] Luca Venturini et al. "Leveraging multiple transcriptome assembly methods for improved gene structure annotation". In: *GigaScience* 7.8 (2018). ISSN: 2047217X. DOI: 10.1093/gigascience/giy093.

[Wan+10] Kai Wang et al. "MapSplice: Accurate mapping of RNA-seq reads for splice junction discovery". In: *Nucleic Acids Research* 38.18 (2010). ISSN: 03051048. DOI: 10.1093/nar/gkq622.

[Wan+13] Su Wang et al. "Target analysis by integration of transcriptome and ChIP-seq data with BETA". In: *Nature protocols* 8 (Nov. 2013), 2502–2515. DOI: https://doi.org/10.1038/nprot.2013.150.

[Wan+19] Zhong Wang et al. "Identification of regulatory elements from nascent transcription using dREG". In: *Genome Research* 29.2 (2019), pp. 293–303. ISSN: 15495469. DOI: 10.1101/gr.238279.118.

[Wat+13] Robert M. Waterhouse et al. "OrthoDB: A hierarchical catalog of animal, fungal and bacterial orthologs". In: *Nucleic Acids Research* 41.D1 (2013). ISSN: 03051048. DOI: 10.1093/nar/gks1116.

[WC53] J. D. Watson and F. H.C. Crick. "Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid". In: *Nature* 171.4356 (1953), pp. 737–738. ISSN: 00280836. DOI: 10.1038/171737a0.

[Wed+17] Axel Wedemeyer et al. "An improved filtering algorithm for big read datasets and its application to single-cell assembly". In: *BMC Bioinformatics* (2017). ISSN: 14712105. DOI: 10.1186/s12859-017-1724-7.

[WGS09] Zhong Wang, Mark Gerstein, and Michael Snyder. *RNA-Seq: A revolutionary tool for transcriptomics*. 2009. DOI: 10.1038/nrg2484.

[Wic+17] Ryan R. Wick et al. "Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads". In: *PLoS Computational Biology* 13.6 (2017). ISSN: 15537358. DOI: 10.1371/journal.pcbi.1005595.

[WJH19] Ryan Wick, Louise Judd, and Kathryn E. Holt. "Performance of neural network basecalling tools for Oxford Nanopore sequencing". In: *BMC genome biology* 20 (June 2019), p. 129. DOI: https://doi.org/10.1186/s13059-019-1727-y.

[Wym+19] Dana Wyman et al. "A technology-agnostic long-read analysis pipeline for transcriptome discovery and quantification". In: *bioRxiv* (2019), p. 672931. DOI: 10.1101/672931. URL: https://www.biorxiv.org/content/10.1101/672931v1.

[Xie+14] Yinlong Xie et al. "SOAPdenovo-Trans: De novo transcriptome assembly with short RNA-Seq reads". In: *Bioinformatics* 30.12 (2014), pp. 1660–1666. ISSN: 14602059. DOI: 10.1093/bioinformatics/btu077.

[Yan+17] Chen Yang et al. *NanoSim: Nanopore sequence read simulator based on statistical characterization*. 2017. DOI: 10.1093/gigascience/gix010.

[Yor+16] Deniz Yorukoglu et al. *Compressive mapping for next-generation sequencing*. 2016. DOI: 10.1038/nbt.3511.

[ZB08] Daniel R. Zerbino and Ewan Birney. "Velvet: Algorithms for de novo short read assembly using de Bruijn graphs". In: *Genome Research* 18.5 (2008), pp. 821–829. ISSN: 10889051. DOI: 10.1101/gr.074492.107.

[Zer+18] Daniel R. Zerbino et al. "Ensembl 2018". In: *Nucleic Acids Research* 46.D1 (2018), pp. D754–D761. ISSN: 13624962. DOI: 10.1093/nar/gkx1098.

[Zha+11] Qiong Yi Zhao et al. "Optimizing de novo transcriptome assembly from short-read RNA-Seq data: a comparative study." In: *BMC bioinformatics* (2011). ISSN: 14712105. DOI: 10.1186/1471-2105-12-S14-S2.

[Zha+14] Shanrong Zhao et al. "Comparison of RNA-Seq and microarray in transcriptome profiling of activated T cells". In: *PLoS ONE* 9.1 (2014), e78644. ISSN: 19326203. DOI: 10.1371/journal.pone.0078644.

[Zha+15] Shanrong Zhao et al. "Comparison of stranded and non-stranded RNA-seq transcriptome profiling and investigation of gene overlap". In: *BMC Genomics* 16.1 (2015), p. 675. ISSN: 14712164. DOI: 10.1186/s12864-015-1876-7.

[Zha+17] Chi Zhang et al. "Evaluation and comparison of computational tools for RNA-seq isoform quantification". In: *BMC Genomics* 18.1 (2017). ISSN: 14712164. DOI: 10.1186/s12864-017-4002-1.

[Zhe+17] Grace X. Y. Zheng et al. "Massively parallel digital transcriptional profiling of single cells". In: *Nature Communications* 8.1 (2017), p. 14049. ISSN: 2041-1723.

[Zim+13] Aleksey V. Zimin et al. "The MaSuRCA genome assembler". In: *Bioinformatics* 29.21 (2013), pp. 2669–2677. ISSN: 13674803. DOI: 10.1093/bioinformatics/btt476.

[ZW14] Zhaojun Zhang and Wei Wang. "RNA-Skim: A rapid method for RNA-Seq quantification at transcript level". In: *Bioinformatics* 30.12 (2014). ISSN: 14602059. DOI: 10.1093/bioinformatics/btu288.