De-novo Assembly and Annotation of mitochondrial genome of Mulberry (*Morus indica* L.) using NGS data

A Major Project dissertation submitted

in partial fulfilment of the requirement for the degree of

Master of Technology

In

Bioinformatics

Submitted by

Harrisham Kaur

(2K11/BIO/06)

Delhi Technological University, Delhi, India

Under the supervision of

Dr. Asmita Das

Department of Biotechnology
Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road,
Delhi-110042, INDIA

CERTIFICATE

This is to certify that the M. Tech. dissertation entitled **"*De-novo* Assembly and Annotation of Mitochondrial Genome of Mulberry (*Morus indica* L.) using NGS data."**, submitted by **HARRISHAM KAUR (2K11/BIO/06)** in partial fulfilment of the requirement for the award of the degree of Master of Engineering, Delhi Technological University (Formerly Delhi College of Engineering, University of Delhi), is an authentic record of the candidate's own work carried out by him/her under my guidance.

The information and data enclosed in this dissertation is original and has not been submitted elsewhere for honouring of any other degree.

Date: June 28, 2013

Dr. Asmita Das
Guide name
(Project Mentor)
Department of Bio-Technology
Delhi Technological University
(Formerly Delhi College of Engineering, University of Delhi)

DECLARATION

The work presented in this dissertation entitled "***De-novo* Assembly and Annotation of mitochondrial genome of Mulberry (*Morus indica* L.) using NGS data**" is original and has been carried out by me under the supervision of **Dr. Asmita Das**, Assistant Professor, Department of Biotechnology, Delhi Technological University, New Delhi and of **Dr. Ramesh K Aggarwal,** Chief Scientist, Centre for Cellular and Molecular Biology, Hyderabad.

I declare that the matter embodied in this thesis has not been submitted by me in any part for award of any degree/diploma of any other institution or university previously.

Place: New Delhi

Date:

Harrisham Kaur

(2K11/BIO/06)

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely fortunate to have got all this along the completion of my project work. Whatever I have done is only due to such guidance and assistance and I would not forget to thank them.

First, of all, I am extremely thankful to **Prof P.B Sharma**, Vice-Chancellor, Delhi Technological University, for providing me an opportunity to study in one of the premier institutes of the country. My warm thanks to **Dr. S Maji**, Head of Department of Biotechnology, DTU for his never ending support and constant encouragement.

I take this opportunity to express my profound gratitude and deep regards to my supervisor **Dr. Asmita Das**, Assistant Professor, DTU, for her exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The motivation, help and guidance given by her time through time shall carry me a long way in the journey of life on which I am about to embark.

I also take this opportunity to express a deep sense of gratitude to my external guide, **Dr. Ramesh K Aggarwal**, Chief Scientist, Center for Cellular and Molecular Biology, Hyderabad, for his cordial support, valuable information and guidance, which helped me in completing this task through various stages.

I am obliged to the faculty and staff members of CCMB, Hyderabad for the valuable information provided by them in their respective fields. I am grateful to **Dr. Shrish Tiwari**, Sr. Scientist, CCMB and **Janani Ma’am** for correcting me at every stage of the project.

Lastly, I thank God, my parents, sisters and friends for their constant encouragement without which this assignment would not be possible.

List of Figures

<u>**List of Tables**</u>

List of Abbreviations

1. mtDNA : Mitochondrial DNA.
2. cpDNA : Chloroplast DNA.
3. ncDNA : Nuclear DNA.
4. Mit-Genome : Mitochondrial DNA.
5. NGS : Next Generation Sequencing.
6. CCD Camera: Charged-Coupled Device Camera.
7. SFF : Standard Flow-gram Format File.
8. SNP : Single Nucleotide Polymorphism.
9. BGI : Beijing Genomics Institute
10. BLAST : Basic Local Alignment Search Tool.
11. SD : Standard Deviation.
12. ORF : Open Reading Frame
13. QC : Quality Control
14. MID : MultiPlex Identifier
15. NCBI : National Centre for Biotechnology Information.
16. GSS : Genome Survey Sequence.
17. GFF : General Feature File.

TABLE OF CONTENTS

CHAPTER #1

ABSTRACT

De-novo Assembly and Annotation of Mitochondrial Genome of Mulberry (*Morus indica* L.) using NGS data.

Harrisham Kaur

Delhi Technological University, Delhi, India

ABSTRACT

Morus, a genus of flowering plants in the family Moraceae, comprises 10–16 species of deciduous trees commonly known as mulberries growing wild and under cultivation in many temperate world regions. Mulberry is a very widespread and important crop for silkworm feed, fruit and timber as well as being an excellent amenity tree. *Morus indica* is a species of mulberry exclusively found in Eastern and Southern Asia and is of great importance to the Asian silk industry. Its complete sequence of the mitochondrial (mt) genome could provide clues for the understanding of the evolution of mitochondrial genomes in plants. In this study we have attempted to assemble and annotate the mitochondrial genome of *Morus indica* L using Roche derived 454 Next Generation Sequencing data. The *Morus indica* mt-genome was sequenced from total genomic DNA without physical separation of chloroplast and nuclear DNA. Various Bioinformatics tools and in-house developed perl and shell scripts were used to assemble and annotate the quality-filtered 454 raw NGS reads. We report the first ever, near complete mt-genome of mulberry in terms of gene-content in closely related species with 27 high-quality, high read-coverage contigs comprising of 45 functional protein coding mt-genes, 2 rRNA genes and 25 tRNAs (transfer RNAs) that recognize 14 different amino-acids. The average coverage of reported mulberry mt-genome is 66x and the estimated mt-genome size is 380,529 nt. A 454 bp segment from plastid origin is incorporated in the 380,529 nt of mulberry mit-genome. We also report a procedure for efficient assembly and annotation of mitochondrial genomes of plants without physical separation of mitochondria. This procedure can be extended to other platforms with low coverage genome sequencing, such as the Illumina HiSeq platform for efficient and straight-forward organellar genome sequencing. The draft Mulberry Mit-genome assembled by our procedure could be an essential resource to biologists, geneticists, plant scientists, and plant breeders and can be used as a reference to assemble mt-genomes of closely related species.

CHAPTER #2

INTRODUCTION

Usually, a plant cell contains three genomes: plastid, mitochondrial, and nuclear. In a typical *Arabidopsis* leaf cell, there are about 100 copies of mitochondrial DNA (mtDNA), about 1,000 copies of chloroplast DNA (cpDNA), and two copies of nuclear DNA (ncDNA) (DC, 2006).

The mitochondrial genome plays fundamental roles in development and metabolism as the major ATP production centre via oxidative phosphorylation (Mackenzie S *et al*,1999). The mitochondrial genetic system in flowering plants exhibit multiple characteristics that distinguish them from other eukaryotes: large genome size with dispersed genes, an incomplete set of tRNAs, trans-splicing, and frequent uptake of plastid DNA or of foreign DNA fragments by horizontal and intracellular gene transfer (Mackenzie S *et al*, 1999), (Keeling PJ *et al*, 2008), (Sloan DB *et al*, 2010), (Alverson AJ *et al*, 2010). Plant mtDNAs are a major resource for evolutionary studies, because coding regions evolve slowly, in contrast to the flexible non-coding DNA. Therefore, the structural evolution and plasticity of plant mtDNAs make them powerful model for exploring the forces that affect their divergence and recombination.

With the emergence of next-generation sequencing technologies, the number of completed plant mitochondrial genomes submitted to GenBank are 78. These are accessible through the URL(http://www.ncbi.nlm.nih.gov/genomes/GenomesGroup.cgi?taxid=33090&opt=organelle. Accessed 2013 Feb 11). Most are from Chlorophyta (17 of green algae) and seed plants (26 of eudicotyledons) (Wang *et al*,2012). These Next-Generation Sequencing technologies have demonstrated the capacity to sequence DNA at unprecedented speed, thereby enabling previously unimaginable scientific achievements and novel biological applications. But, the massive data produced by NGS also presents a significant challenge for data storage, analyses, and management solutions. Therefore advanced bioinformatics tools and careful scrutiny of the raw data are essential for the successful application of NGS technology (Jun Zhang *et al*, 2011).

Morus, a genus of flowering plants in the family Moraceae, comprises 10–16 species of deciduous trees commonly known as mulberries growing wild and under cultivation in many temperate world regions. Mulberry is a very widespread and important crop for silkworm feed, fruit and timber as well as being an excellent amenity tree. *Morus indica* is a species of mulberry exclusively found in Eastern and Southern Asia and is of great importance to the Asian silk industry. Its complete sequence of the mitochondrial (mt) genome could provide clues for the understanding of the evolution of mt genomes in plant. This study aims at using Roche derived 454 Pyro-sequencing data to assemble and annotate *Morus indica* L. mitochondrial (mt) genome. This is the first ever, *De novo* mt-genome assembly of mulberry. We used the raw NGS reads of two parents of a mapping population of mulberry derived from a single plate run of 454 Pyrosequencing run to assemble a mt-genome of mulberry. The mitochondria of flowering plants is mostly conserved across species, so the pooling of sequencing reads of two parents provides a confidence in depth of read coverage in regions conserved in both the parents and fills in the information missing in individual parents, thus providing a confident and more informative assembly.

With the use of commercially and publically available tools and some in-house developed perl and shell scripts we present a near complete mt-genome of mulberry *Morus indica* L. in terms of gene-content in closely related species. We report 27 high-quality contigs with an average coverage of 66x, comprising of 41 functional protein coding mt-genes, 3 RNA genes and 25 tRNAs (transfer RNAs) that recognize 14 different amino-acids. This draft genome can be an essential resource to biologists, geneticists, plant scientists, and plant breeders and can be used as a reference to assemble mt-genomes of closely related species.

CHAPTER #3

REVIEW OF LITERATURE

3.1 *Morus*

Morus, a genus of flowering plants in the family Moraceae, comprises 10–16 species of deciduous trees commonly known as mulberries growing wild and under cultivation in many temperate world regions (JM *et al*, 2012). The closely related genus *Broussonetia* is also commonly known as mulberry, notably the Paper Mulberry, *Broussonetia papyrifera*. Mulberries are swift-growing when young, but soon become slow-growing and rarely exceed 10–15 m (33–49 ft) tall. The leaves are alternately arranged, simple, often lobed, more often lobed on juvenile shoots than on mature trees, and serrated on the margin. Depending on the species, they can be monoecious or dioecious (JM *et al*, 2012). The mulberry fruit is a multiple fruit, 2–3 cm (0.79–1.2 in) long. Immature fruits are white, green, or pale yellow. In most species, the fruits turn pink then red while ripening, then dark purple or black and have a sweet flavor when fully ripe. The fruits of the white-fruited cultivar are white when ripe; the fruit in this cultivar is also sweet but has a very mild flavor compared with the darker variety.

3.1.1 General Description

Mulberry is a fast growing deciduous woody perennial plant. It has a deep-root system. The leaves are simple, alternate, stipulate, petiolate, entire or lobed. Number of lobes varies from 1 to 5. Plants are generally dioecious. Inflorescence is catkin with pendent or drooping peduncle bearing unisexual flowers. Inflorescence is always auxiliary. Male catkins are usually longer than the female catkins. Male flowers are loosely arranged and after shedding the pollen, the inflorescence dries and falls off. Number of parianth lobes are 4. Number of stamens are 4 and implexed in bud. Female inflorescence is usually short and the flowers are very compactly arranged. Number of parianth lobes are 4 and persistent. Ovary is one-celled and stigma is bifid. The chief pollinating agent in mulberry is wind. Fruit is a sorosis and the colour of the fruit is mainly violet black.

Most of the species of the genus Morus and cultivated varieties are diploid having 28 chromosomes. However, triploids (2n=(3x)=42) are also extensively cultivated for their adaptability, vigorous growth and quality of leaves (Datta, 2012).

3.1.2 Uses of Mulberry

a) Silk Industry: Mulberry leaves, particularly those of the white mulberry, are ecologically important as the sole food source of the silkworm (*Bombyx mori*, named after the mulberry genus *Morus*), the pupa/cocoon of which is used to make silk (Ombrello, 2012) (Mulberry Silk, 2012). Other Lepidoptera larvae also sometimes feed on the plant including common emerald, lime hawk-moth, and sycamore moth.

a) Anthocyanins from mulberry fruit: Anthocyanins are pigments which hold potential use as dietary modulators of mechanisms for various diseases (DX *et al*, 2003) and as natural food colorants. Due to increasing demand for natural food colorants, their significance in the food industry is increasing. Anthocyanins are responsible for the attractive colors of fresh plant foods, producing colors such as orange, red, purple, black, and blue. They are water-soluble and easily extractable. A cheap and industrially feasible method to purify anthocyanins from mulberry fruit which could be used as a fabric tanning agent or food colorant of high color value (of above 100) has been established. Scientists found that out of 31 Chinese mulberry cultivars tested, the total anthocyanin yield varied from 148 mg to 2725 mg per liter of fruit juice (Liu *et al*, 2004). Total sugars, total acids, and vitamins remained intact in the residual juice after removal of anthocyanins and that the residual juice could be fermented to produce products such as juice, wine, and sauce. Anthocyanin content depends on climate, area of cultivation, and is particularly higher in sunny climates (Matus *et al*, 2009).This finding holds promise for tropical sericulture countries to profit from industrial anthocyanin production from mulberry through anthocyanin recovery.

c) Mulberry is non-toxic natural therapeutic agent shown to possess hypoglycemic, hypotensive, and diuretic properties (Bondada *et al*, 2001).

3.1.3 Species and varieties under cultivation in India

There are about 68 species of the genus *Morus*, the majority of them occur in Asia, especially in China (24 species) and Japan (19). Continental America is also rich in its *Morus* species. The genus is poorly represented in Africa, Europe and Middle East, and it is not present in Australia.

In India, there are many species of *Morus*, of which *Morus alba*, *M. indica*. M. *serrata* and *M. laevigata* grow wild in the Himalayas. Several varieties have been introduced belonging to *M. multicaulis*, *M. nigra*,*M. sinensis* and M. *phillippinensis*. Most of the Indian varieties of mulberry belong to *M. indica* (Datta, 2012).

Though mulberry cultivation is practiced in various climates, the major area is in tropical zone covering Karnataka, Andhra Pradesh and Tamil Nadu states, with about 90%. In the sub-tropical zone, West Bengal, Himachal Pradesh and north-eastern states have major areas under mulberry cultivation.

Figure #1 ***Morus indica*** **L. (adopted from www.crfg.org)**

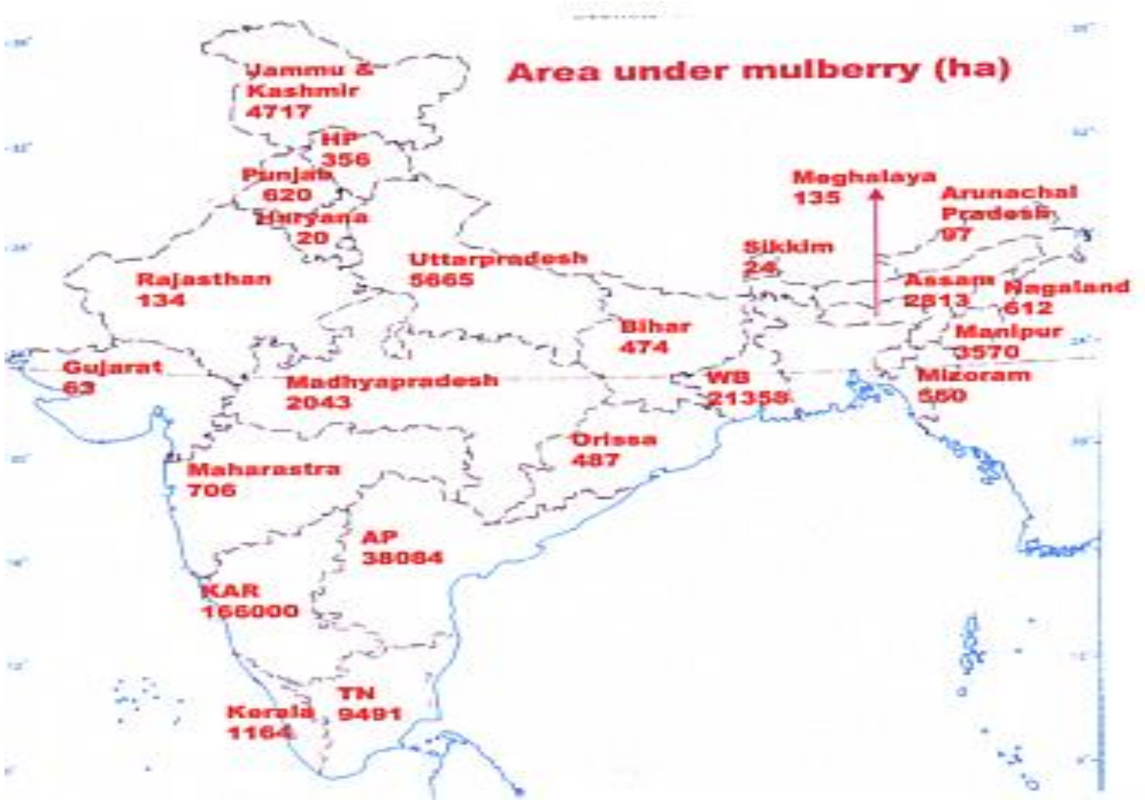


Figure #2 Area under mulberry cultivation in different states **(Datta, 2012)**

3.2 Sequencing and assembling mitochondrial (mt) -genome of *Morus indica* L.

The complete mt-genome of mulberry has not been reported, so sequencing and assembling the mulberry mt-genome will provide a great leap to the plant genomic resources. Besides that plant mitochondrial genomes, encoding necessary proteins are involved in the system of energy production, and play an important role in the development and reproduction of the plant. They occupy a specific evolutionary pattern relative to their nuclear counterparts (Cui *et al*, 2009). Hence the assembly of mitochondria can be imperative in unravelling the evolutionary mechanism manifesting themselves in plant families. The mt-genome can be an essential resource to biologists, geneticists, plant scientists, and plant breeders and can be used as a reference to assemble mt-genomes of closely related species.

3.3 Complexity of plant mt-genomes.

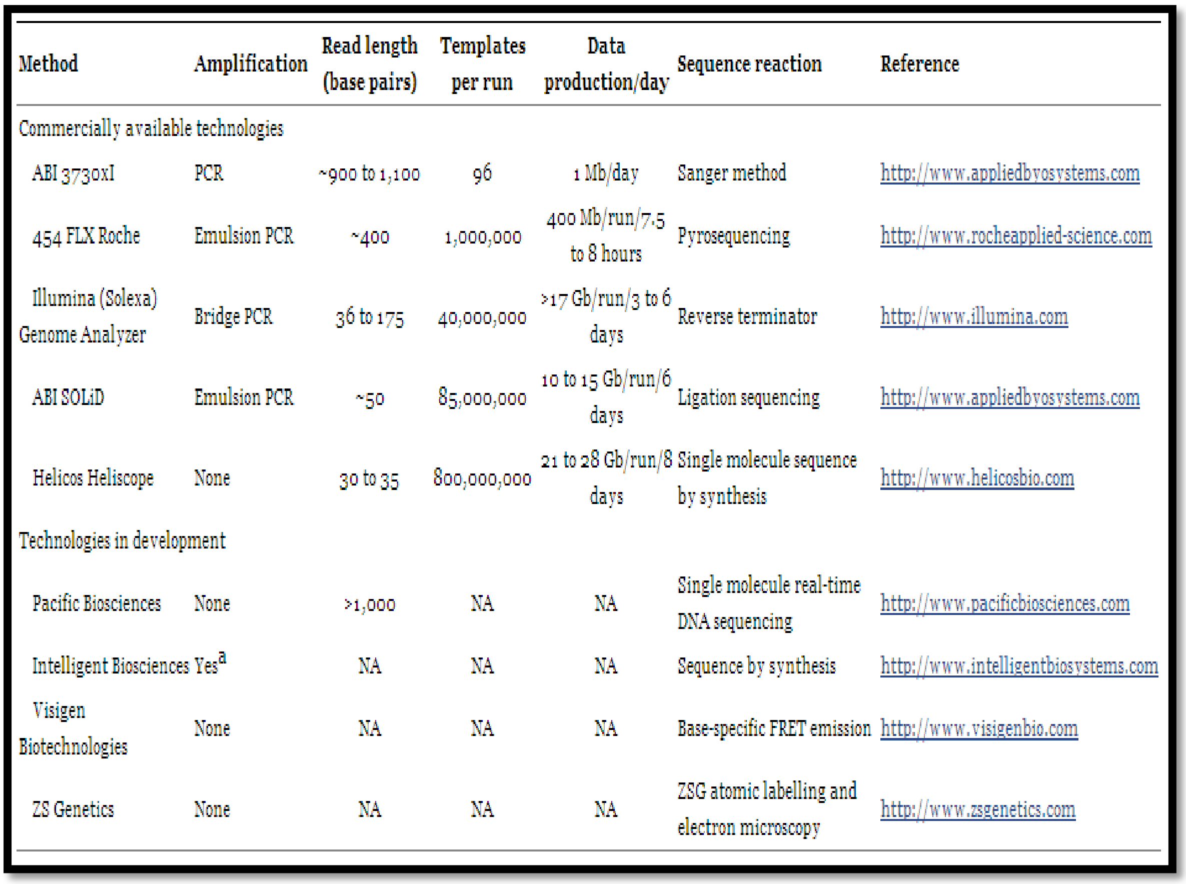
Plant mitochondrial genomes are complex because they encode significantly more genes than do their fungal and animal counterparts. Investigations of the mitochondrial genome sequences of at least 13 angiosperm species, including *Arabidopsis thaliana*, *Beta vulgaris*, *Oryza sativa*, *Brassica napus*, *Zea mays* , *Nicotiana tabacum*, *Triticum aestivum*, *Vitis vinifera*, *Citrullus lanatus* and *Cucurbita pepo*, and *Vigna radiata*, together with physical mapping , have showed several properties of plant mitochondrial genomes, such as large size (200-2400 kb), slow rates of evolutionary change, incorporation of foreign DNA, a multipartite structure, and specific modes of gene expression (e.g. *cis* and *trans* splicing, RNA editing), etc (Schuster *et al*, 1994).

To date, 78 mitochondrial genomes in plants have been fully sequenced and analysed http://www.ncbi.nlm.nih.gov/Genomes/. These mitochondrial genomes are extremely variable in size, ranging from 221 kb (*Brassica napus*) to 2,740 kb (*Cucumis melo*). Sequence analysis revealed that the most abundant portion of the mitochondrial genomes is non-coding (Kubo *et al*, 2008), which includes "promiscuous" DNA of plastid and nuclear origin (Kubo *et al*, 2007), as well as sequences of horizontal origin from foreign genomes (Richardson *et al*, 2007) (Archibald *et al*, 2010). Structural analysis, through use of Southern hybridization or paired-end data, revealed a high frequency of intra- and intermolecular recombination due to accumulation of repetitive sequences. This process has generated a structurally dynamic assemblage of genome configurations within a species (Ogihara *et al*, 2005) (Chang *et al*, 2011) and a scrambling of gene order within closely related species (Alverson AJ W. X., 2010). This dynamic organization of the plant mitochondrial genome provides a powerful model for the study of genome structure and evolution. In addition, the increasing availability of plant organelle and nuclear genome sequence data provides an understanding of the mechanisms driving plant genome evolution. Indeed, there is a strong structural and functional interaction among plastid, mitochondrial, and nuclear genomes (Woodson *et al*, 2008). Transfer of DNA among these three compartments in higher plants has been reported, with exception of transfer into the plastid genome (Kleine T *et al*, 2009).

3.4 Next-Generation Sequencing Technology and its advent on mt-genome assembly and annotation:

Despite the importance of mt-genome assembly and annotation, the technical obstacles of DNA isolation and sequence assembly limit the sequencing of mitochondrial genomes. Conventional approaches to mitochondrial genome sequencing involve extraction and enrichment of mitochondrial DNA, cloning, and sequencing. Large repeats and the dynamic mitochondrial genome organization complicate sequence assembly. The development of next generation sequencing technologies (NGS), such as the Roche and Illumina platforms, provides a new opportunity for rapid characterization of mitochondrial genomes.

Next-Generation Sequencing(NGS) or massively parallel sequencing- For the past 15 years, Sanger sequencing and fluorescence based electrophoresis technologies have been extensively used in somatic and germline genetic studies. Improvements in instrumentation coupled with the development of high performance computing and bioinformatics have reduced the cost of sequencing. However, increases in the throughput of Sanger DNA sequencing are achieved by the use of additional sequencers in parallel, owing to the requirement of gel electrophoresis or additional wells for the capillary sequencing of each reaction. Using different approaches, massively parallel sequencing methods overcome the limited scalability of traditional Sanger sequencing by either creating micro-reactors and/or attaching the DNA molecules to be sequenced to solid surfaces or beads, allowing for millions of sequencing reactions to happen in parallel. At present, there are four technologies commercially available and several other promising approaches are in various stages of development and implementation (Table 1) (Pettersson E, 2009). The current generation of massively parallel sequencers has led to a quantum leap in our ability to sequence genomes, so much so that 10-fold coverage of the human genome (30 Gb DNA sequence) can be obtained in a single run for no more than US$15,000 toUS$20,000. (Note that the Human Genome Sequencing Consortium generated 3 Gb at the cost of approximately US$3 billion and took 13 years!) (Reis-Filho *et al*, 2009).
Next-generation sequencing (also known as massively parallel sequencing) technologies are revolutionising our ability to characterise cancers at the genomic, transcriptomic and epigenetic levels. Cataloguing all mutations, copy number aberrations and somatic rearrangements in an entire cancer genome at base pair resolution can now be performed in a matter of weeks. Furthermore, massively parallel sequencing can be used as a means for unbiased transcriptomic analysis of mRNAs, small RNAs and noncoding RNAs, genome-wide methylation assays and high-throughput chromatin immunoprecipitation assays (Reis-Filho *et al*, 2009).



Method	Amplification	Read length (base pairs)	Templates per run	Data production/day	Sequence reaction	Reference
Commercially available technologies						
ABI 3730xl	PCR	~900 to 1,100	96	1 Mb/day	Sanger method	http://www.appliedbyosystems.com
454 FLX Roche	Emulsion PCR	~400	1,000,000	400 Mb/run/7.5 to 8 hours	Pyrosequencing	http://www.rocheapplied-science.com
Illumina (Solexa) Genome Analyzer	Bridge PCR	36 to 175	40,000,000	>17 Gb/run/3 to 6 days	Reverse terminator	http://www.illumina.com
ABI SOLiD	Emulsion PCR	~50	85,000,000	10 to 15 Gb/run/6 days	Ligation sequencing	http://www.appliedbyosystems.com
Helicos Heliscope	None	30 to 35	800,000,000	21 to 28 Gb/run/8 days	Single molecule sequence by synthesis	http://www.helicosbio.com
Technologies in development						
Pacific Biosciences	None	>1,000	NA	NA	Single molecule real-time DNA sequencing	http://www.pacificbiosciences.com
Intelligent Biosciences	Yes[a]	NA	NA	NA	Sequence by synthesis	http://www.intelligentbiosystems.com
Visigen Biotechnologies	None	NA	NA	NA	Base-specific FRET emission	http://www.visigenbio.com
ZS Genetics	None	NA	NA	NA	ZSG atomic labelling and electron microscopy	http://www.zsgenetics.com

Table #1: Summary of available NGS platforms

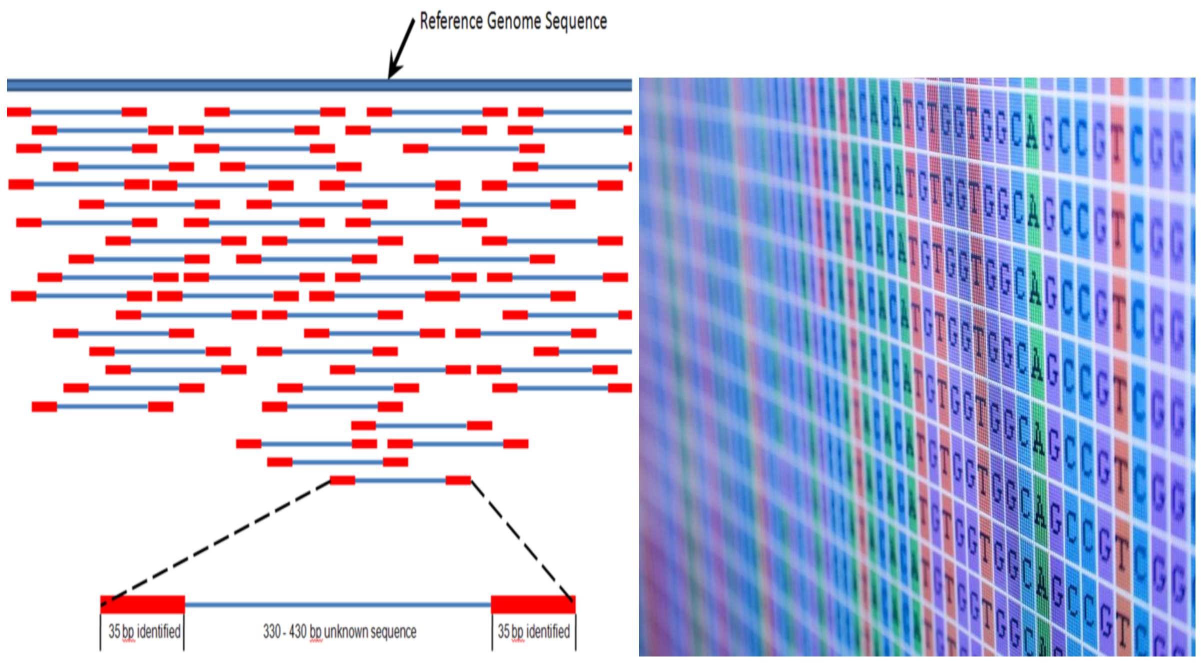


Figure #3 Genome Mapping using NGS Approach

Advent of NGS on mitochondrial genome assembly: The development of next generation sequencing technologies (NGS), such as the Roche and Illumina platforms, provides a new opportunity for rapid characterization of mitochondrial genomes. Non-enriched whole genome DNA libraries, both shotgun and paired-end, include plastid and mitochondrial DNA that is sequenced along with the nuclear DNA during the sequencing run thus eliminating the need for tedious organellar DNA isolation and characterization. NGS technologies have already been used for sequencing the small mitochondrial genome of nematodes (Jex *et al*, 2010), human (Gunnarsdóttir *et al*, 2011) and fish (Cui *et al*, 2009) with no library enrichment. Recently, sequencing data from non-enriched libraries has been successfully used to assemble plastid genomes of wild and domesticated rice, mung bean, date palm, and milkweed (Yang *et al*, 2010). The major limitations for use of this approach on *de-novo* assembly of mitochondrial genomes are the ability to overcome assembly problems related to large repeat regions, presence of promiscuous DNA, and sequence ambiguity due to sequencing technologies. The aim of this study was to demonstrate how next generation sequence (particulary Roche derived 454 NGS data from total genomic DNA can be used to *de-novo* assemble the mitochondrial genome of mulberry (*Morus indica* L.).

3.5 Roche derived 454 Next-Generation Sequencing Approach:

Sequencing Background- How is genome sequencing done?

Using 454 Sequencing on the Genome Sequencer FLX System, DNA from a genome is converted into sequence data through four primary steps:

Step One – DNA sample preparation;

Step Two – Proprietary process to load DNA sample onto beads;

Step Three – Sequencing DNA on Genome Sequencer FLX instrument; and

Step Four –Analysis of the genome.

Step 1: Sample Preparation

Starting with whole genome DNA or targeted gene fragments, the initial step in the process employed by 454 Sequencing System is a universal library preparation for any sample. One library preparation is sufficient for sequencing any DNA sample from a virus to a bacteria to a human. The first step is to break the double-helix DNA ladder into shorter double-stranded fragments of approximately 400 to 600 base pairs. The next step is to attach adapters to the DNA fragments. Finally, the double-stranded DNA fragments are separated into single strands (Sciences).

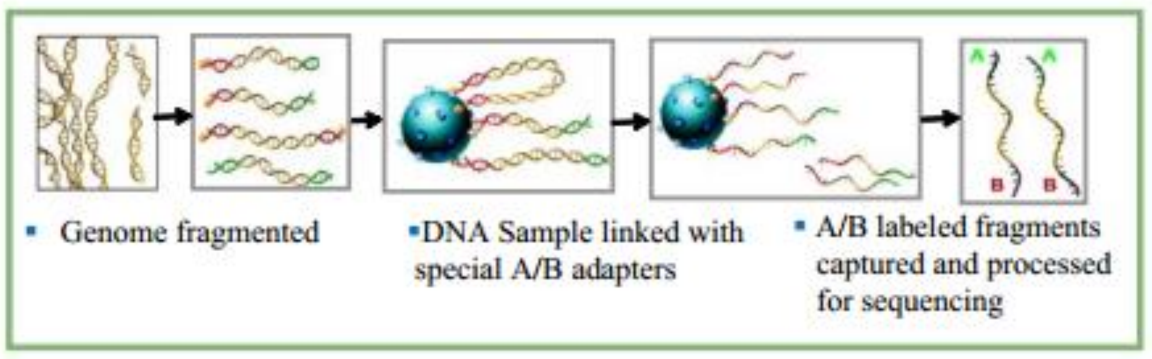

A
A
B
B
Genome fragmented
DNA Sample linked with special A/B adapters
A/B labeled fragments captured and processed for sequencing

Figure #4 Sample Preparation using 454 FLX Platform

Step 2: Loading DNA Sample onto Beads

Through the process of emulsion-based clonal amplification, or emPCR, the DNA library fragments are put onto micron-sized beads. As a result of the amplification of the DNA fragments, the signals produced during the sequencing step are easily detectable. This process takes approximately eight hours. Using the conventional Sanger method of cloning DNA in bacteria, the amplification process currently takes approximately three weeks and also introduces bias in the DNA samples (T *et al*, 2002). In the initial phase of the amplification process, the DNA library fragments along with capture beads and enzyme reagents in a water mixture, are injected into small, cylindrical plastic containers containing a synthetic oil. The combination of these materials and vigorous shaking causes the water mixture to form droplets around the beads, called an emulsion. Typically, most droplets that contain DNA will contain only one DNA fragment. The water mixture includes an enzyme that causes the single and isolated DNA fragment in each droplet to be amplified into millions of copies of DNA. This reaction is also known as a polymerase chain reaction, or PCR. Through this reaction, a single DNA fragment is amplified into approximately ten million identical copies that are immobilized on the capture beads. When the PCR reaction is complete, the beads are screened from the oil and cleaned. Those beads that do not hold DNA are eliminated. Those beads that hold more than one type of DNA fragment are readily filtered out during sequencing signal processing (Sciences) (T *et al*, 2002).

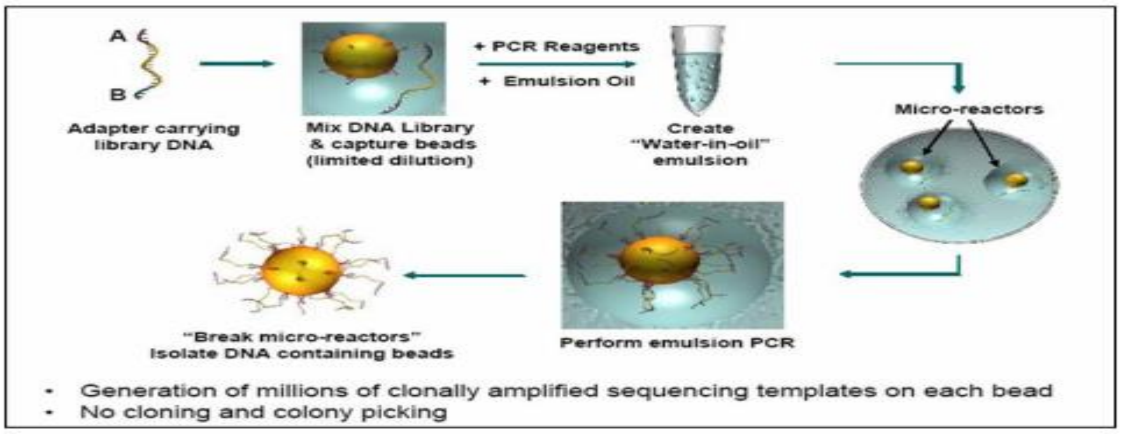

A
B
Adapter carrying library DNA
Mix DNA Library & capture beads (limited dilution)
+ PCR Reagents
+ Emulsion Oil
Create "Water-in-oil" emulsion
Micro-reactors
"Break micro-reactors" Isolate DNA containing beads
Perform emulsion PCR
• Generation of millions of clonally amplified sequencing templates on each bead
• No cloning and colony picking

Figure #5: Loading of DNA Samples onto Beads.

Step 3: Sequencing

The 454 Sequencing process uses sequencing by synthesis approach to generate sequence data. In sequencing by synthesis, a single-stranded DNA fragment is copied with the use of an enzyme making the fragment double stranded. Starting at one end of the DNA fragment, the enzyme sequentially adds a single nucleotide that is the match of the nucleotide on the single strand. Nucleotides are paired one by one as the enzyme moves down the single stranded fragment to extend the double-helix ladder structure (Legkari, 2010).

Following the separation and amplification of DNA strands with the library preparation and emPCR kits, the DNA-capture beads are placed on our Pico Titer Plate for sequencing. The Pico Titer Plate is a major technological advancement because it enables the miniaturization of sequencing with our technology. One side of the Pico Titer Plate is polished and the other side of the plate contains wells that are 75 picoliters in volume. Each Pico Titer Plate comprises 1.6 million wells. The diameter of the wells is designed so that only a single capture bead will fit into each well (Sciences).

How the 454 Sequencing process works?

a) Bases (TACG) are flown sequentially and always in the same order (100 times for a large FLX run) across the PicoTiterPlate during a sequencing run.

b) A nucleotide complementary to the template strand generates a light signal

c) The light signal is recorded by the CCD camera

d) The signal strength is proportional to the number of nucleotides being incorporated.

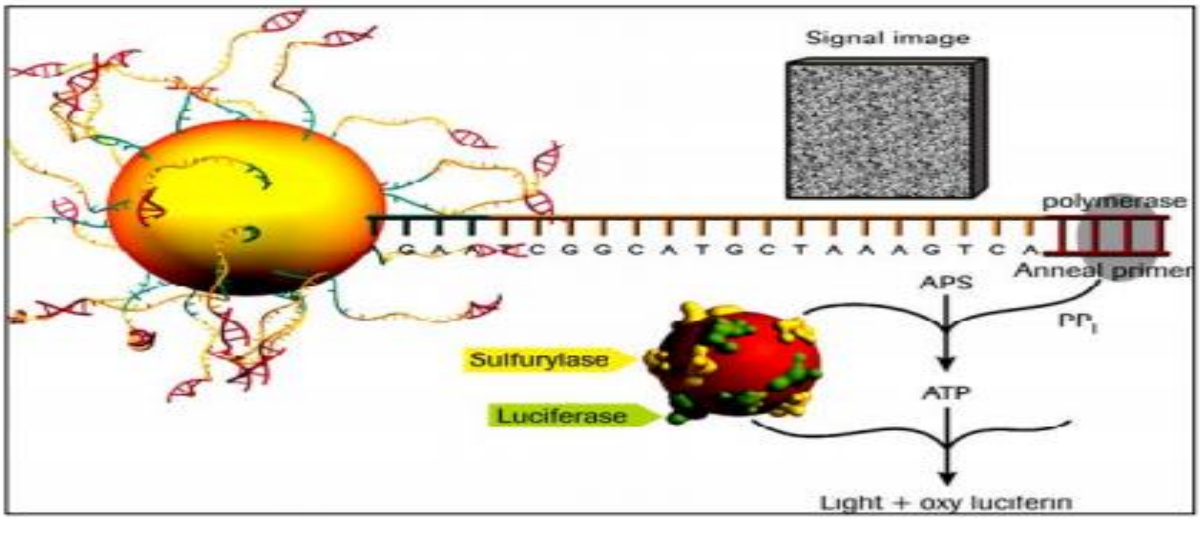


Figure #6: Pyrosequencing based 454 derived NGS approach

The chemi-luminescent signal produced in this reaction is detected by the CCD camera assembly included in the instrument. A CCD camera uses a small, rectangular piece of silicon rather than a piece of film to receive incoming light. This is a special piece of silicon called a charge-coupled device, or CCD. The intensity of light generated during the flow of a single nucleotide varies proportionately with the consecutive number of complementary nucleotides on the single-stranded DNA fragment being analyzed. For example, if there are three consecutive A's in the single-stranded fragment, the amount of light generated would be three times that of a single A in the fragment. The signals created in the sequencing process are then analyzed by the 454 Sequencing System's software to generate millions of sequenced bases per hour from a single run (Legkari, 2010).

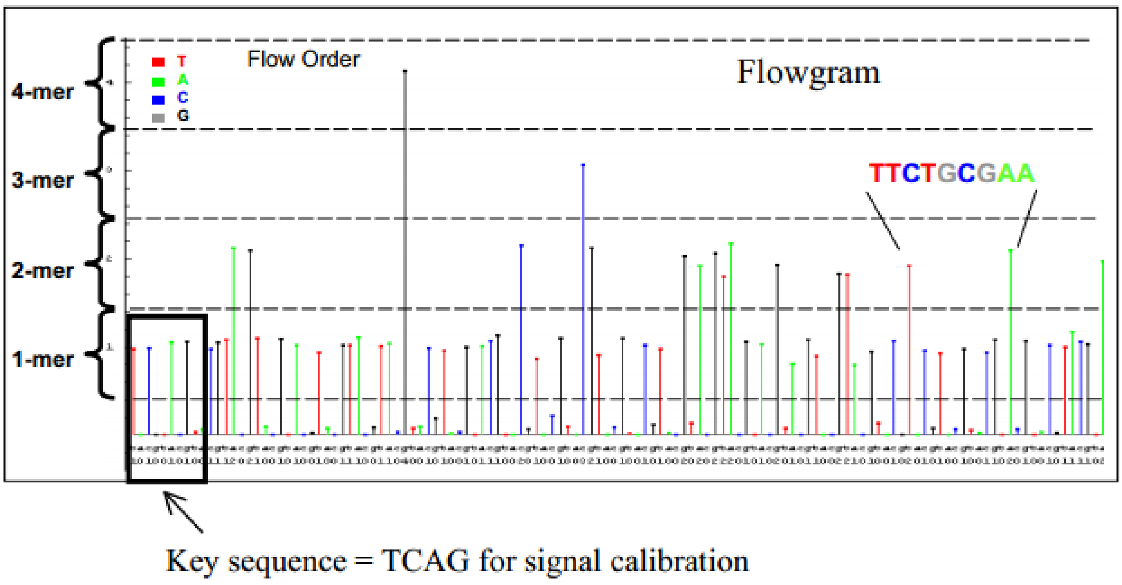


Figure #7: Flow-gram generated by Pyro-sequencing. This Flow-gram is created based upon the chemi-luminescent signal. It's a bar-graph of light intensities for each well contained on PicoTitrePlate. The signal strength is proportional to the number of nucleotides incorporated.

Step 4: Analysis of the Genome

Data generated by 454 Sequencing on the Genome Sequencer FLX has the unique advantage of high throughput combined with longer read length to create a more complete picture of the human genome. By eliminating bias from sample preparation known to exist from traditional sequencing technologies and speeding up the time, quality and depth of sequencing results per run, one is able to now tackle the analysis of an entire individuals' genome. Results of each GS FLX run (a multitude of flowgrams) are collected and compared to the reference genome, such as that generated from the Human Genome Project, to detect regions of exact match and differences (Sciences).

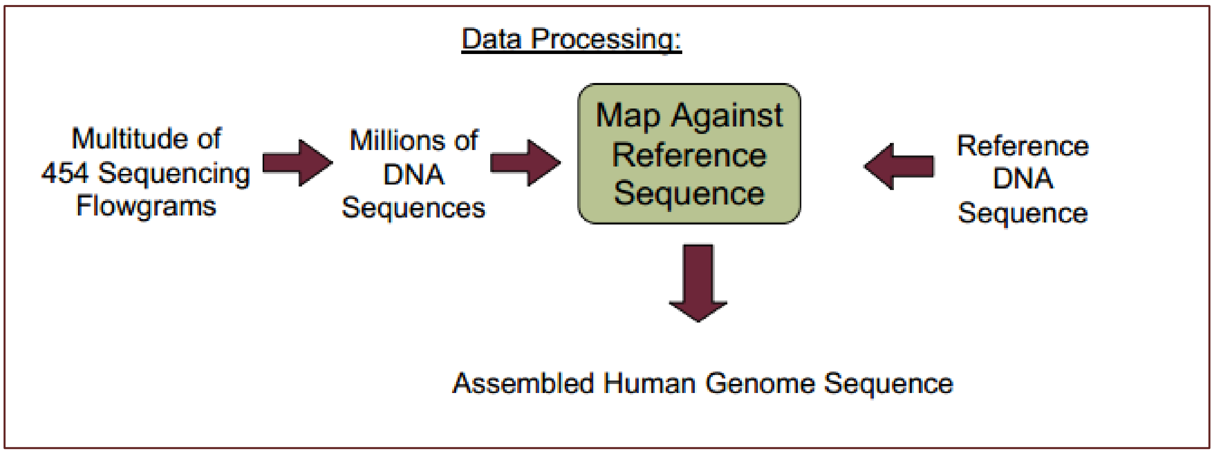

Data Processing:
Multitude of 454 Sequencing Flowgrams
Millions of DNA Sequences
Map Against Reference Sequence
Reference DNA Sequence
Assembled Human Genome Sequence

Figure #8: Data Processing after obtaining the sequencing reads.

3.6 Genome Assembly

Genome Assembly: Genome assembly refers to the process of taking a large number of short DNA sequences and putting them back together to create a representation of the original chromosomes from which the DNA originated.

In a shotgun sequencing project, all the DNA from a source (usually a single organism, anything from a bacterium to a mammal) is first fractured/sheared into millions of small pieces. These pieces are then read by automated sequencing machines, which can read up to 1000 nucleotides or bases at a time. A genome assembly algorithm works by taking all the pieces and aligning them to one another, and detecting all places where two of the short sequences called reads, overlap. These overlapping reads can be merged, and the process continues (Krasileva *et al*, 2013).

Genome assembly is a very difficult computational problem, made more difficult because many genomes contain large numbers of identical sequences, known as repeats. These repeats can be thousands of nucleotides long, and some occur in thousands of different locations, especially in the large genomes of plants and animals.

The resulting (draft) genome sequence is produced by combining the information sequenced contigs and then employing linking information to create scaffolds. Scaffolds are positioned along the physical map of the chromosomes creating a "golden path" (Yang *et al*, 2013).

3.7 Assembly software

Originally, most large-scale DNA sequencing centres developed their own software for assembling the sequences that they produced. However, this scenario has changed as the software has grown more complex and as the number of sequencing centres has increased. An example of such tailor-made assembler is *Short Oligonucleotide Analysis Package* developed by BGI for de novo assembly of human-sized genomes, alignment, SNP detection, re-sequencing, indel finding, and structural variation analysis (Li *et al*, 2010). To assemble a genome, computer programs typically use data consisting of single and paired reads. Single reads are simply the short sequenced fragments themselves; which can be joined up through overlapping regions into a continuous sequence known as a 'contig'. Repetitive sequences, polymorphisms, missing data and mistakes eventually limit the length of the contigs that assemblers can build.

Paired reads typically are about the same length as single reads, but they come from either end of DNA fragments that are too long to be sequenced straight through. Depending on the library preparation technique, the distance between the paired reads can be as short as 200 base pairs or as large as several tens of kilobases (Eren *et al*, 2013). Since the paired reads are generated from the same piece of DNA, they can help link contigs into 'scaffolds', which are ordered assemblies of contigs with gaps in between. Paired-read data can also indicate the size of repetitive regions and the distance between the two contigs (Eren *et al*, 2013).

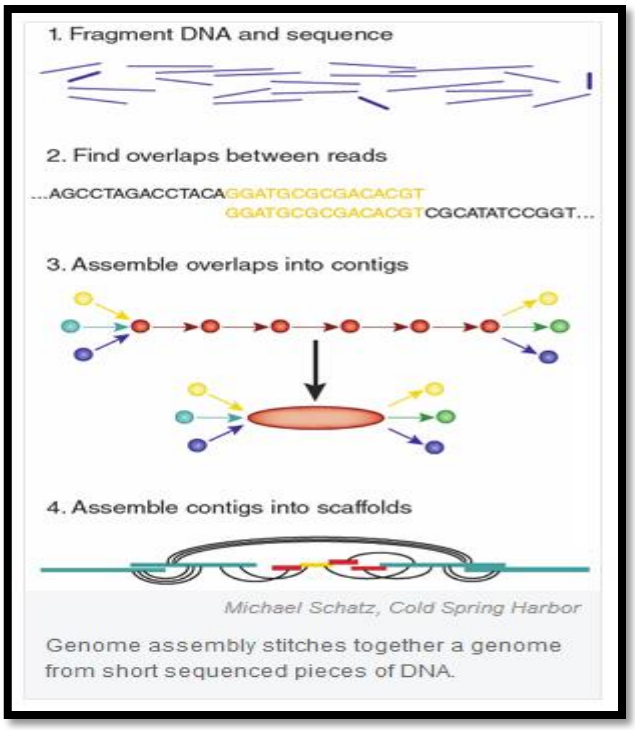

1. Fragment DNA and sequence
2. Find overlaps between reads
...AGCCTAGACCTACAGGATGCGCGACACGT
GGATGCGCGACACGTCGCATATCCGGT...
3. Assemble overlaps into contigs
4. Assemble contigs into scaffolds
Michael Schatz, Cold Spring Harbor
Genome assembly stitches together a genome from short sequenced pieces of DNA.

Figure #9 Overview of Genome Assembly

3.8 *De novo* vs. Mapping Assembly

In sequence assembly, two different types can be distinguished:

1. *De-novo*: assembling short reads to create full-length novel sequences.
2. Mapping: assembling reads against an existing backbone sequence, building a sequence that is similar but not necessarily identical to the backbone sequence

In terms of complexity and time requirements, de-novo assemblies are orders of magnitude slower and more memory intensive than mapping assemblies. This is mostly due to the fact that the assembly algorithm needs to compare every read with every other read (an operation that has a complexity of O(n^2) but can be reduced to O(n log(n)) (Góngora *et al*, 2013).

3.9 Judging Genome

In the absence of a high-quality reference genome, new genome assemblies are often evaluated on the basis of the number of scaffolds and contigs required to represent the genome, the proportion of reads that can be assembled, the absolute length of contigs and scaffolds, and the length of contigs and scaffolds relative to the size of the genome. The most commonly used metric is N50, the smallest scaffold or contig above which 50% of an assembly would be represented. But this metric may not accurately reflect the quality of an assembly (Baker, 2012). An early assembly of the sea squirt *Ciona intestinalis* had an N50 of 234 kilobases. A subsequent assembly extended the N50 more than tenfold, but a recent analysis showed that this assembly lacked several conserved genes, perhaps because algorithms discarded repetitive sequences (Korf *et al*, 2012). This is not an isolated example: the same analysis found that an assembly of the chicken genome lacks 36 genes that are conserved across yeast, plants and other organisms. But these genes seem to be missing from the assembly rather than the organism. The focused re-analysis of the raw data found most of these genes in sequences that had not been included in the assembly (Korf *et al*, 2012).

3.10 Genome Annotation

Genome annotation is the process of attaching biological information to sequences (Stein *et al*, 2001). It consists of three main steps:

1. Identifying portions of the genome that do not code for proteins.
2. Identifying elements on the genome, a process called gene prediction, and
3. Attaching biological information to these elements.

Automatic annotation tools try to perform all this by computer analysis, as opposed to manual annotation (a.k.a. curation) which involves human expertise. Ideally, these approaches co-exist and complement each other in the same annotation pipeline.

The basic level of annotation is using BLAST for finding similarities, and then annotating genomes based on that (Pevsner *et al*, 2009). However, nowadays more and more additional information is added to the annotation platform. The additional information allows manual annotators to de-convolute discrepancies between genes that are given the same annotation. Some databases use genome context information, similarity scores, experimental data, and integrations of other resources to provide genome annotations through their Subsystems approach. Other databases (e.g. Ensembl) rely on both curated data sources as well as a range of different software tools in their automated genome annotation pipeline.

Structural annotation consists of the identification of genomic elements.

- ORFs and their localisation
- Gene structure
- Coding regions
- Location of regulatory motifs

Functional annotation consists of attaching biological information to genomic elements.

- Biochemical function
- Biological function
- Involved regulation and interactions
- Expression

Various biological Experiments are required to accomplish these steps. Proteo-genomics based approaches utilize information from expressed proteins, often derived from mass spectrometry, to improve genomics annotations. (Gupta *et al*, 2009)

CHAPTER #4

METHODOLOGY

4.1 Quality Control of the 454 Pyro-sequencing derived raw read files.

Sequencing technologies are not perfect and the quality control (QC) is an essential step to ensure that the data used for downstream analysis is not compromised of low-quality sequences, sequence artefacts, or sequence contamination that might lead to erroneous conclusions.

4.1.1 Standard Flow-gram Format (SFF)

The raw reads files obtained from the 454 Pyro-sequencing experiment is called a Standard Flow-gram Format file (SFF file). Standard flow-gram format (SFF) is a binary file format used to encode results of pyro-sequencing from the 454 Life Sciences platform for high-throughput sequencing. These files hold the information about:

a) The Flow-gram,
b) The called sequence,
c) The quality of the called sequence.
d) And the recommended quality and adaptor clippings.

These recommended clippings are given by the 454 sequencer. The Roche software takes into account the quality and the adaptor sequence to recommend a clipping for each sequence. This is done based upon initial library preparation protocol. Binary Format files cannot be accessible by usual text editors and special programs are designed to view and edit the raw reads before assembling. There are several tools to extract the sequences and to convert them to a more usable format. Roche provides some executable files to perform this task. Alternatively we can use the sff_extract tool to obtain a fasta file. sff_extract extracts the reads from the sff files and stores them into fasta and xml or caf text files.

Tools which comes with 454 machine helps in extracting FASTA and QUALITY (QUAL) files from raw reads sff files like sffinfo. The QUAL files are files containing the Phred quality scores of each base in the raw read file.

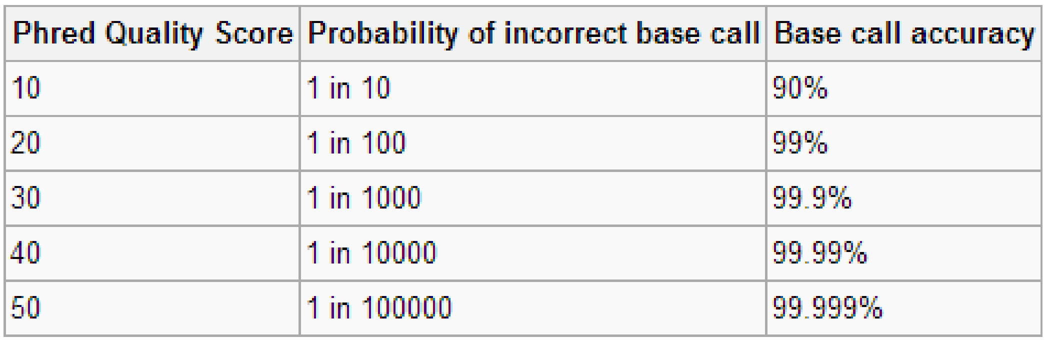
Parameters to be kept in mind for Quality Control are as follows:

4.1.2 Phred Quality Score

Phred quality scores ‘Q’ are defined as a property which is logarithmically related to the base-calling error probabilities ‘P’ (Ewing B, 1998).

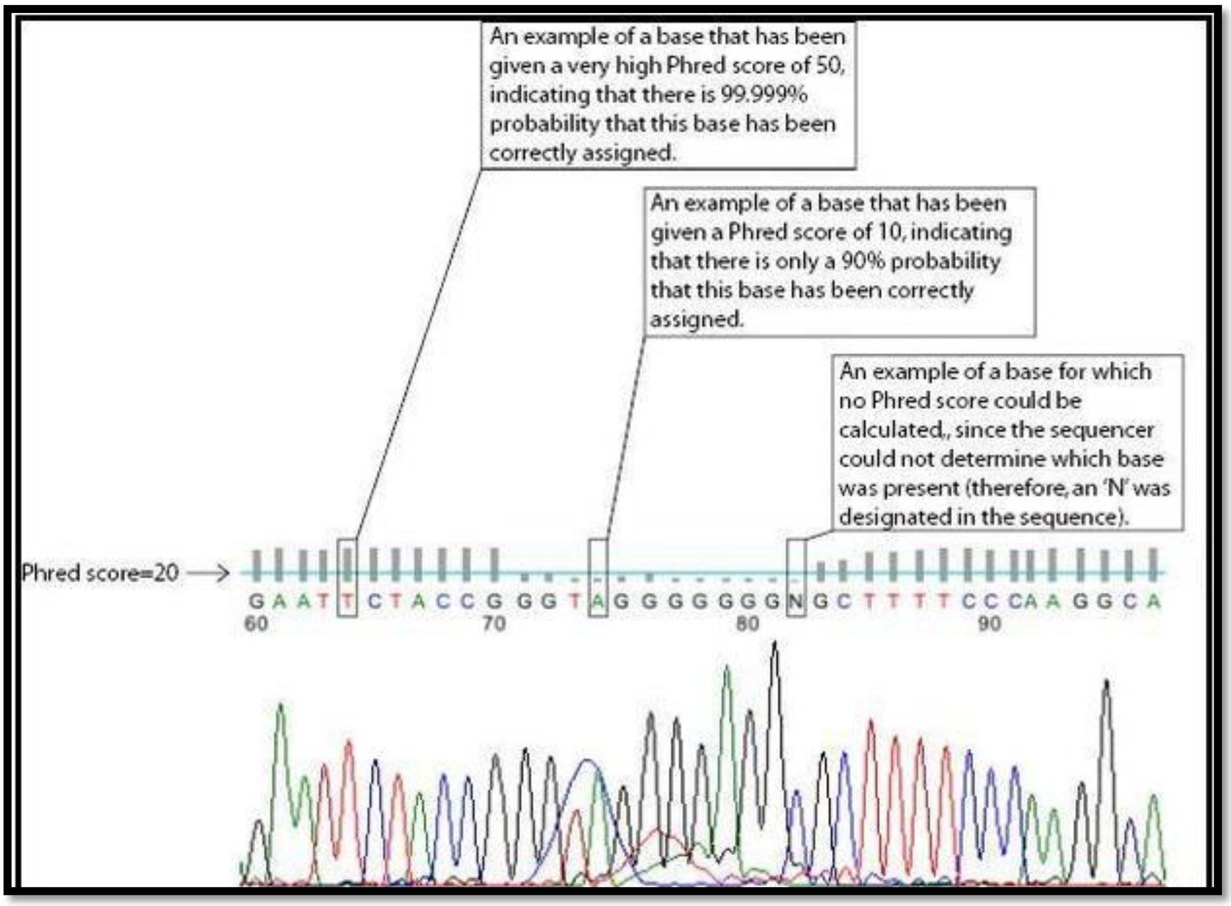
$$\mathbf{Q = -10 \log_{10} P}$$

For example, if Phred assigns a quality score of 30 to a base, the chances that this base is called incorrectly are 1 in 1000. The most commonly used method is to count the bases with a quality score of 20 and above. The high accuracy of Phred quality scores make them an ideal parameter to assess the quality of sequences.



Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10000	99.99%
50	1 in 100000	99.999%

Table #2 Phred quality scores are logarithmically linked to error probabilities


An example of a base that has been given a very high Phred score of 50, indicating that there is 99.999% probability that this base has been correctly assigned.

An example of a base that has been given a Phred score of 10, indicating that there is only a 90% probability that this base has been correctly assigned.

An example of a base for which no Phred score could be calculated,, since the sequencer could not determine which base was present (therefore, an 'N' was designated in the sequence).

Phred score=20
G A A T T C T A C C G G G T A G G G G G G G N G C T T T T C C C A A G G C A
60
70
80
90

Figure #10 DNA sequence traced according to Phred scores (grey bars) in a typical DNA sequencing base-call.

4.1.3 Number and Length of Sequences

The length distribution of sequence reads can be used as quality measure for the sequencing run. Best data-set usually follow a normal distribution. However, most sequencing results show a slowly increasing and then a steep falling distribution which is quite expected as the sequence distribution will increase as the length of the read increases.

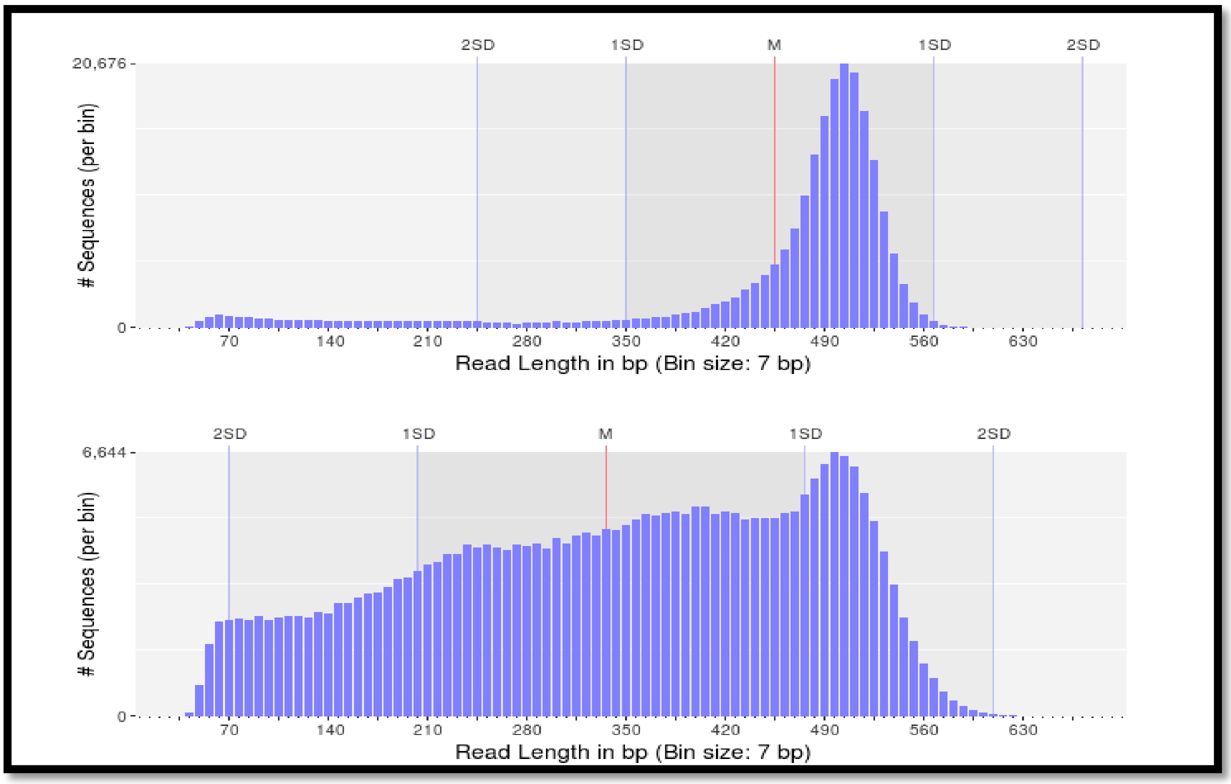


Figure #11 Example of Sequence length distributions in two random samples.

Both distributions have the highest number of sequences around 500 bp, but for the first dataset the mean of the sequence lengths is higher and the standard deviation is lower. A certain number of shorter reads might be expected, but if the sample contained mainly longer fragments, it should be low. Assuming that both samples contained enough fragments of at least 500 bp and all fragments were sequenced with the same number of cycles (sequencing flows), we would expect that the majority of the sequences would have approximately the same length. The higher amount of shorter reads in the second dataset suggests that those reads might have been of lower quality and were trimmed during the signal processing. If the sample contained many short fragments, the shorter reads might be from those fragments and not of lower quality.

Minimum and maximum read length

Sequences in the SFF files can be as short as 40 bp (shorter sequences are filtered during signal processing). For multiplexed samples, the MID trimmed sequences can be as short at 28 bp (assuming a 12 bp MID tag). Such short sequences can cause problems during, for example, database searches to find similar sequences. Short sequences are more likely to match at a random position by chance than longer sequences and may therefore result in false positive functional or taxonomical assignments. In some cases, sequences can be much longer than several standard deviations above the mean length (e.g. 1,500+ bp for a 500 bp mean length with a 100 bp standard deviation). Those sequences should be used with caution as they likely contain long stretches of homopolymer runs as in the following example below. Homopolymers are a known issue of pyro-sequencing technologies such as 454/Roche.

```
aactttaaccttttaaaacccccttaaaaaaactttaaaccccgtaaaccccccgggttt
tttttaaaaaaccgttttttacgggggtttaccccgttttaccggggttttgggggttt
taaaaaaaacgggttttaaacgggttaacccccgggttttccgggggtttaaaaagtttt
tttaaacgggggttttcccgtaaaaaaaaaaccccgtttaaaaaaaggggttaaaaaaaa
aaggggttaacccccccggggtttaaaaaaaaccttttttttttaaaaaaaacgttttt
ttttttaaaagggggtttttttacgggggtaaacgggggggttaaaaaaaaacccccc
cgggggttttaaaaaaaaaaccccccggttttaaaaaaccccgttttaaccccctttaaaa
aaaaaacggggggtttaaaaaaaaaaaggggtttttttttttaaaaacccgttttta
aaaacccccccgtttttaacccgggttaaaccccccccgggggggtaaaacccccccccc
ggggtaacccccttttttaaaaccccccccccgttttttacccgggggttttaccccccg
gggggggtaaaaaaacgggggggtttttttttttaaaaccggggtttttttttttaaa
ccccggttttaaaaaccggttttaccccgggggggttttacccccgggggggggttttt
aaaacccccggtttaaaactttaaaaacccgggtaaccccggggttttaaaaaaaaaaaa
aaaacccccccccgttaaaaaaaaaaaacccgtttttttttaaaaaaaaccccccccccgg
ttttaaaacccccccccgggggttttaccccggggtttaaaaaaaaacccgtttaaaaaa
accgggttttttaaaggggttttaaaccccccccccc
```

The above sequence represents homopolymer ends which has to be handled with care.

In genomics, a homopolymer is a sequence of identical bases, like AAAA or TTTTTTTT. Homopolymers appear as subsequences in larger sequences; in this case the size of the homopolymer is referred to as the homopolymer length (Beuf *et al*, 2012).

Very long homopolymers form repeats and are difficult to sequence. They are fortunately very rare, though they do appear in genomes more often than statistical randomness would suggest, especially in junk DNA.

Homopolymers in 454 Sequencing

The 454 sequencing method does not call bases directly. Instead it calls flows, which are indicated by a light signal. Each flow represents a homopolymer, and the brightness of the light indicates the length of the homopolymer. Hence the sequence TAAAAA would appear as a small light to mark the T, followed by a much brighter light to mark the 5 A's. The danger in this process is that the brightness of the light is easy to mis-calibrate, especially for long homopolymers. As a result, 454 reads often contain homopolymer-length sequencing errors, such as calling AAAAA as AAAAAA or vice versa.

GC content

The GC content distribution of most samples should follow a normal distribution. In some cases, a bi-modal distribution can be observed, especially for meta-genomic data sets. The GC content plot in PRINSEQ marks the mean GC content (M) and the GC content for one and two standard deviations (1SD and 2SD). This can help to decide where to set the GC content thresholds, if a GC content filter will be applied. The plot can also be used to find the thresholds or range to select sequences from a bi-modal distribution.

Poly-A/T tails

Poly-A/T tails are considered repeats of As or Ts at the sequence ends. In PRINSEQ, the minimum length of a tail is 5 bp and sequences that contain only As or Ts are counted for both ends. A small number of tails can occur even after trimming poly-A/T tails. For example, a sequence that ends with AAAAATTTTT and that has been trimmed for the poly-T will contain the Poly A. Trimming poly-A/T tails can reduce the number of false positives during database searches, as long tails tend to align well to sequences with low complexity or sequences with tails (e.g. viral sequences) in the database.

Sequence duplications

Assuming a random sampling of the genomic material in an environment such as in metagenomic studies, reads should not start at the same position and have the same errors (at least not in the numbers that they have been observed in most metagenomes). Recent Study (Gomez-Alvarez *et al*, 2010) investigated the problem in more detail and did not find a specific pattern or location on the sequencing plate that could explain the duplications. Duplicates can arise when there are too few fragments present at any stage prior to sequencing, especially during any PCR step. Furthermore, the theoretical idea of one micro-reactor containing one bead for 454/Roche sequencing does not always translate into practice where many beads can be found in a single micro-reactor. Unfortunately, artificial duplicates are difficult to distinguish from exactly overlapping reads that naturally occur within deep sequence samples. The number of expected sequence duplicates highly depends on the depth of the library, the type of library being sequenced (whole genome, transcriptome, 16S, metagenome,), and the sequencing technology used. The sequence duplicates can be defined using different methods. Exact duplicates are identical sequence copies, whereas 5' or 3' duplicates are sequences that are identical with the 5' or 3' end of a longer sequence. Considering the double-stranded nature of DNA, duplicates could also be considered sequences that are identical with the reverse complement of another sequence.
Depending on the dataset and downstream analysis, it should be considered to filter sequence duplicates. The main purpose of removing duplicates is to mitigate the effects of PCR amplification bias introduced during library construction. In addition, removing duplicates can result in computational benefits by reducing the number of sequences that need to be processed and by lowering the memory requirements. Sequence duplicates can also impact abundance or expression measures and can result in false variant (SNP) calling.

Sequence complexity

Genome sequences can exhibit intervals with low-complexity, which may be part of the sequence dataset when using random sampling techniques. Low-complexity sequences are defined as having commonly found stretches of nucleotides with limited information content (e.g. the dinucleotide repeat CACACACACA). Such sequences can produce a large number of high-scoring but biologically insignificant results in database searches.

Tag sequences

Tag sequences are artifacts at the ends of sequence reads such as multiplex identifiers, adapters, and primer sequences that were introduced during pre-amplification with primer-based methods. The base frequencies across the reads present an easy way to check for tag sequences. If the distribution seems uneven (high frequencies for certain bases over several positions), it could indicate some residual tag sequences.

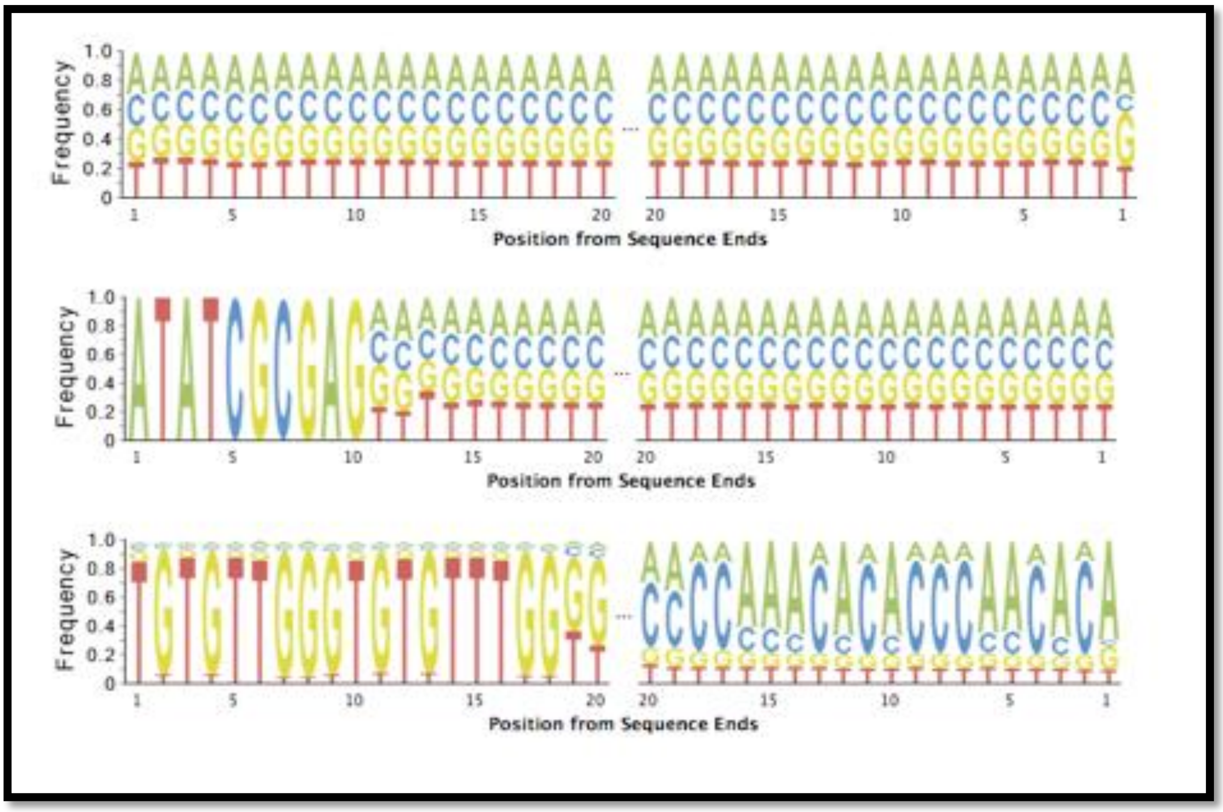


Figure #12 Graphical representation showing sequences with tags.

Assembly quality measures

The Nxx contig size is a weighted median that is defined as the length of the smallest contig C in the sorted list of all contigs where the cumulative length from the largest contig to contig C is at least xx% of the total length (sum of contig lengths). Replace xx by the preferred value such as 90 to get the N90 contig size. The higher the Nxx value, the higher the rate of longer contigs and the better the dataset. If the dataset does not contain contigs or scaffolds, this information can be ignored.

PrinSeq

PRINSEQ is a tool that generates summary statistics of sequence and quality data and that is used to filter, reformat and trim next-generation sequence data. It is particular designed for 454/Roche data, but can also be used for other types of sequence data. PRINSEQ is available through a user-friendly web interface or as standalone version. The standalone version is primarily designed for data preprocessing and does not generate summary statistics in graphical form. This tools first generates a summary report of the raw data and the provides an option for processing the input data according to the summary generated.

PrinSeq was used for Quality Control of the input sff file. It takes into account

a) Number and Length Distribution of the data.

b) Phred Quality Scores

c) Sequence Contamination

d) Sequence Complexity

e) Homopolymer trimming

f) GC Content

g) Poly A/T Tails

A web server and a standalone version of the tool is available (Schmieder, 2011).

4.2 Sequence Assembly

For assembling the raw reads into contigs and scaffolds gsAssembler or newbler was used. Newbler is a 454 platform specific software is used to assemble 454 Pyrosequencing reads.

While assembling Newbler generates a file called 454NewblerProgress.txt which explains the step-by-step assembly algorithm followed by Newbler. Newbler like most assembly softwares works on the principle of de-bruijn graph. In graph theory, an *n*-dimensional De Bruijn graph of *m* symbols is a directed graph representing overlaps between sequences of symbols. Applying De Bruijn graph to genome assembly each read is represented by a node and overlap between reads is represented by an arrow (called a directed-edge) between the two reads. For instance, two nodes representing reads may be connected with a directed edge if the reads overlap by at least five nucleotides (Phillip *et al*, 2011). While assembling the 454NewblerProgress.txt the first message is indexing reads. During indexing, newbler scans the input file, performs some checks and trims the reads (sometimes more than the base-calling software already did). One of the checks is for possible 3′ and 5′ primers: if a certain percentage of reads contains the same sequence on either the 3′ or 5′ end, this is mentioned. The next phase is to find overlap between the reads. Newbler splits this phase into one for long reads (this goes very fast) and shorter reads (can take quite some time). As aligning all reads against each other would take too long time, newbler (and many other programs) actually make seeds, 16-mers of each read, where each seed starts 12 bases upstream of the previous one. These seed length and step sizes can be changed if you want . When two different reads have identical seeds the program tries to extend the overlap between the reads until the minimum overlap (default 40 bp) with the minimum alignment percentage default 90%) has been reached. After long overlap follows short overlap. Last stage is checkpointing. Basically, checkpointing means writing the intermediate results to disc, so that in the case of a crash, you could continue the assembly from the last ‘checkpoint’. At this point, newbler, as many other assemblers, has created a contig graph. Aligned reads form the ‘nodes’, reads going from one contig to another form the ‘edges’. For example, a small part of the graph could like like this:

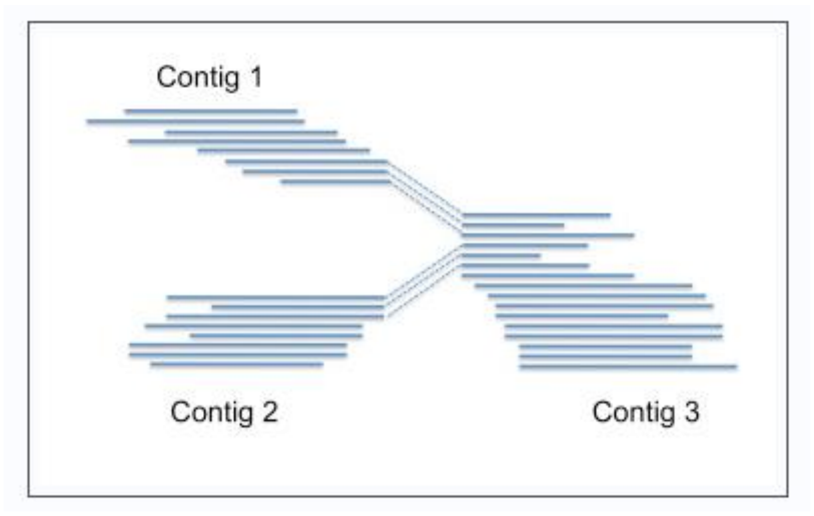


Figure #13 Contig-Graph (Equivalent to De-Bruijn Graph)

After aligning all the reads, the contig graph potentially has many nodes and edges. The size and complexity of the graph depend on the size of the genome and the repeat structure. The ‘real’ genome is a path through the graph visiting all nodes (Flxlex, 2010).

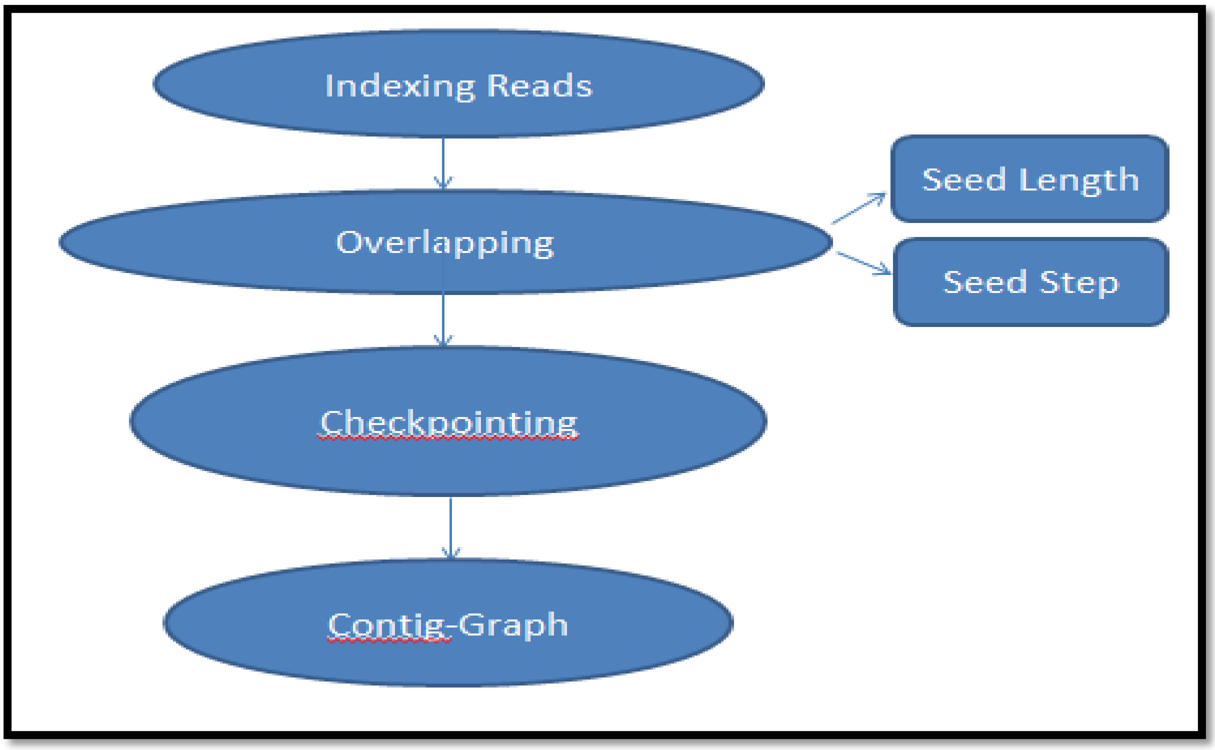


Figure #14 Newbler’s Algorithm.

4.2.1 Parameters used in the assembly

The assembly of quality-filtered reads was done using Newbler with the following parametrs:

a) Trimming Database: A Primer sequence database used in library preparation before sequencing was trimmed before assembly into contigs.
b) Screening Database: Before Assembly reads were screened with a local Plant Chloroplast database accessible at NCBI.
c) Seed Step = 12
d) Seed Length = 16
e) Minimum Overlap Length = 40
f) Minimum Overlap Identity = 90

The work focussed on combining the raw sequence data of two mulberry parents of the mapping population to increase the read coverage in the pooled data set and to include the regions which were not sequenced in either of the parents. Hence, the assembly of raw reads from both the parents was carried out using the above parameters. This generated two 454Contigs.fna and 454Contigs.qual files for both the parents

4.3 Plant Mitochondrial Genome Database:

The published mitochondrial genomes of plants were downloaded from http://www.ncbi.nlm.nih.gov/genomes/GenomesGroup.cgi?taxid=33090&opt=organelle. A local database was made by makeblastdb (make blast database) by ncbi toolkit. This was used to find mt-like contigs in assembled contigs files generated by gsAssembler run by local BLAST.

4.4 Extracting Reads from mt-like contigs

Contigs from 454Contigs.fna (FASTA file of contigs) files from both the parents which showed best hit in blast results were extracted. The reads which formed these contigs were extracted from 454ReadStatus.txt file of both the parents by in-house developed shell script. The mt-like reads (reads forming mt-like contigs) were used to make a new SFF file (raw read file) for the Newbler's De-novo Assembly. This was executed by the Newbler's inbuilt command called sfffile. By running this command on the linux-shell, one can make a raw Standard Flowgram Format file from FASTA file or from the QUAL file of the reads.

The mitochondrial like reads in both the parents could be identified by simply blasting the FASTA files of raw reads with the local plant mitochondrial database. This appears quite uncomplicated but it can increase the artifacts produced during the assembly, and may produce false-negatives. Contigs from pre-assembled raw read files would take care of the false-negatives, as contigs are large sequences formed by high quality overlapping reads.

4.5 De-novo Assembly of mt-like reads of both the parents

The raw mt-like reads SFF files were fed into the assembly program with the parameters stated in the previous assembly. The assembly was carried out for individual parents as well as the pooled data set (combining the mt-like SFFs of both the parents into a new joint_mt_like_sff file). The merging of the two mt-like SFF files was done by sfffile command of Newbler.

The joint mitochondrial assembly was done to increase the coverage (read-depth) of the regions common in both the parents and to include the regions being missed out while sequencing in either of the parents. This was possible because the mitochondrial genomes of plants are conserved in terms of gene-content. Hence, pooling the data would validate the genes found in both the parents and may also find genes missing in either of the parents (Lima J *et al*, 2012).

4.6 Statistical Evaluation of Contigs formed

The number and length of reads, the quality score, the N50 value and the average read coverage for each contigs was evaluated. The contigs passing this filter were further selected for annotation.

4.7 Sreening for nuclear DNA (numts) in mitochondrial De-novo Contigs

Plant Mitochondrial genome has some copies of numts which are usually pseudogenes. These need to be screened before annotation of mitochondria. Read Coverage (number of reads overlapping to form a contig) is an effective stat which was used to initially estimate the nuclear copies in mitochondria. Newbler estimates the read coverage for each contig in 454ContigGraph.txt file which is generated after the assembly process. Ideally the read coverage of a nuclear copy would be low as compared to the organellar copies of a particular genomic DNA. This is attributed to the fact that a nucleus contains many copies of mitochondria. So a portion of DNA which is present in both the nucleus as well as in the mitochondria should be represented in many numbers in the mitochondrial genome. Hence, the contigs assembled by the joint_mt_like_sff file were checked for their read coverage to remove nuclear counterfacts. The read coverage of 30 or above was chosen to be coming from mitochondria and the contigs with the read coverage <10 were attributed to be the nuclear copies (Michalovova *et al*, 2013).

The predicted nuclear copies were further validated by blasting the sequence of those contigs against NCBI's non-redundant nt (translated) database to check for the presence of putative conserved domains. The putative domains, if found by the BLAST search were then fed into ORFPredictor to find a functional gene. No ORF / CDS validated the presence of pseudogenes and hence numts.

4.8 Annotation of Mitochondrial genome

Annotation of mitochondrial genome was done by MITOFY (Alverson, 2010) and tRNAScan-SE. The High-quality, high read-coverage were checked for the presence of known ORFs, mitochondrial genes, tRNA genes and RNA genes. Some perl scripts were also written for orfprediction and formatting the contigs before before MITOFY annotation.

4.9. Scaffolding – Genome Finishing

4.9.1 Establishing Contig-Connections

For Connections or nodes between de-novo contigs a perl script bb.454contignet.pl developed by Simon Lab was used. This is a Perl program that will take an assembly of Roche 454 sequences generated by the Roche newbler/gsAssembler, and use the connection information to link generated contigs into a graphical map (Massimo Iorizzo *et al*, 2012). A large amount of information about connections between various contigs in the gsAssembler assembly is contained in the 454ContigGraph.txt file generated by gsAssembler. It's this information which is exploited by the perl script to generate contig connections graph.

4.9.2 Aligning the contigs with each other to look for possible overlaps

All the contigs were aligned to each other by CodonCode Aligner. Parameters used were 90 % identity with the minimum overlap of 40 with Large-gap alignment option for CodonCode Aligner.

The neat connections in between the contigs shown by the contig connection graph were tested by aligning them together in CodonCode. Alignment of contigs with each other provided information about repeats in the genome.

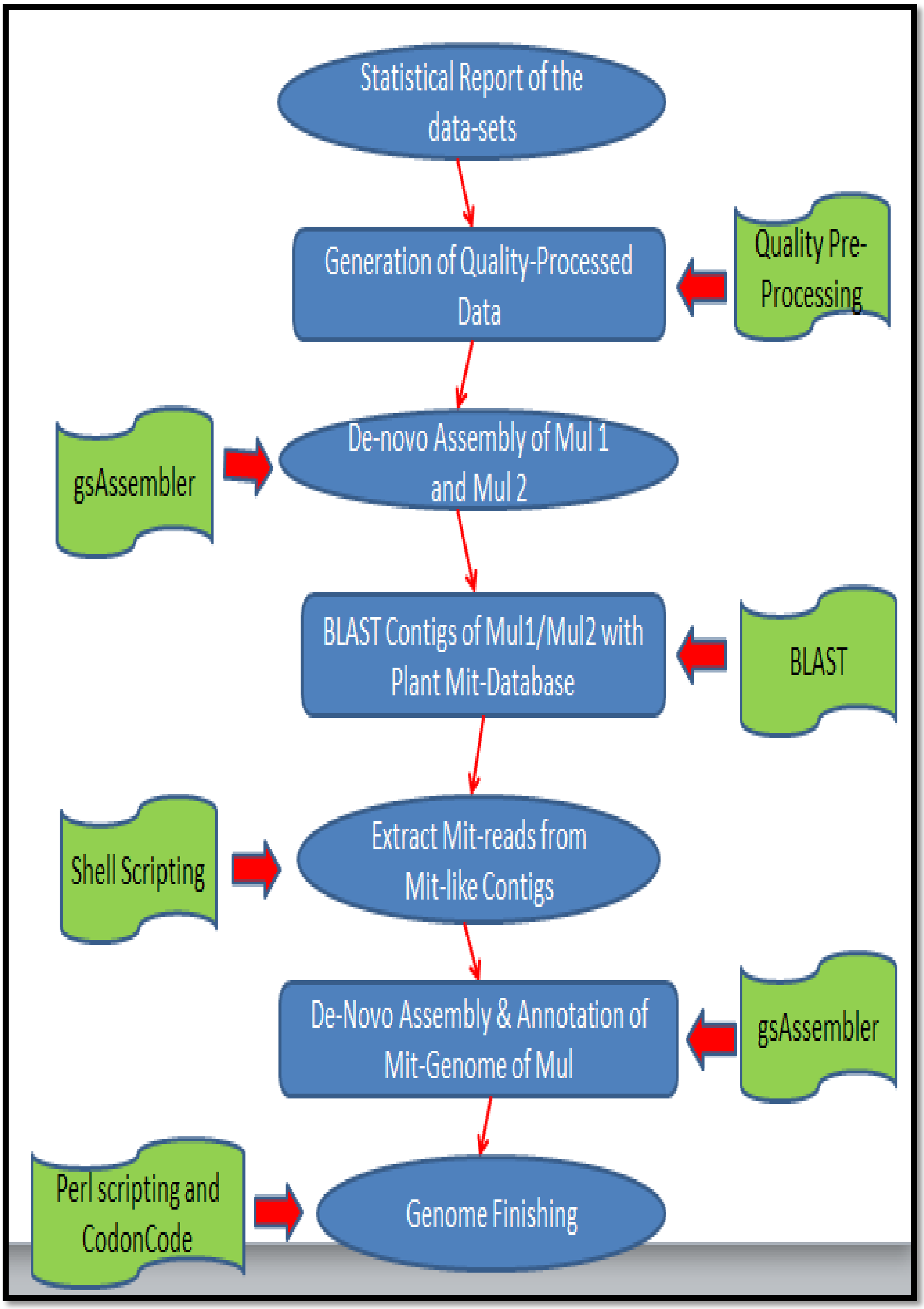

Statistical Report of the data-sets
Generation of Quality-Processed Data
Quality Pre-Processing
gsAssembler
De-novo Assembly of Mul 1 and Mul 2
BLAST Contigs of Mul1/Mul2 with Plant Mit-Database
BLAST
Shell Scripting
Extract Mit-reads from Mit-like Contigs
De-Novo Assembly & Annotation of Mit-Genome of Mul
gsAssembler
Perl scripting and CodonCode
Genome Finishing

Figure #15 Methodology of Mulberry Mitochondrial Genome Assembly and Annotation.

CHAPTER #5

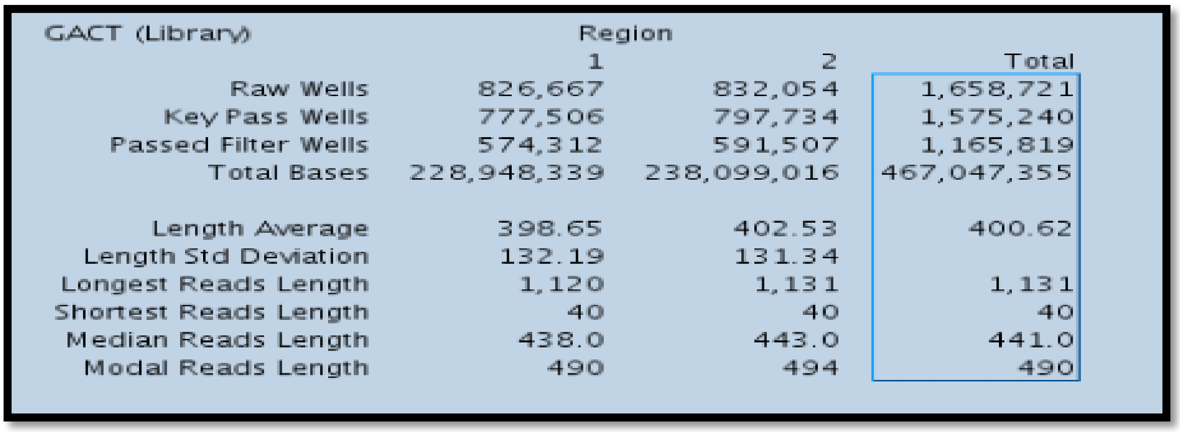
RESULTS AND DISCUSSIONS

5.1 Pre-Processing of raw 454 reads of Mulberry_parent_1 and Mulberry_Parent_2 obtained by 454 Roche pyro-sequencing run

Quality Control of NGS reads is essential to ensure that the assembled data is free from sequence artifacts and sequence contamination that may lead to erroneous downstream results. The easiest way to look at quality of the raw data is to generate the summary statistics of the data. The statistical report of the both parents of mulberry was generated using PRINSEQ and FASTQC. The input file used to generate summary statistics was FASTQ format (a file containing FASTA sequences-text format and Phred Quality score of raw reads) (Cock *et al*, 2009) which was generated using Newbler's inbuilt sffinfo command.

Statistical Report of Standard Flow-gram Format (SFF) files:

a) Number and Length of sequences:



GACT (Library)	Region 1	2	Total
Raw Wells	826,667	832,054	1,658,721
Key Pass Wells	777,506	797,734	1,575,240
Passed Filter Wells	574,312	591,507	1,165,819
Total Bases	228,948,339	238,099,016	467,047,355
Length Average	398.65	402.53	400.62
Length Std Deviation	132.19	131.34	
Longest Reads Length	1,120	1,131	1,131
Shortest Reads Length	40	40	40
Median Reads Length	438.0	443.0	441.0
Modal Reads Length	490	494	490

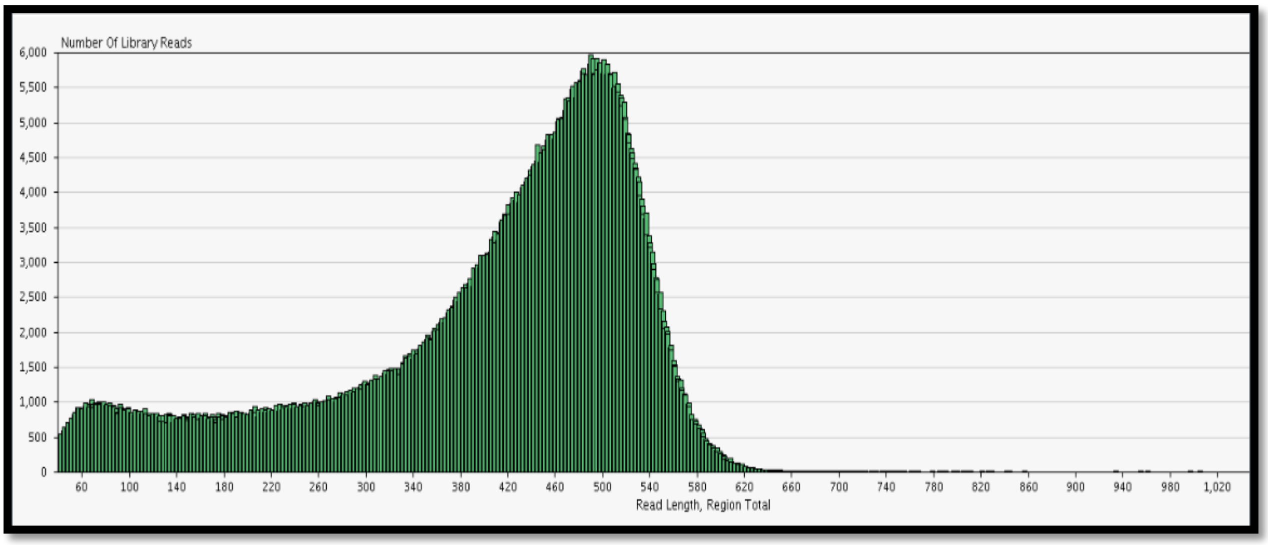
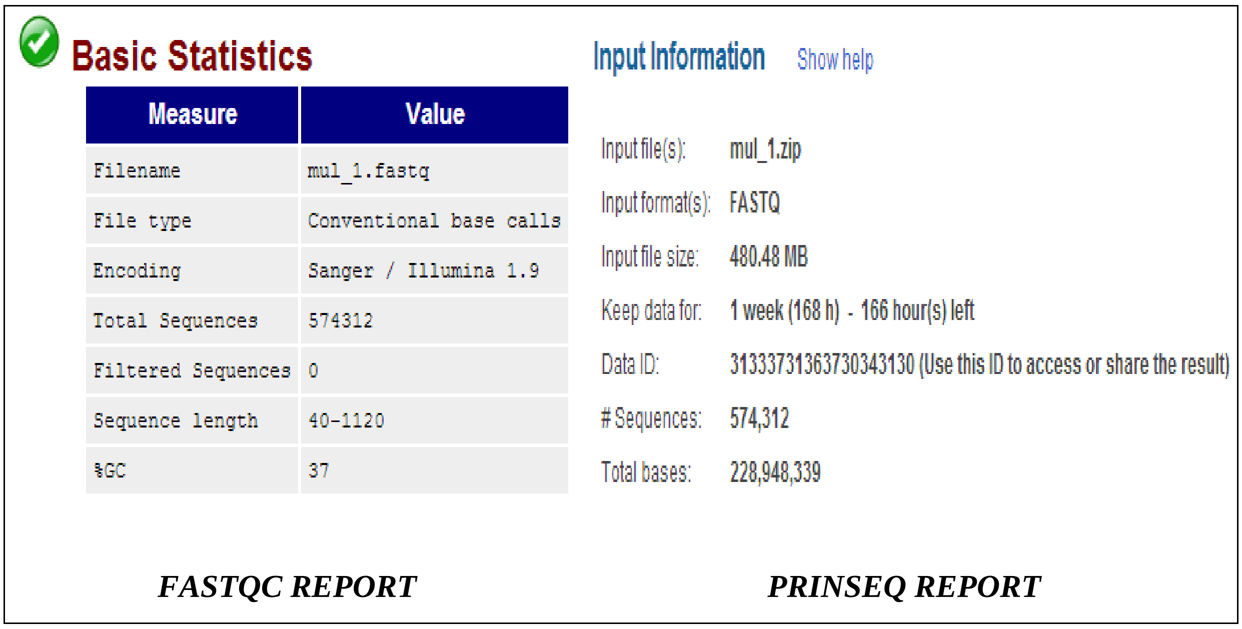


Figure #16 and #17 shows the number and length distribution of reads as generated by gsRunBrowser (Newbler). Region 1 is parent 1 of mulberry mapping population and region 2 is parent 2 of mulberry mapping population.



Basic Statistics

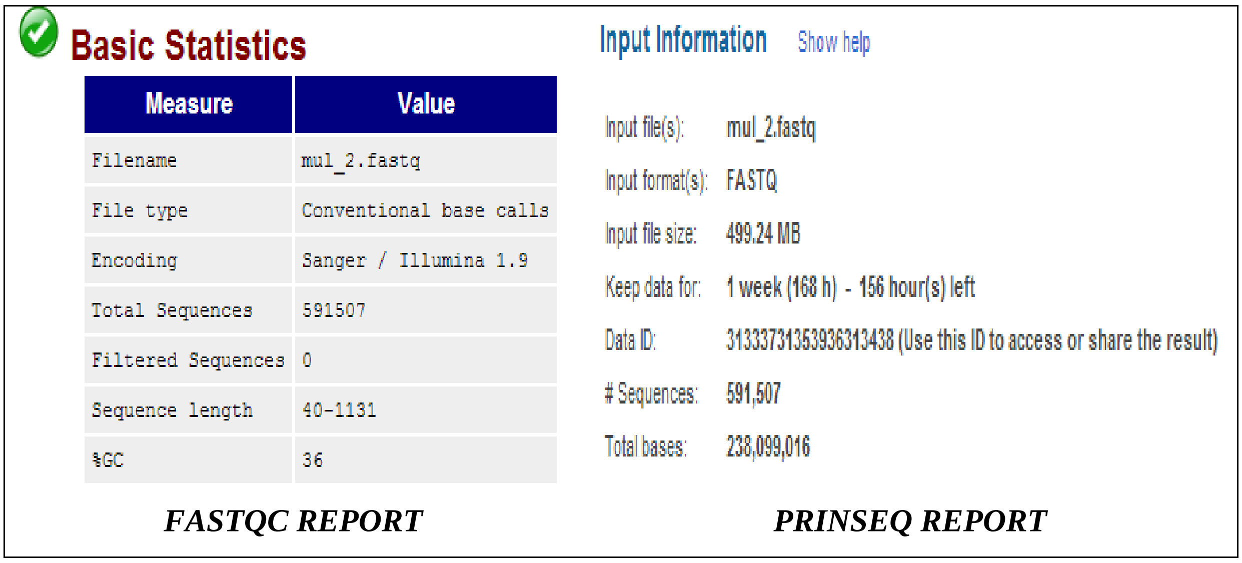
Measure	Value
Filename	mul_1.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	574312
Filtered Sequences	0
Sequence length	40-1120
%GC	37

Input Information Show help

Input file(s):	mul_1.zip
Input format(s):	FASTQ
Input file size:	480.48 MB
Keep data for:	1 week (168 h) - 166 hour(s) left
Data ID:	31333731363730343130 (Use this ID to access or share the result)
# Sequences:	574,312
Total bases:	228,948,339

FASTQC REPORT *PRINSEQ REPORT*

Figure #18 Basic Statistics of Mulberry_Parent_1 raw sff file.



Basic Statistics

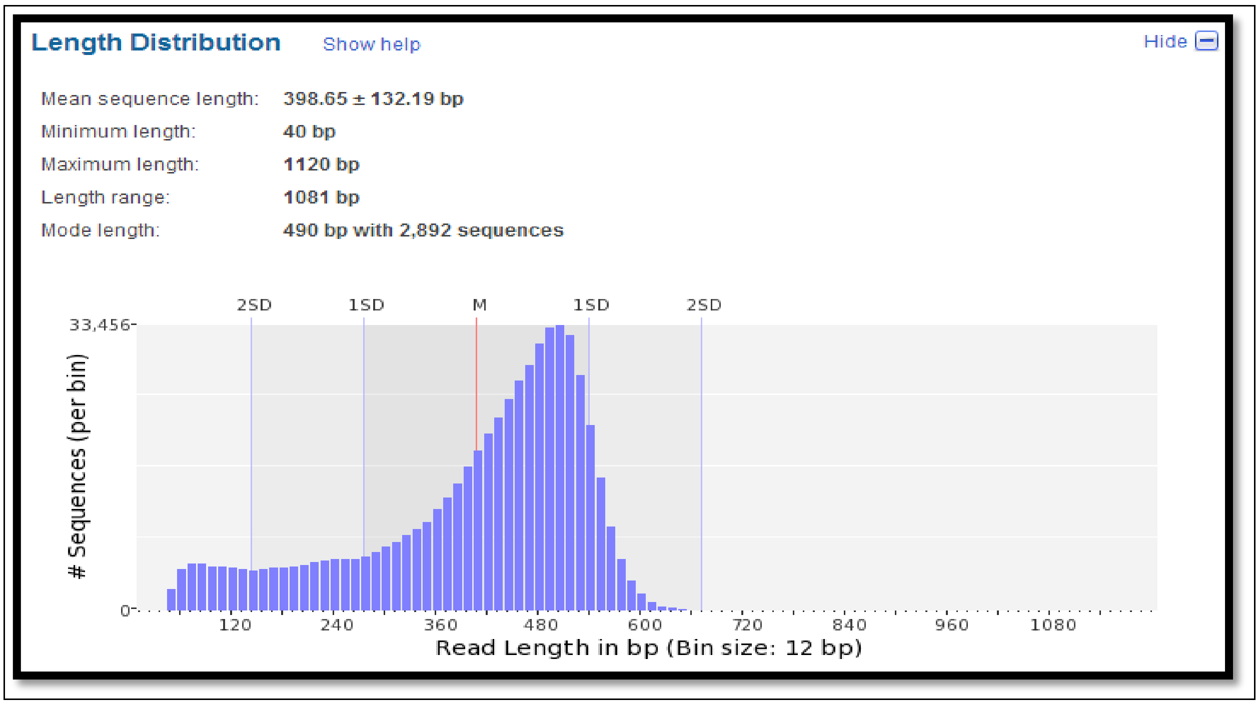
Measure	Value
Filename	mul_2.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	591507
Filtered Sequences	0
Sequence length	40-1131
%GC	36

Input Information Show help

Input file(s):	mul_2.fastq
Input format(s):	FASTQ
Input file size:	499.24 MB
Keep data for:	1 week (168 h) - 156 hour(s) left
Data ID:	31333731353936313438 (Use this ID to access or share the result)
# Sequences:	591,507
Total bases:	238,099,016

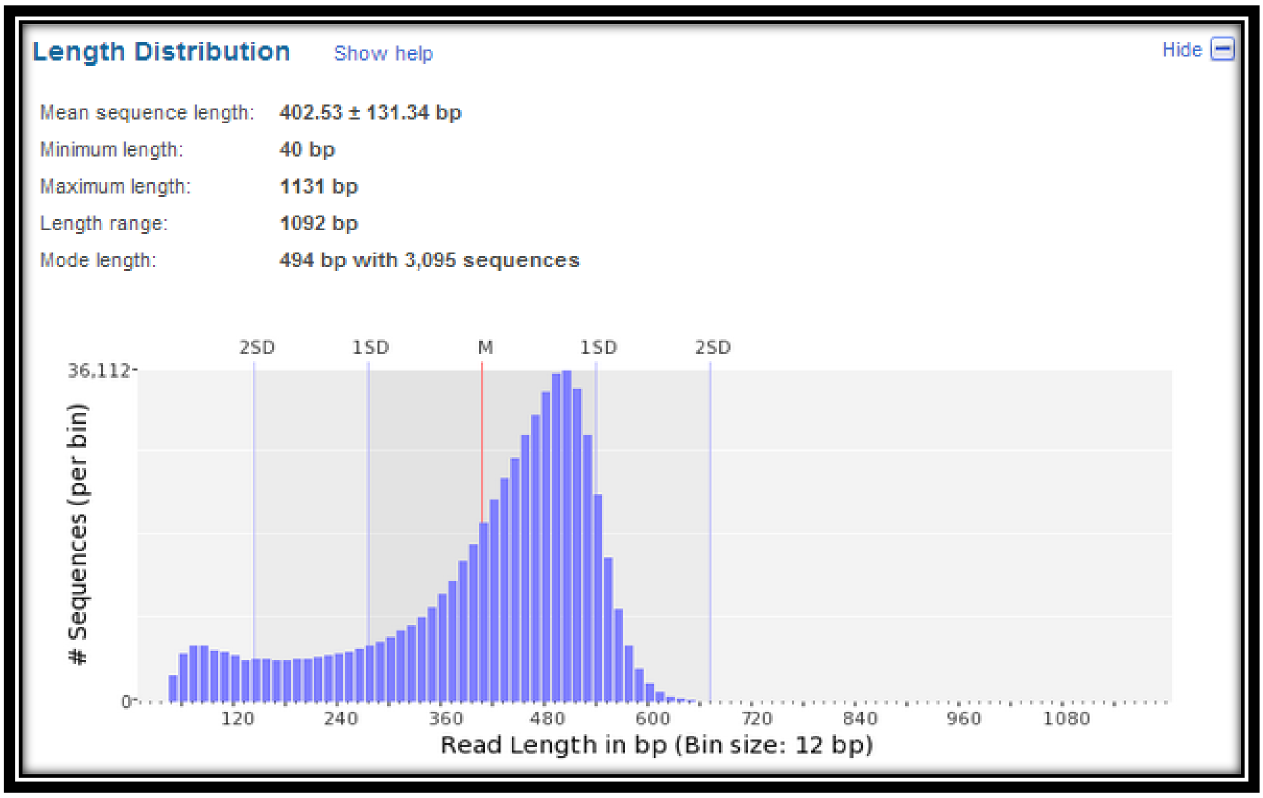
FASTQC REPORT *PRINSEQ REPORT*

Figure #19 Basic Statistics of Mulberry_Parent_2 raw sff file.


Length Distribution Show help Hide

Mean sequence length: 398.65 ± 132.19 bp
Minimum length: 40 bp
Maximum length: 1120 bp
Length range: 1081 bp
Mode length: 490 bp with 2,892 sequences

Figure #20 showing Length Distribution of mulberry_parent_1 raw reads generated by 454 Roche Pyro-sequencing NGS run.

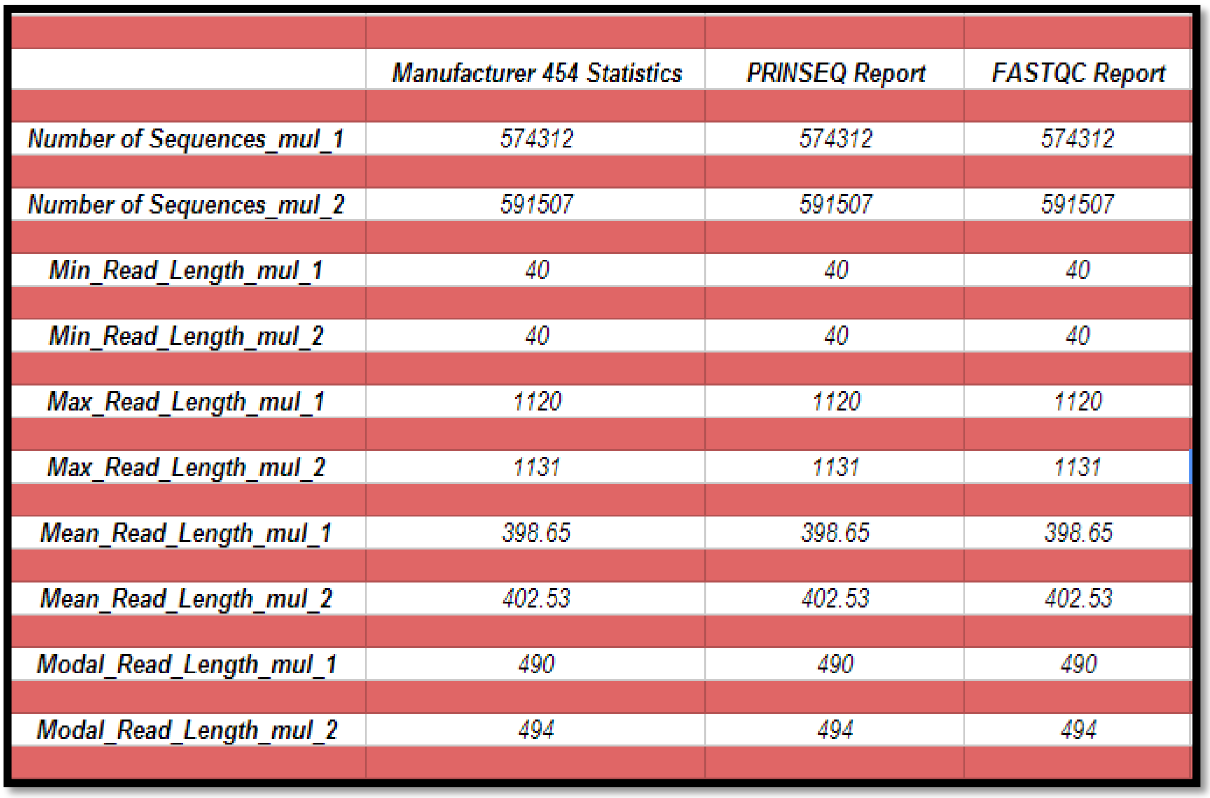

Length Distribution Show help Hide

Mean sequence length: 402.53 ± 131.34 bp
Minimum length: 40 bp
Maximum length: 1131 bp
Length range: 1092 bp
Mode length: 494 bp with 3,095 sequences

Figure #21 showing Length Distribution of mulberry_parent_2 raw reads generated by 454 Roche Pyro-sequencing NGS run.

The graphs of the Length-Distribution are generated by binning graph method. Binning means grouping so if a particular character say length of entities (here reads generated from a 454 run) is plotted on x axis, then a bin size of 50 means grouping the length of reads in bins or groups of 50 say 50-100,100-150 and so on. On the y axis, the other attribute say number of sequences would be plotted.

The FASTQC and PRINSEQ report for the number and length of sequences generated by 454 run for both mulberry_parent_1 and mulberry_parent_2 is in accordance with the manufacturer statistics for the 454 run.



	Manufacturer 454 Statistics	PRINSEQ Report	FASTQC Report
Number of Sequences_mul_1	574312	574312	574312
Number of Sequences_mul_2	591507	591507	591507
Min_Read_Length_mul_1	40	40	40
Min_Read_Length_mul_2	40	40	40
Max_Read_Length_mul_1	1120	1120	1120
Max_Read_Length_mul_2	1131	1131	1131
Mean_Read_Length_mul_1	398.65	398.65	398.65
Mean_Read_Length_mul_2	402.53	402.53	402.53
Modal_Read_Length_mul_1	490	490	490
Modal_Read_Length_mul_2	494	494	494

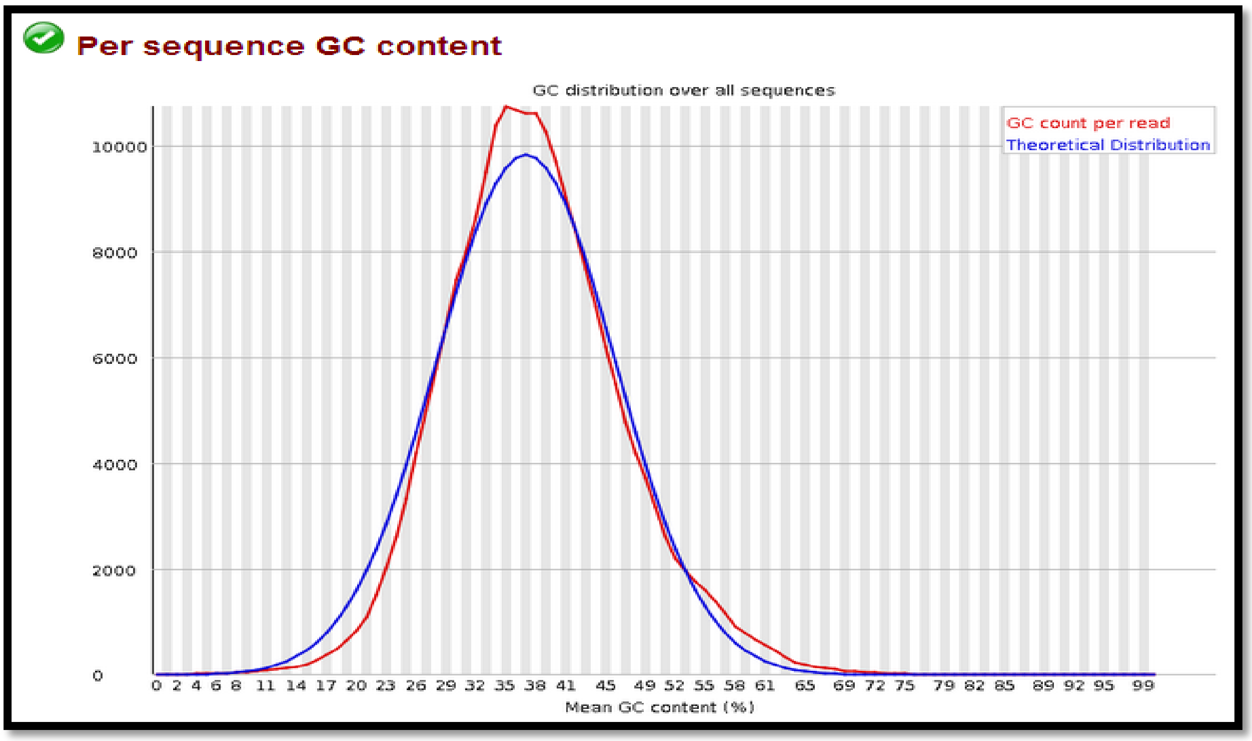
Table #3 Summary of number and length distribution of the data generated by manufacturer 454BaseCalling statistics, PRINSEQ and FASTQC

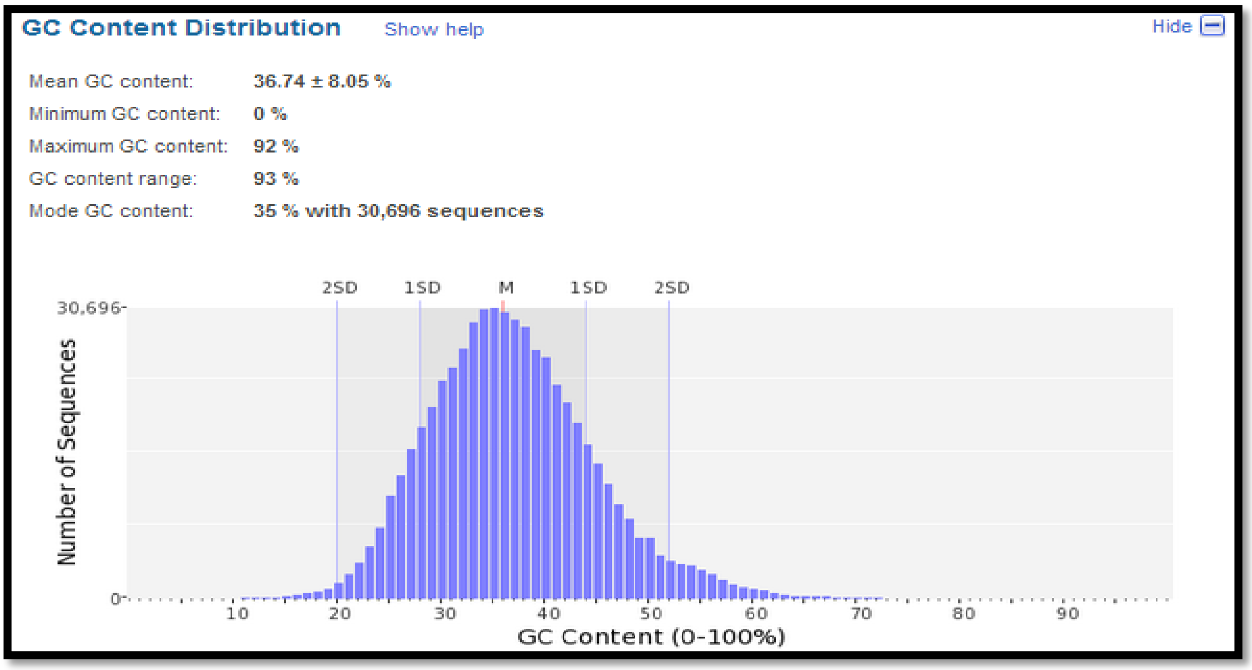
In general, during assembly process reads less than 60 bases (accounting to 20 amino acids) are discarded. The length for reads used in the assembly process should range from 60 to twice the mean length of the reads (Balzer *et al*, 2010).

b) GC content distribution

The GC content distribution in most samples should follow a normal bell-shaped distribution (Shedko *et al*, 2013).

Mulberry_Parent_1

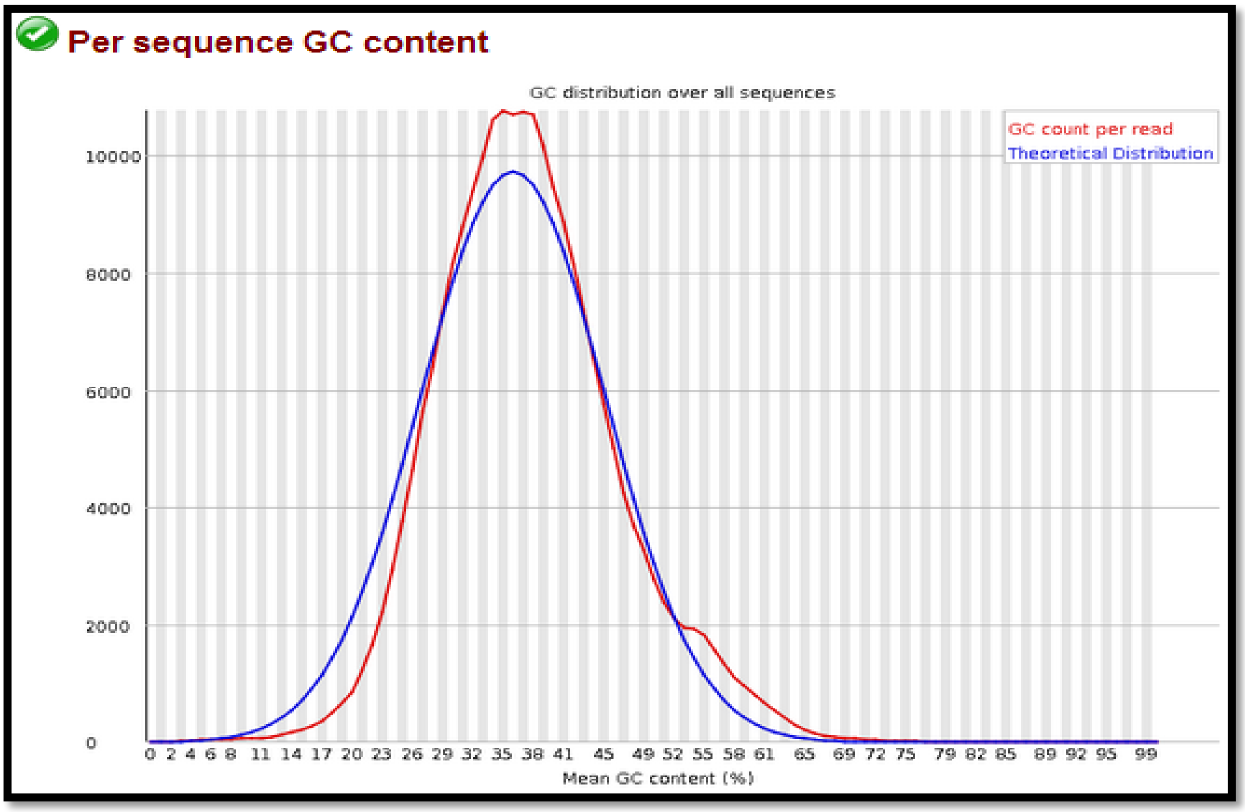


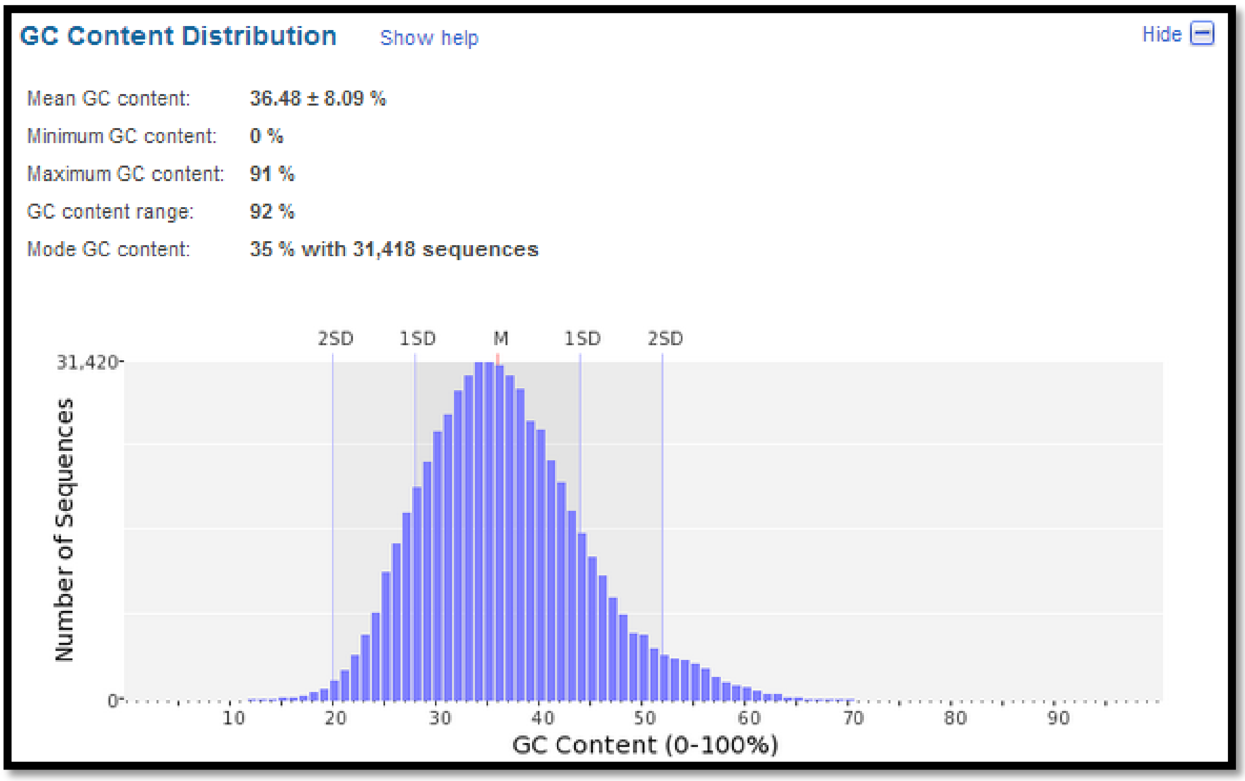

GC Content Distribution Show help Hide

Mean GC content:	36.74 ± 8.05 %
Minimum GC content:	0 %
Maximum GC content:	92 %
GC content range:	93 %
Mode GC content:	35 % with 30,696 sequences

Figure #22 shows GC distribution of reads in mulberry_parent_1 generated by FASTQC and PRINSEQ. The distribution is normal with a mean of 36.74%.

Mulberry_Parent_2





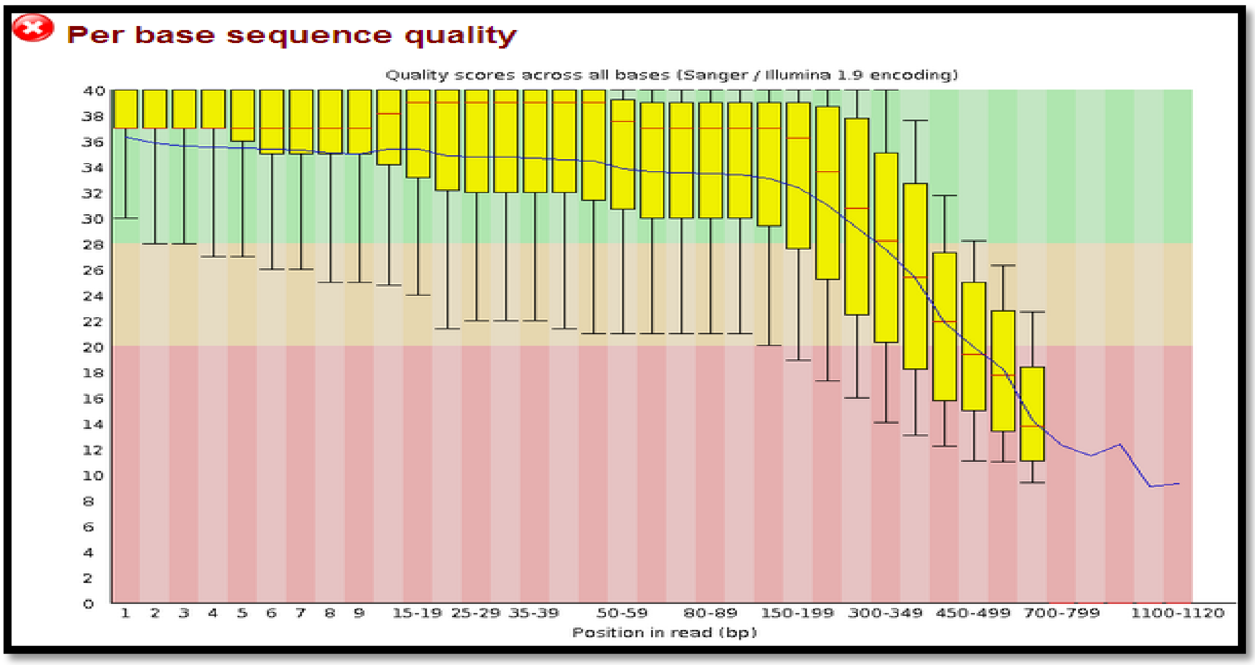
Mean GC content:	36.48 ± 8.09 %
Minimum GC content:	0 %
Maximum GC content:	91 %
GC content range:	92 %
Mode GC content:	35 % with 31,418 sequences

Figure #23 shows GC distribution of reads in mulberry_parent_2 generated by FASTQC and PRINSEQ. The distribution is normal with a mean of 36.48%.

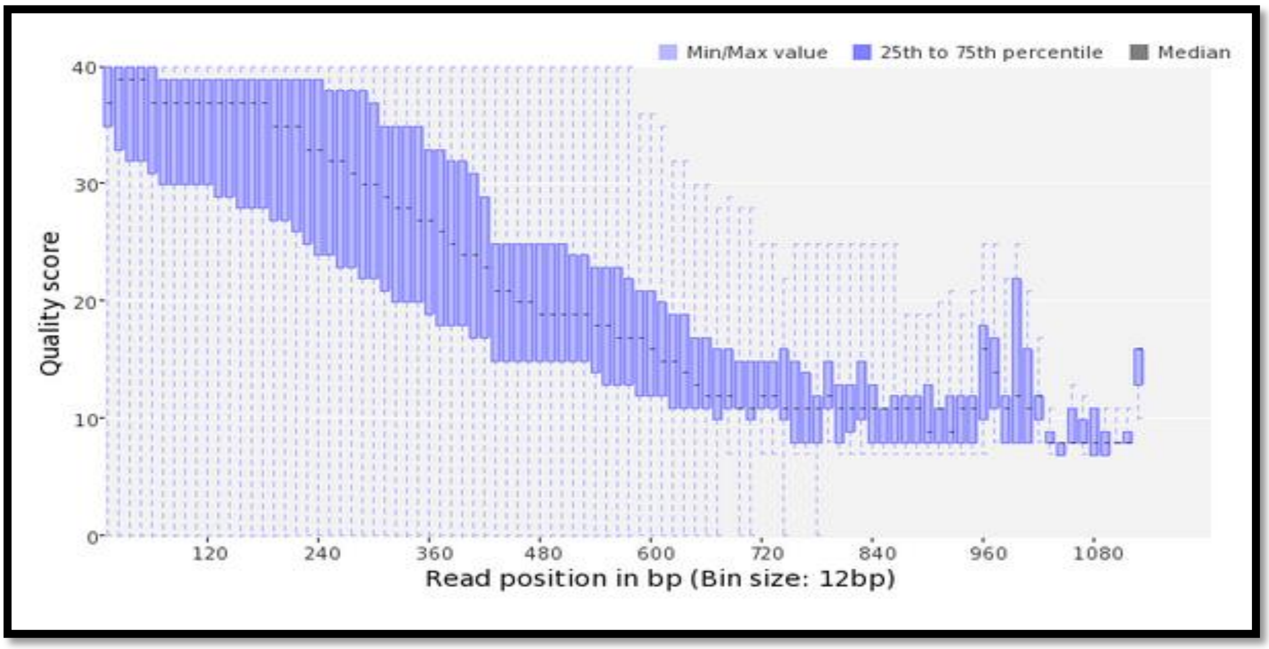
c) Base Quality Distribution

The Phred base Quality determines the quality of the base incorporated during the base call. The acceptable Phred quality threshold is from 15-25 (PRINSEQ).

Mulberry_parent_1



FASTQC per base sequence quality for mulberry_parent_1

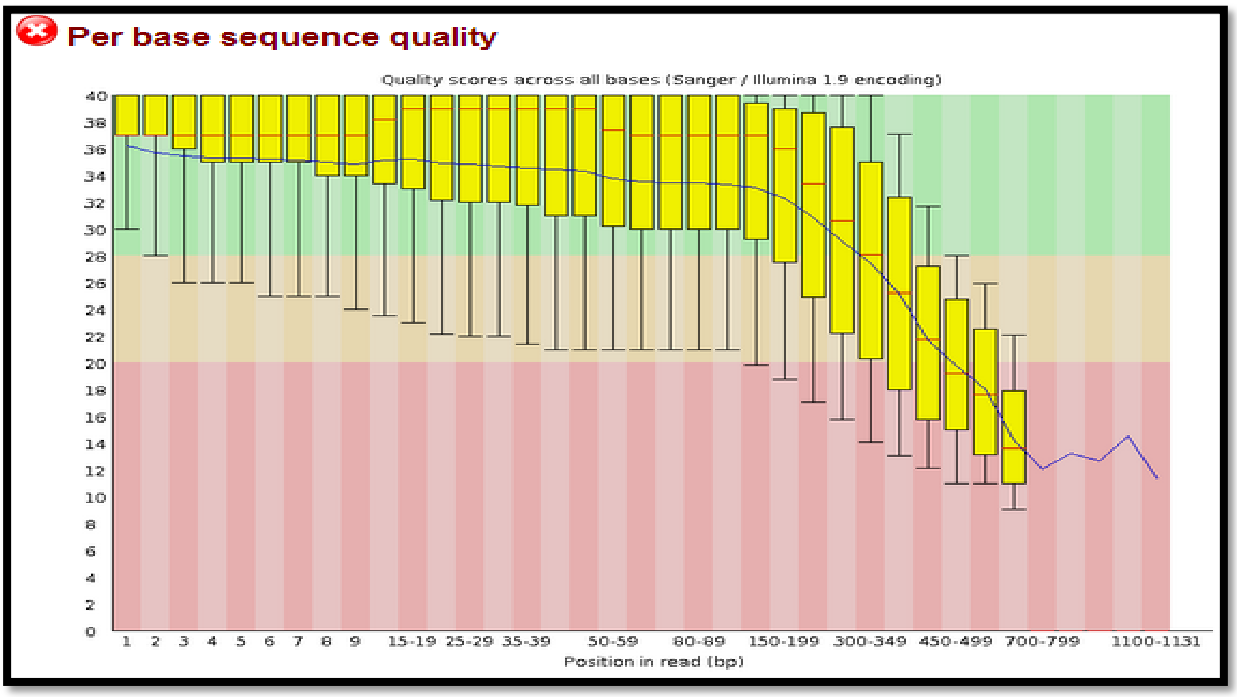


PRINSEQ per base sequence quality for mulberry_parent_1

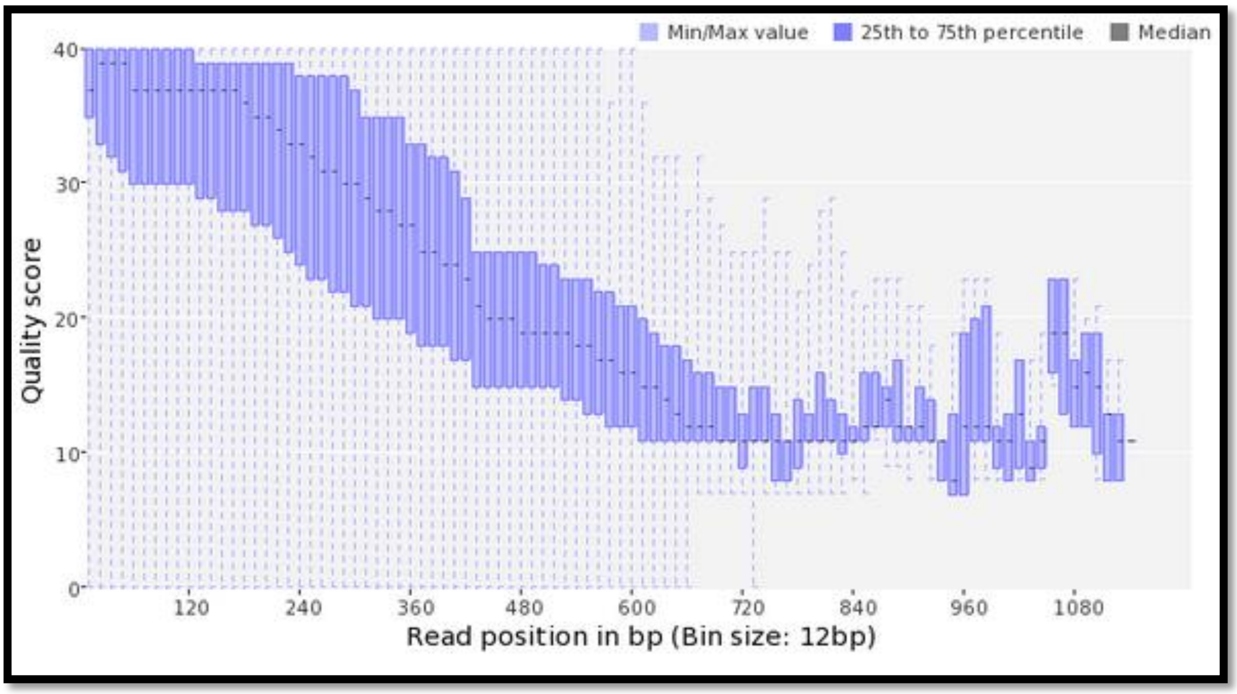
Figure #24 Base Quality Report of Mulberry_Parent_1 Raw reads

The box-whisker plot showing per base sequence quality of mulberry_parent_1 raw 454 NGS reads shows that the base quality is deteriorating in reads longer than 700 bp.

<u>Mulberry_Parent_2</u>



FASTQC per base sequence quality for mulberry_parent_2



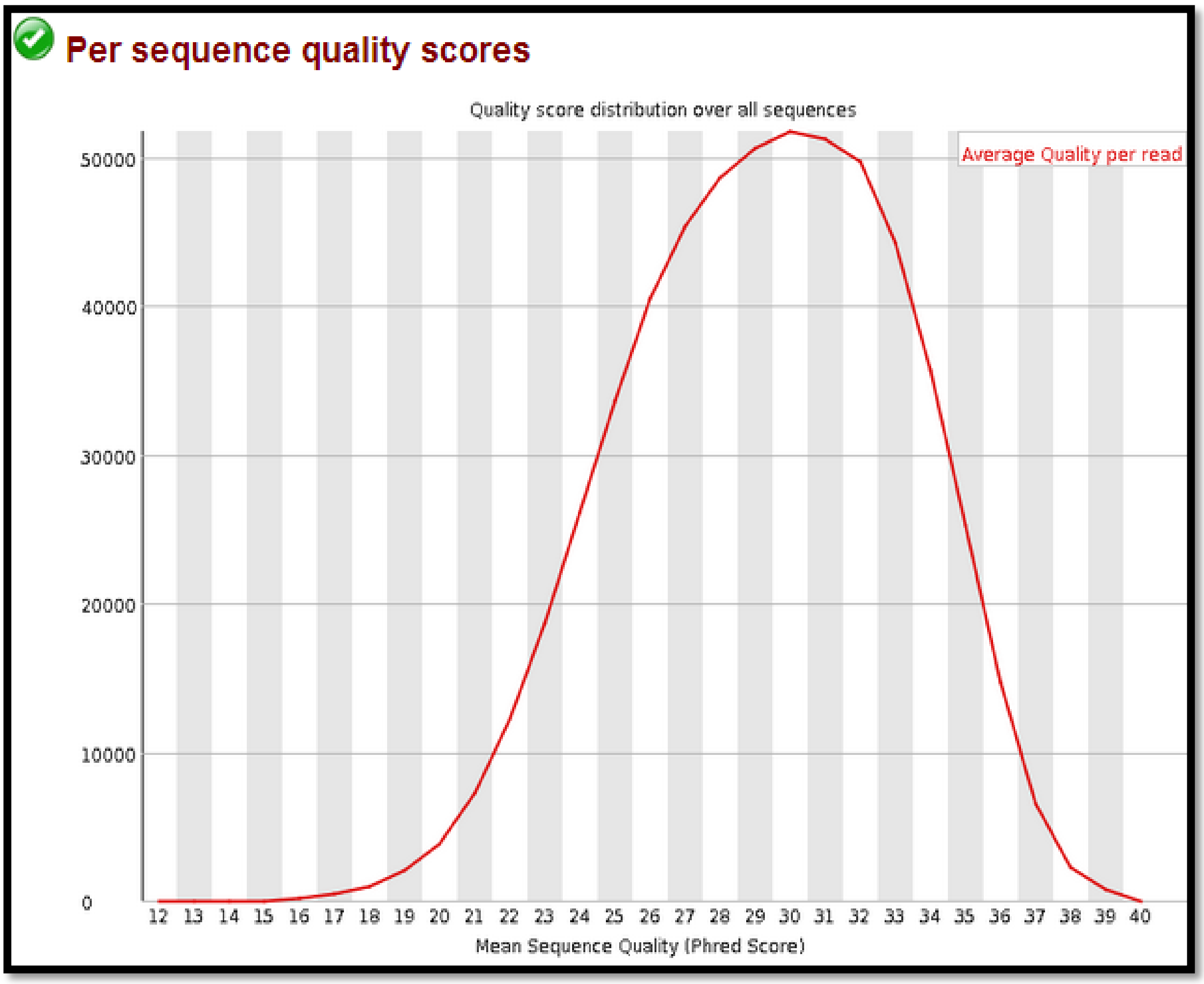
PRINSEQ per base sequence quality for mulberry_parent_2

Figure #25 Base Quality Distribution for Mulberry_Parent_2 raw reads

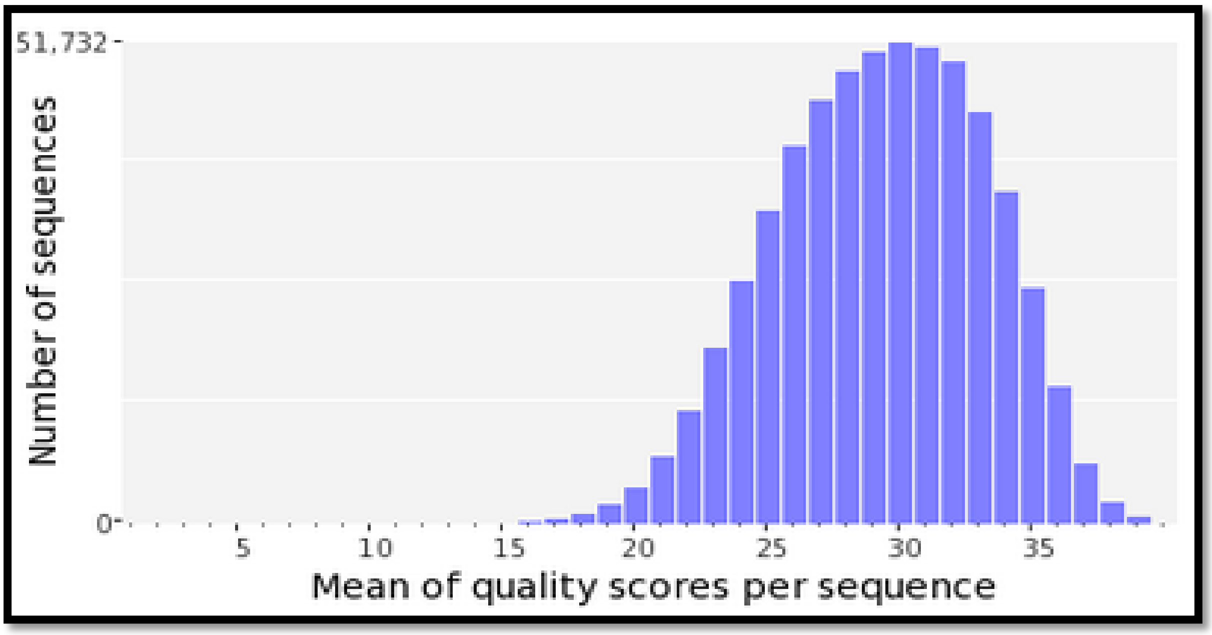
The box-whisker plot showing per base sequence quality of mulberry_parent_2 raw 454 NGS reads shows that the base quality is deteriorating in reads longer than 700 bp. The low Phred base quality in longer reads is the most common sequencing error of 454 Pyro-sequencing reactions. This is called homo-polymer error which results from the flow calls rather than base calls from the 454 run. The 454 Sequencing method doesn't calls bases rather it calls flows. Each flow represents a homo-polymer, and the brightness of the light indicates the length of the homo-polymer. Hence the sequence TAAAAA would appear as a small light to mark the T, followed by a much brighter light to mark the 5 A's. The danger in this process is that the brightness of the light is easy to mis-calibrate, especially for long homo-polymers. Thus the longer the reads, the more is the chance of low quality homo-polymer errors. The Phred base quality also deteriorates towards the 3-prime ends of the reads (Balzer, 2010). Hence, the quality trimming in 454 data should be done based upon the following criteria:

Phred Base Quality Score threshold of 15-25 towards 3-prime end of the reads.

The longer homo-polymer reads should be trimmed towards the 3-prime end only based upon the Phred Quality threshold. The trimming of complete homopolymer long reads may result in more false negatives in the downstream processing of the data.

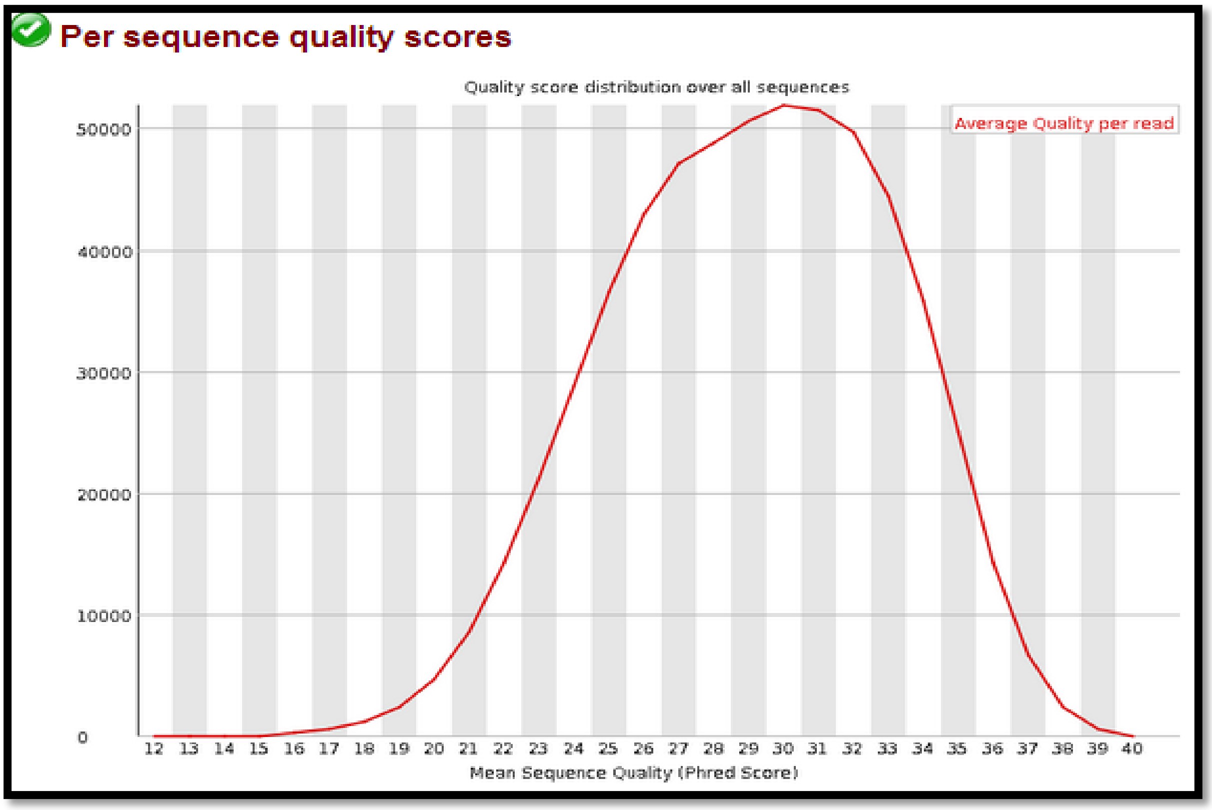


FASTQC Report of mean sequence quality of mulberry_parent_1

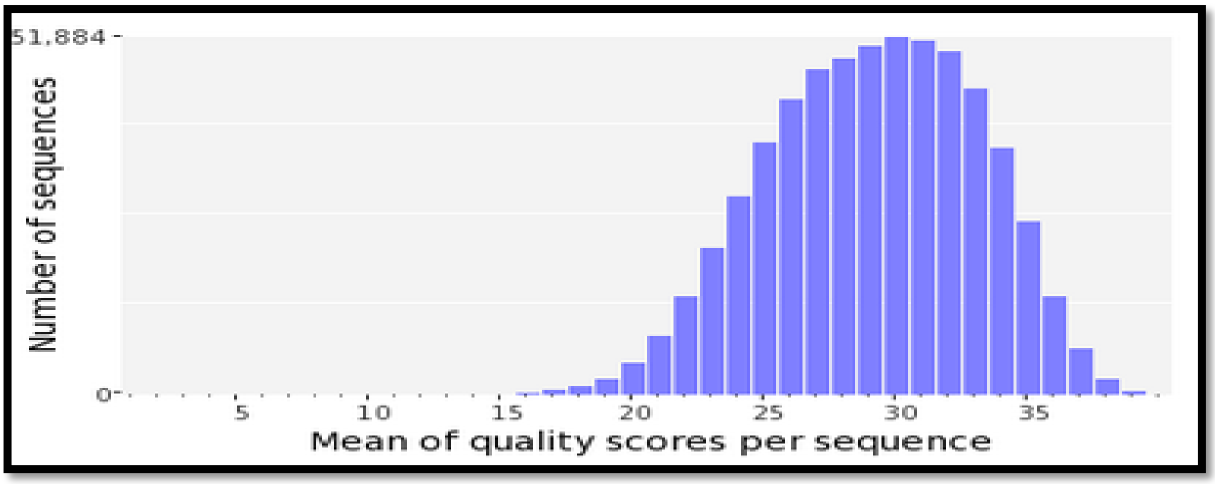


PRINSEQ Report of mean sequence quality of mulberry_parent_1

Figure #26 Mean Sequence Quality of Mulberry_Parent_1 raw reads



FASTQC Report of mean sequence quality of mulberry_parent_2



PRINSEQ Report of mean sequence quality of mulberry_parent_2

Figure #27 Mean Sequence Quality of Mulberry_Parent_2 raw reads

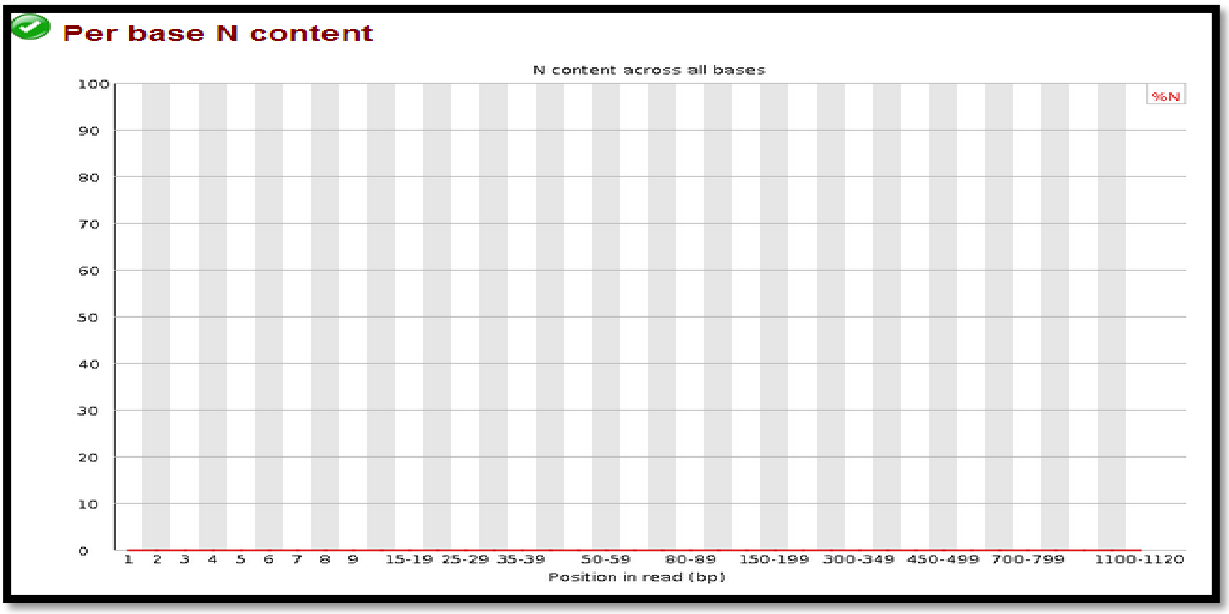
The mean sequence quality of both the parents is as follows:

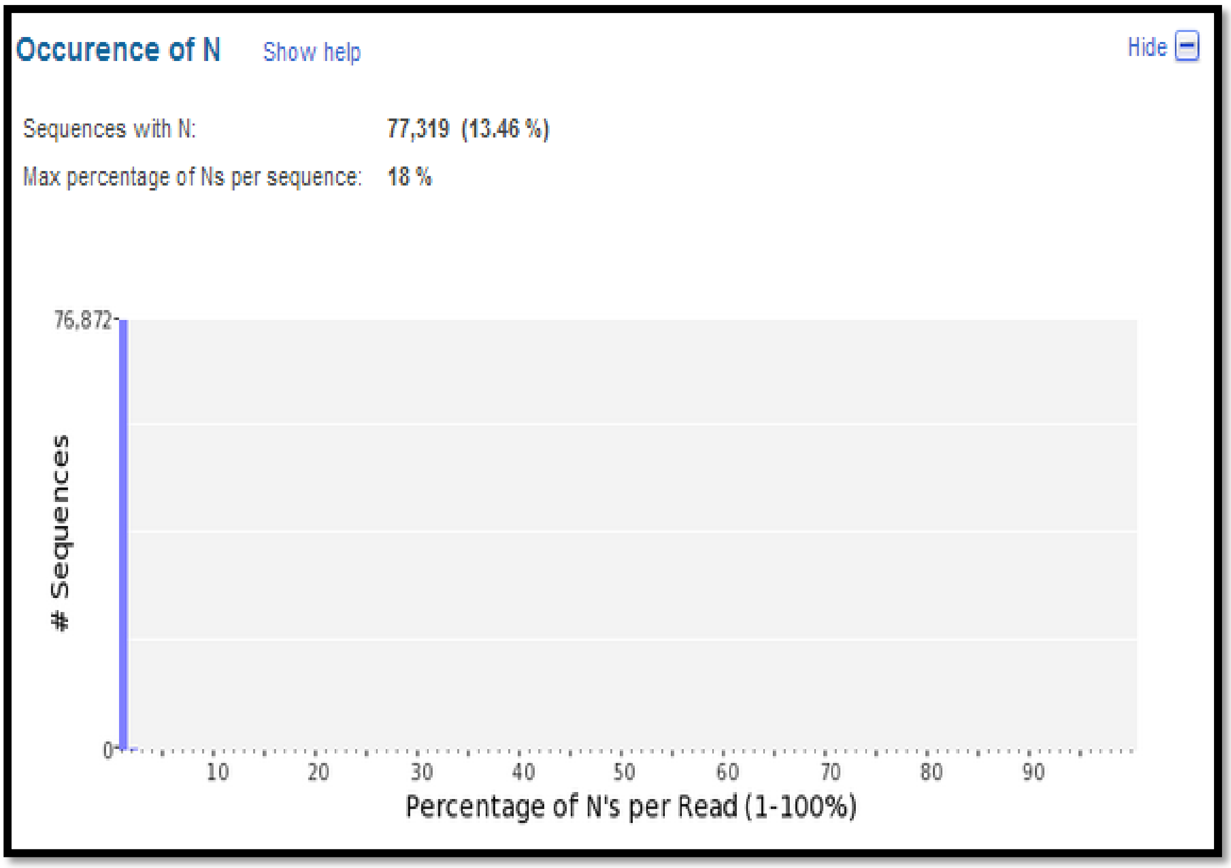
Mulberry_Parent_1 : 30

Mulberry_Parent_2 : 30

Occurrence of N:

Sequences can contain the ambiguous base N for positions that could not be identified as a particular base. A high number of Ns can be a sign for a low quality sequence or even dataset. If no quality scores are available, the sequence quality can be inferred from the percent of Ns found in a sequence or dataset. A recent study found that the presence of any ambiguous base calls was a sign for overall poor sequence quality (Huse *et al*, 2007). The amount of ambiguous bases being present in the sequences should account to just 1 %.

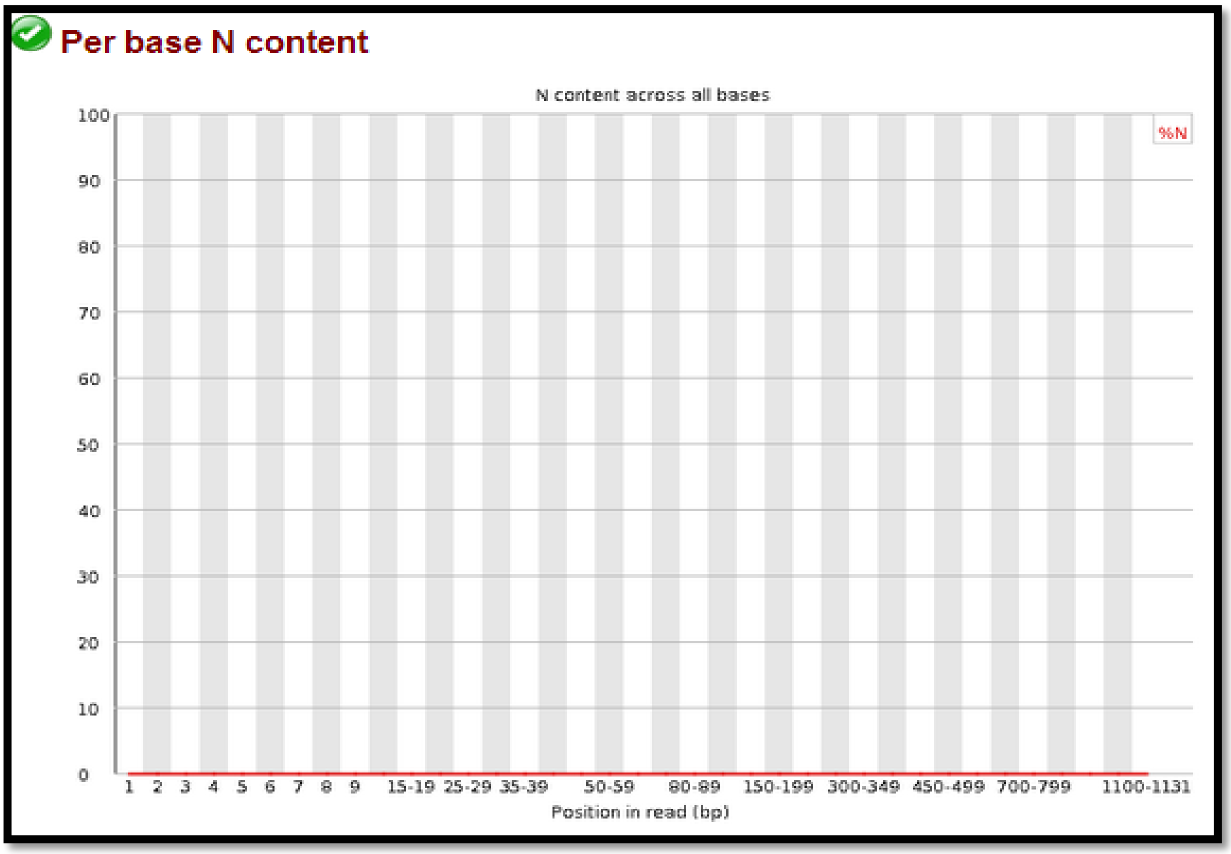


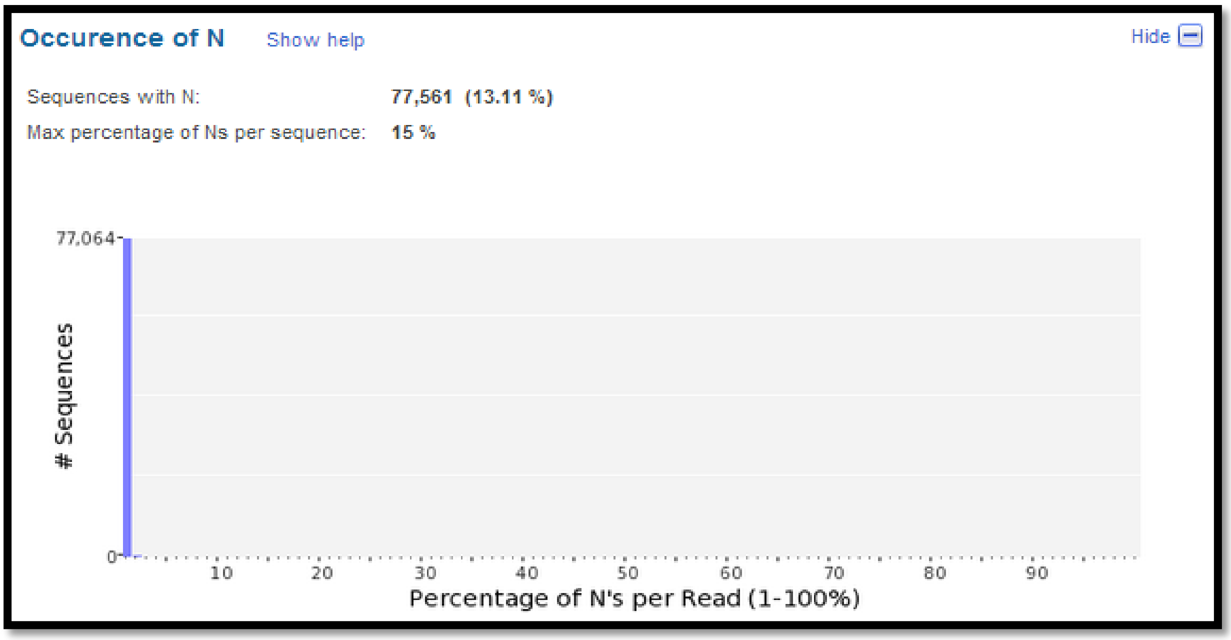


Sequences with N: 77,319 (13.46 %)

Max percentage of Ns per sequence: 18 %

Figure #28 FATSQC and PRINSEQ Report for ambiguous bases in mulberry_parent_1




Occurence of N Show help Hide

Sequences with N: 77,561 (13.11 %)

Max percentage of Ns per sequence: 15 %

Figure #29 FASTQC and PRINSEQ Report for ambiguous bases in mulberry_parent_2

The occurrence of sequences with 'N' in parent_1 is 18% and in parent_2 is 15%.

d) **Poly A/T tails**:

Poly-A/T tails are considered repeats of As or Ts with a minimum length of 5 bp. Sequences that contain only As or Ts are counted for both ends. These repeats can bind to low complexity regions in database searches or can with regions having stretches of Poly A/T tails (Huse *et al*, 2007).

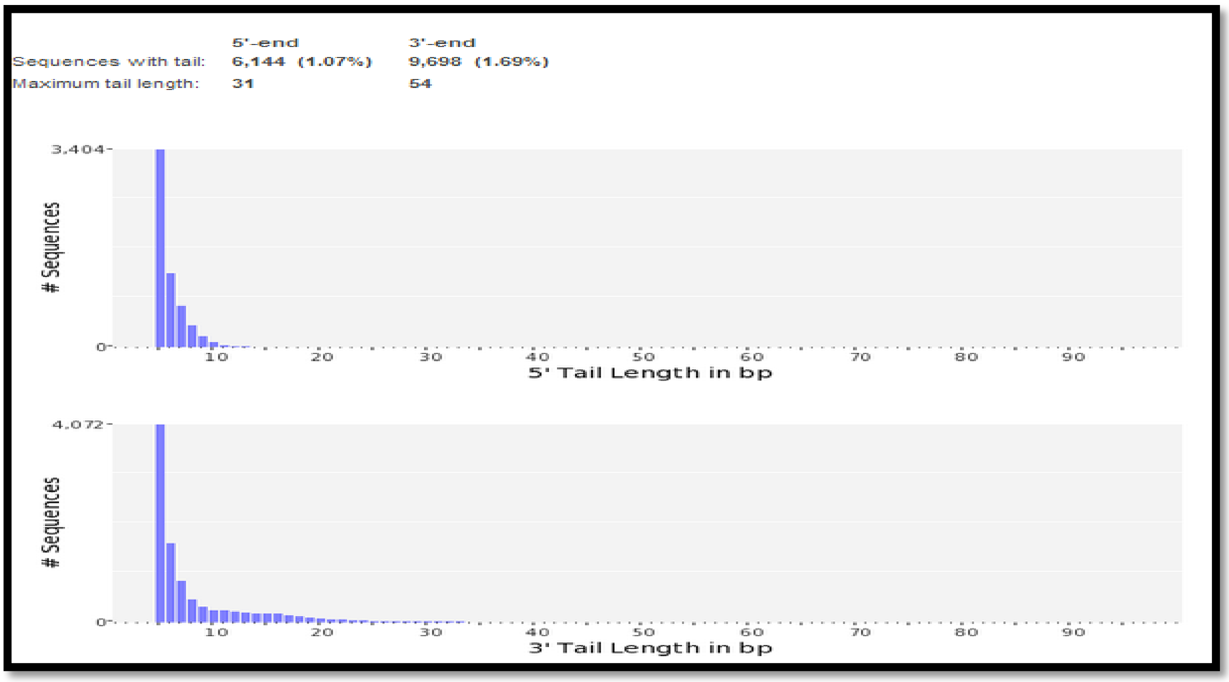

	5'-end	3'-end
Sequences with tail:	6,144 (1.07%)	9,698 (1.69%)
Maximum tail length:	31	54

Figure #30 PRINSEQ Report of Poly A/T tails in mulberry_parent_1

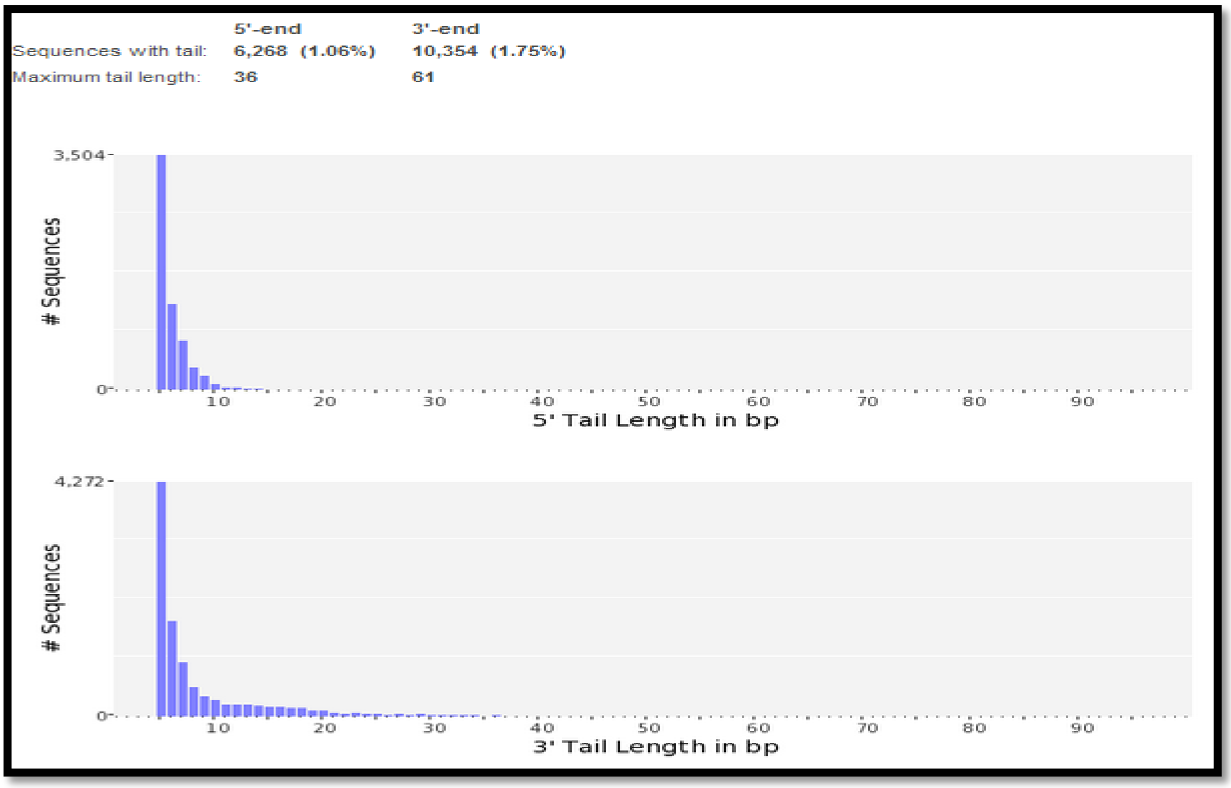
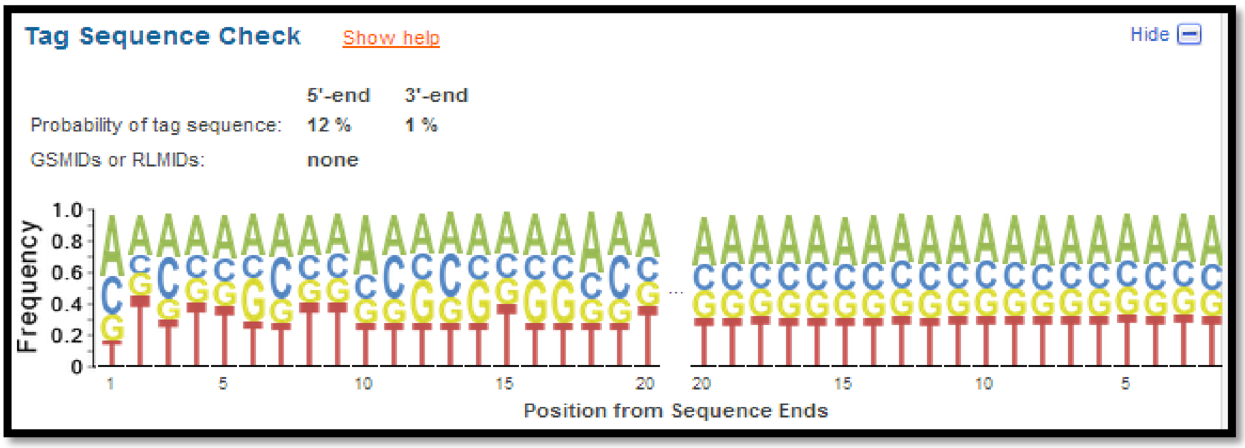

```
                          5'-end            3'-end
Sequences with tail:      6,268  (1.06%)    10,354  (1.75%)
Maximum tail length:      36                61
```

Figure #31 PRINSEQ Report of Poly A/T tails in mulberry_parent_2

PRINSEQ reports for both parents shows that Poly A/T tails for most of the sequences are 5bp long.

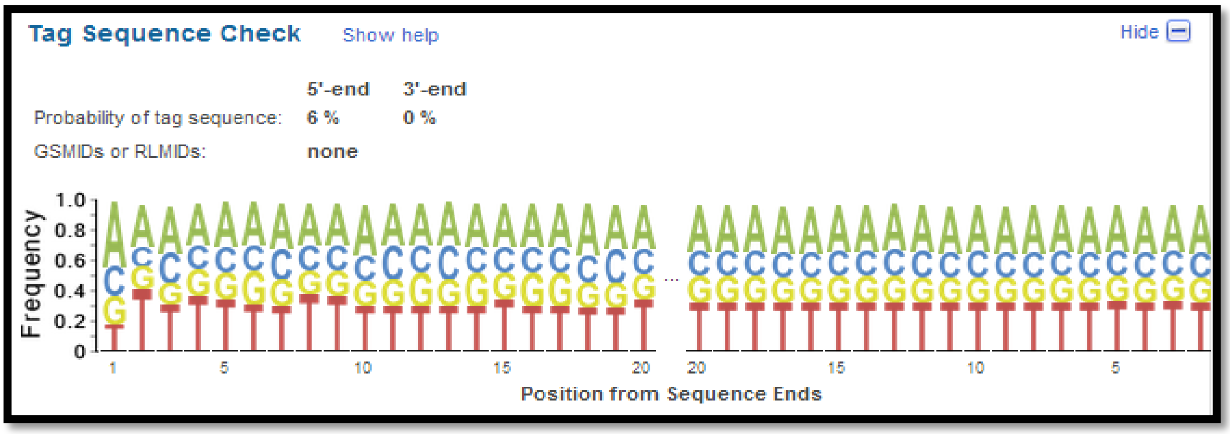
e) **Tag Sequence Check**

The tags in raw reads include the adaptors/primers used for construction of the library prepared for the sequencing run. The base frequency in a sequence determines whether a sequence is tagged or not. A uniform base frequency represents an un-tagged sequence, while a non-uniform base frequency calls for adaptor/primer/tag trimming before assembly and annotation (Huse *et al*,2007).


Tag Sequence Check

```
                               5'-end    3'-end
Probability of tag sequence:   12 %      1 %
GSMIDs or RLMIDs:              none
```

Figure #32 PRINSEQ report for tag sequence check in mulberry_parent_1

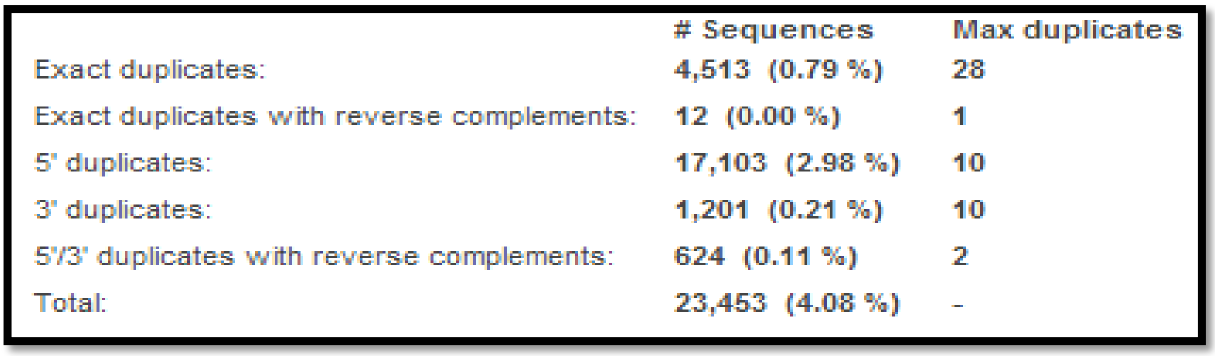

Tag Sequence Check Show help Hide

	5'-end	3'-end
Probability of tag sequence:	6 %	0 %
GSMIDs or RLMIDs:	none	

Figure #33 PRINSEQ report for tag sequence check in mulberry_parent_2

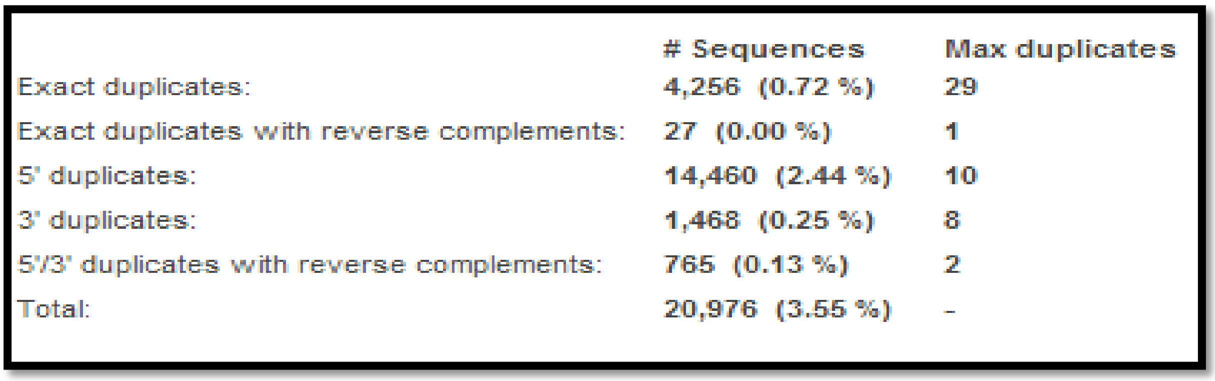
The frequency v/s position graph is skewed towards the start of sequences and is nearly uniform towards the end. Hence, the reports suggest the presence of tags/adaptors/primers at the 5-prime ends of certain percentage of reads in both the parents.

f) **Sequence Duplication**



	# Sequences	Max duplicates
Exact duplicates:	4,513 (0.79 %)	28
Exact duplicates with reverse complements:	12 (0.00 %)	1
5' duplicates:	17,103 (2.98 %)	10
3' duplicates:	1,201 (0.21 %)	10
5'/3' duplicates with reverse complements:	624 (0.11 %)	2
Total:	23,453 (4.08 %)	-

Figure #34 PRINSEQ Report for sequence duplication levels in Mulberry_Parent_1



	# Sequences	Max duplicates
Exact duplicates:	4,256 (0.72 %)	29
Exact duplicates with reverse complements:	27 (0.00 %)	1
5' duplicates:	14,460 (2.44 %)	10
3' duplicates:	1,468 (0.25 %)	8
5'/3' duplicates with reverse complements:	765 (0.13 %)	2
Total:	20,976 (3.55 %)	-

Figure #35 PRINSEQ Report for sequence duplication levels in Mulberry_Parent_2

Duplicates can arise when there are too few fragments present at any stage prior to sequencing, especially during any PCR step. Furthermore, the theoretical idea of one micro-reactor containing one bead for 454/Roche sequencing does not always translate into practice where many beads can be found in a single micro-reactor. Unfortunately, artificial duplicates are difficult to distinguish from exactly overlapping reads (real dupliactes) that naturally occur within deep sequence or high coverage samples (Gomez-Alvarez *et al*, 2009).

Fortunately Newbler (454 Platform specific assembler) treats exact duplicate reads as a single read before assembly, if otherwise specified by changing the default settings. This is true for many assemblers. Further, one needs to be cautious while working with data-sets having high sequence duplication levels. Assembling reads without duplicate reads removal might lead to false coverage values.

g) Sequence Complexity:

Genome sequences can exhibit intervals with low-complexity, which may be part of your sequence dataset when using random sampling techniques. Low-complexity sequences are defined as having commonly found stretches of nucleotides with limited information content (e.g. the dinucleotide repeat CACACACACA). Such sequences can produce a large number of high-scoring but biologically insignificant results in database searches. The complexity of a sequence can be estimated using many different approaches. The charts below are generated using the DUST and Entropy approaches as they present two commonly used examples.

The DUST approach is adapted from the algorithm used to mask low-complexity regions during BLAST search preprocessing. The scores are computed based on how often different trinucleotides occur and are scaled from 0 to 100. Higher scores imply lower complexity and complexity scores above 7 may be considered low-complexity. A sequence of homopolymer repeats (e.g. TTTTTTTTT) has a score of 100, of dinucleotide repeats (e.g. TATATATATA) has a score around 49, and of trinucleotide repeats (e.g. TAGTAGTAGTAG) has a score of around 32.

The Entropy approach evaluates the entropy of trinucleotides in a sequence. The entropy values are scaled from 0 to 100 and lower entropy values imply lower complexity. A sequence of homopolymer repeats (e.g. TTTTTTTTT) has an entropy value of 0, of dinucleotide repeats (e.g. TATATATATA) has a value around 16, and of trinucleotide repeats (e.g. TAGTAGTAGTAG) has a value around 26. Sequences with an entropy value below 70 may be considered low-complexity (Balzer *et al*, 2010).

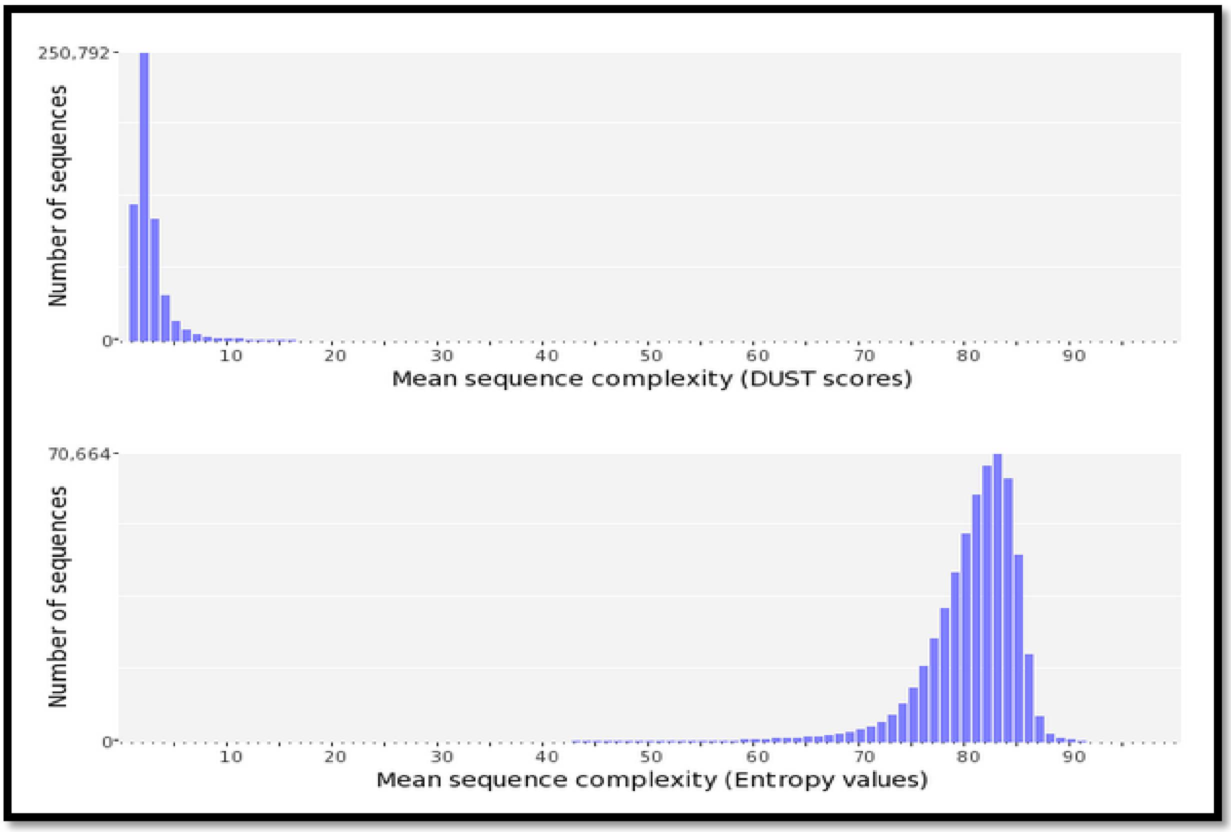


Figure #36 PRINSEQ Report for sequence complexity in mulberry_parent_1

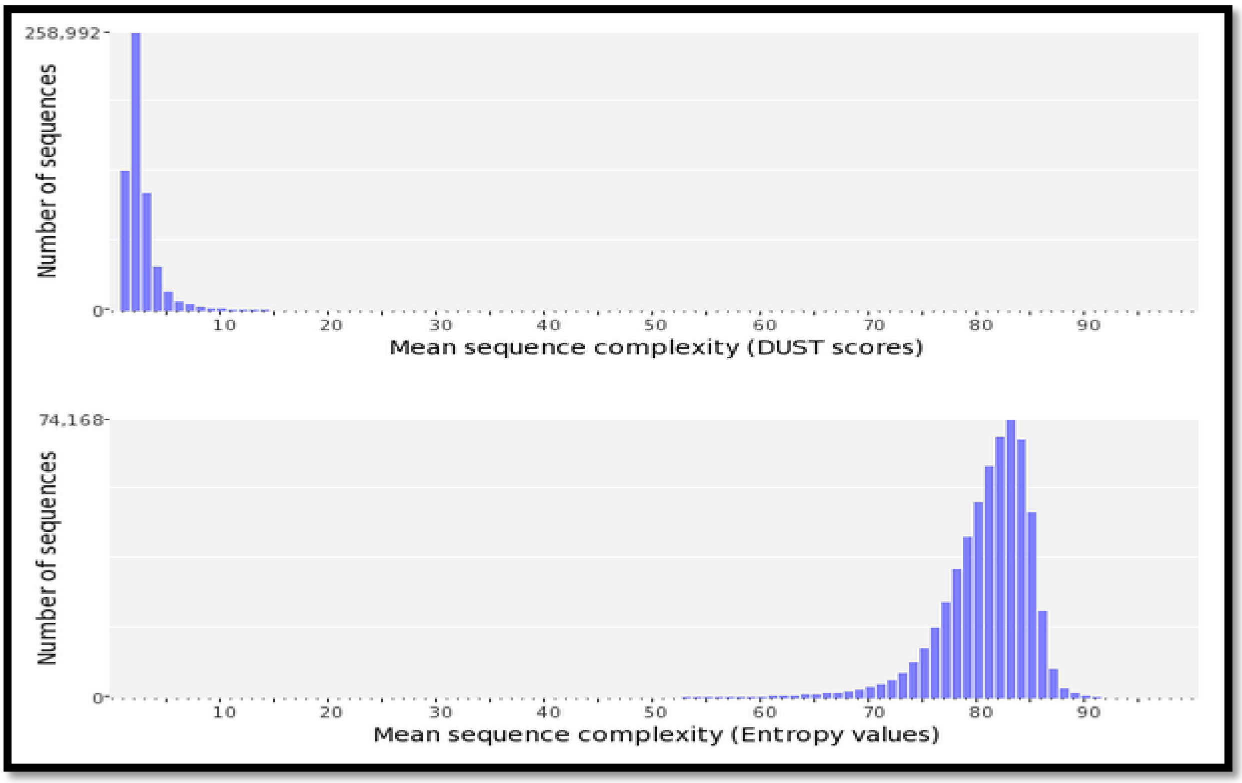


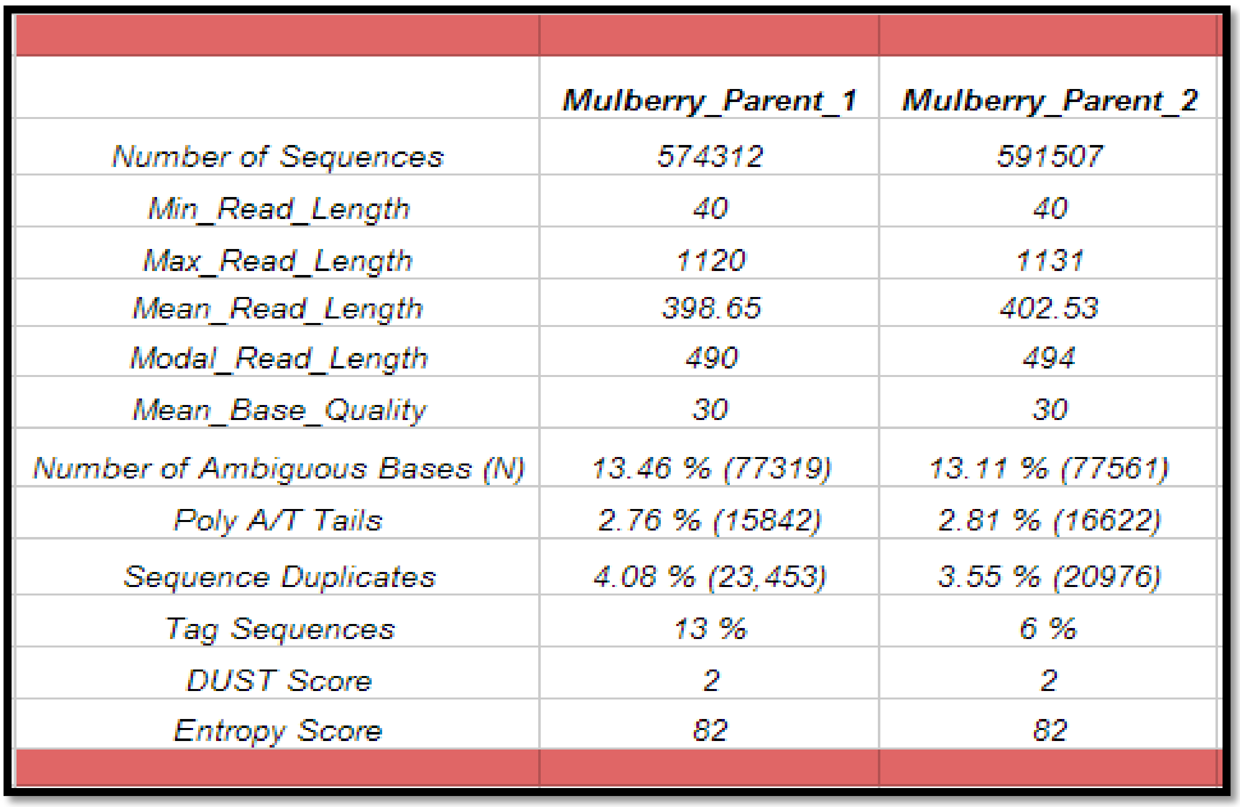
Figure #37 PRINSEQ Report for sequence complexity in mulberry_parent_2

The Statistics obtained from PRINSEQ for sequence complexity are as follows:

- DUST Score for parent_1 : 2 (acceptable)
- DUST Score for parent_2 : 2 (acceptable)
- Entropy for parent_1 : 82 (acceptable)
- Entropy for parent_2 : 82 (acceptable)

This means that the sequences from NGS sequencing run are free from LCRs.

Table #4 shows the overall statistics of the data of mulberry_parent_1 and mulberry_parent_2



	Mulberry_Parent_1	Mulberry_Parent_2
Number of Sequences	574312	591507
Min_Read_Length	40	40
Max_Read_Length	1120	1131
Mean_Read_Length	398.65	402.53
Modal_Read_Length	490	494
Mean_Base_Quality	30	30
Number of Ambiguous Bases (N)	13.46 % (77319)	13.11 % (77561)
Poly A/T Tails	2.76 % (15842)	2.81 % (16622)
Sequence Duplicates	4.08 % (23,453)	3.55 % (20976)
Tag Sequences	13 %	6 %
DUST Score	2	2
Entropy Score	82	82

The Quality Pre-Processing of the data-sets of both the parents should be done based upon the statistical summary of the data-sets as represented in table # 4.

Based upon the statistical measures the parameters for quality pre-processing are as follows:

- Filter_by_Quality: The reads with quality threshold less than 15 should be filtered out. This was done using PRINSEQ and a perl script by SeqCrumbs – filter_by_quality.pl. The quality trimming was mostly done for longer reads especially at 3-prime ends. This was done to ensure mitigation of homo-polymer errors.
- Filter_by_Length: The length of reads should fall between 60 to more than twice the mean read length. Trimming entire reads more than twice the mean length (700) may result in loss of information, hence the maximum read length was set to 1000. For longer reads Phred base quality threshold at 3-prime end was kept as 15. PRINSEQ

and a perl script by SeqCrumbs filter_by_length.pl was used to filter sequences by length.

- Ambiguous Bases: The maximum allowed rate of N was kept as 1 %. Reads having more than 1 % of N were trimmed. This was done using PRINSEQ.
- Poly A/T Tails: A threshold of 5bp was set for the removal of both 5-prime and 3-prime Poly A/T Tails in the data-sets. This means that reads having a minimum of 5 bp repeats of As and Ts were trimmed.
- Sequence Complexity: To remove the LCRs, a DUST threshold of 7 was used. Reads falling above this threshold were discarded.
- Homo-polymer reads need to be manually checked as certain long reads might not be a result of 454 homo-polymer errors. Only those long reads with long homo-polymer strecthes at the 3-prime end should be trimmed from the start of a homo-polymer stretch to its end. For example, in a 1000 bp read with a homo-polymer stretch starting from 900 base position in the read, the bases of only the homo-polymer stretch should be discarded. This was done by an in-house perl script trim_homopolymers.pl. This script takes FASTQ file as an input, identifies reads > 700 bp and trims the homo-polymer stretch in the long reads based upon its sequence and low Phred Quality score at the 3-prime end of long reads.

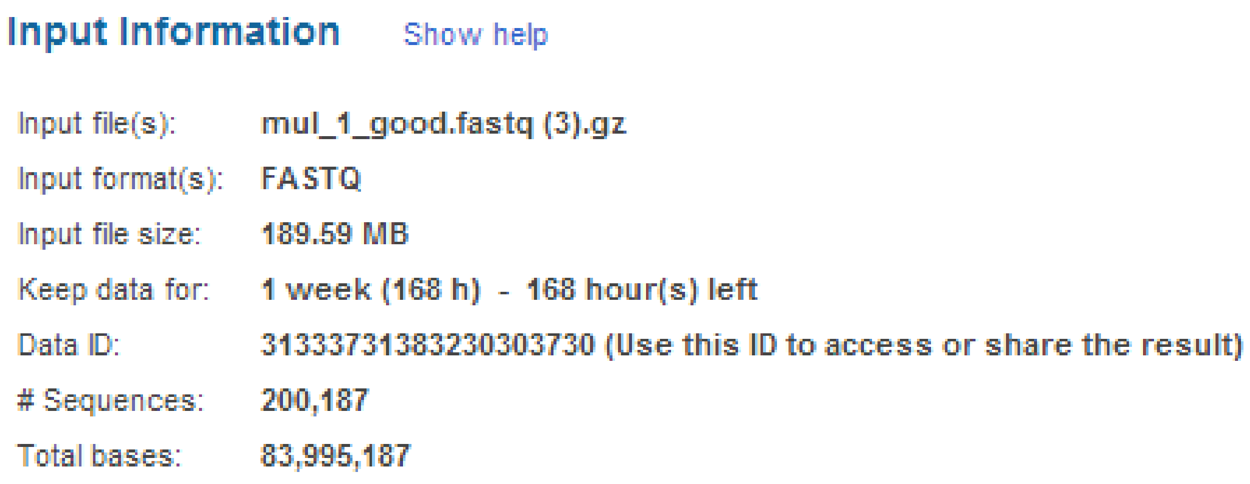
Parameters used for Data Processing	***Tools/Scripts***
Sequence Length Range – 60 to 1000 bp	***PRINSEQ, Filter_by_length.pl (perl script by SeqCrumbs)***
Base Quality Threshold – 15	***PRINSEQ, Filter_by_Quality.pl (perl script by SeqCrumbs)***
Low Complexity Regions	***DUST approach by PRINSEQ***
Ambiguous Bases (%) – 1%	***PRINSEQ***
Poly A/T removal threshold – 5 bp	***PRINSEQ***
Homo-polymer trimming	***Trim_homopolymer.pl (in-house perl script)***

Table #5 Parameters for Quality Trimming of the data-sets

5.2 Statistical Report of Processed Mulberry_Parent_1 and Mulberry_Parent_2 data-sets according to the parameters defined for quality-processing.

Significant improvement in read length, quality, ambiguous bases, Poly A/T tails was observed after processing of the data-sets by PRINSEQ and the above mentioned perl scripts. The improved statistics are as follows:

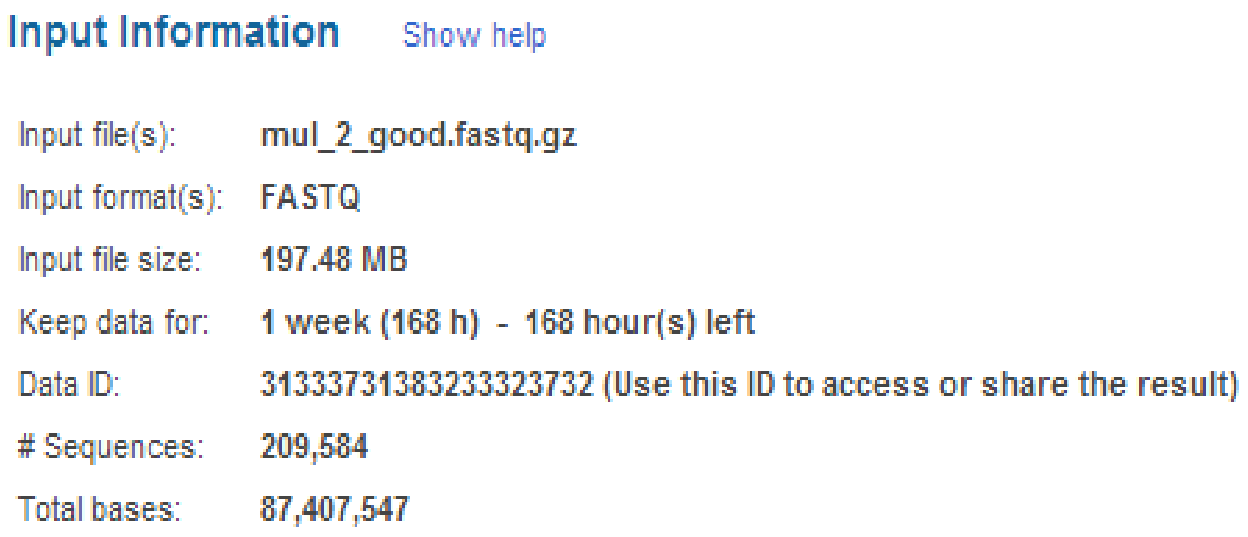
a) Number and Length of Sequences:



Input Information   Show help

Input file(s):	mul_1_good.fastq (3).gz
Input format(s):	FASTQ
Input file size:	189.59 MB
Keep data for:	1 week (168 h) - 168 hour(s) left
Data ID:	3133373138323030 3730 (Use this ID to access or share the result)
# Sequences:	200,187
Total bases:	83,995,187

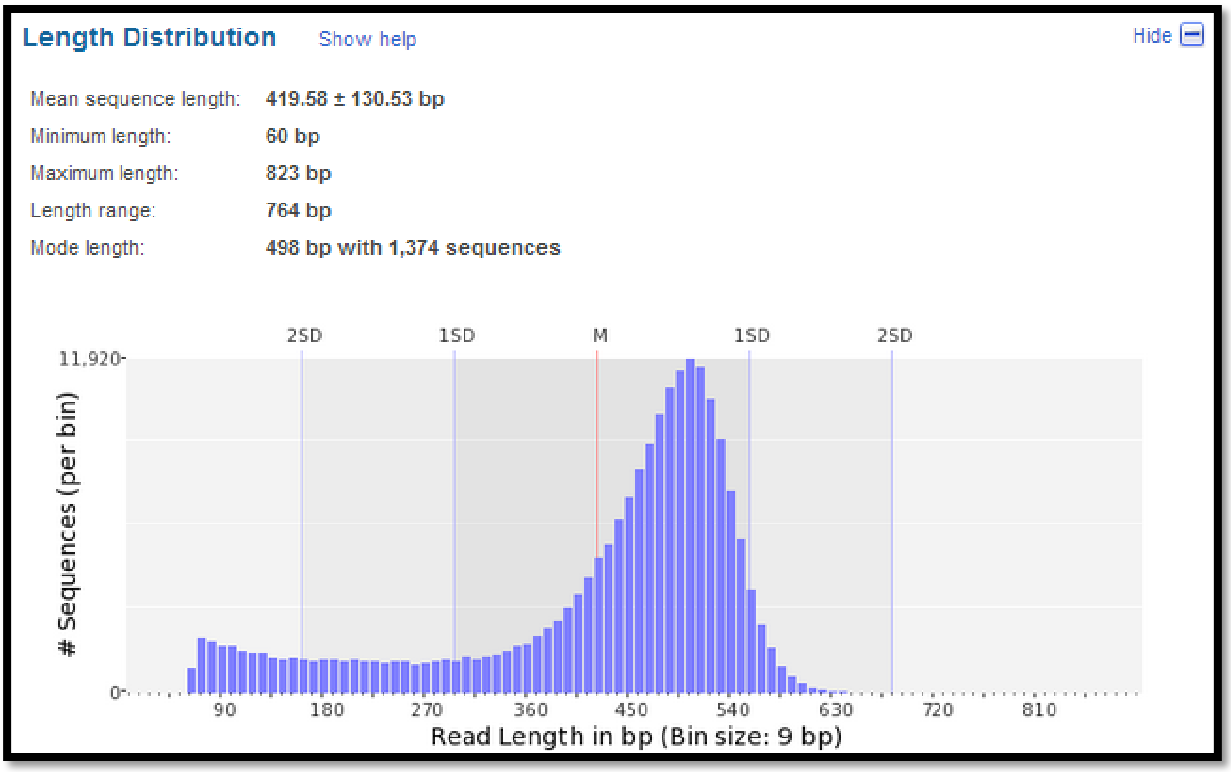
Figure #38 PRINSEQ Report of input data information for quality processed Mulberry_Parent_1 data-set



Input Information   Show help

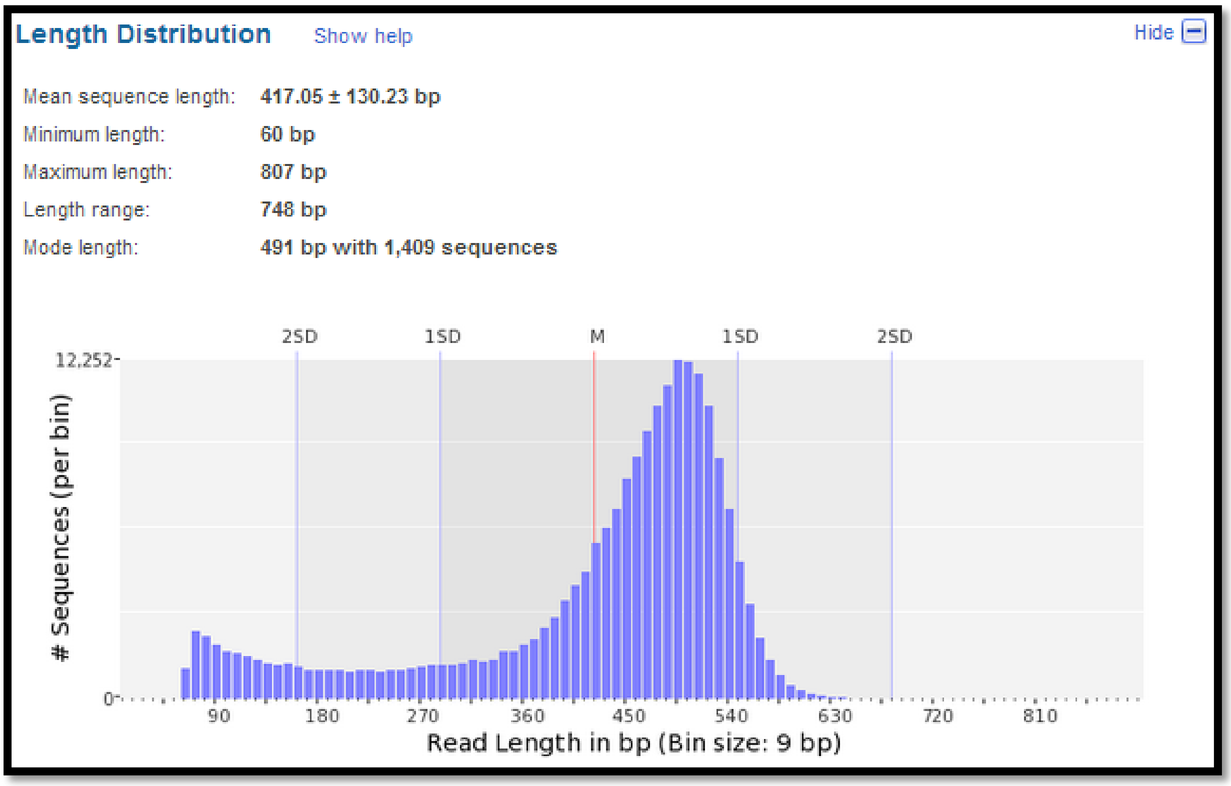
Input file(s):	mul_2_good.fastq.gz
Input format(s):	FASTQ
Input file size:	197.48 MB
Keep data for:	1 week (168 h) - 168 hour(s) left
Data ID:	3133373138323333323732 (Use this ID to access or share the result)
# Sequences:	209,584
Total bases:	87,407,547

Figure #39 PRINSEQ Report of input data information for quality processed Mulberry_Parent_2 data-set


Length Distribution

Mean sequence length: 419.58 ± 130.53 bp
Minimum length: 60 bp
Maximum length: 823 bp
Length range: 764 bp
Mode length: 498 bp with 1,374 sequences

Figure #40 PRINSEQ Report of Length Distribution for quality processed Mulberry_Parent_1 data-set


Length Distribution

Mean sequence length: 417.05 ± 130.23 bp
Minimum length: 60 bp
Maximum length: 807 bp
Length range: 748 bp
Mode length: 491 bp with 1,409 sequences

Figure #41 PRINSEQ Report of Length Distribution for quality processed Mulberry_Parent_2 data-set

The total number of reads in both the data-sets reduced as a consequence of removal of low-quality, ambiguous reads, longer reads with low quality at 3-prime end and reads with Poly A/T tails. The range of length for reads of both the data-sets after quality pre-processing of the data changed from 40-1130 bp to 60-823 bp. The mean sequence length became 417bp and 419bp for Mulberry_Parent_1 and for Mulberry_Parent_2 respectively.

b) Base-Quality Distribution:

The Phred Quality Score for processed data-sets improved drastically with the quality score threshold of 15.

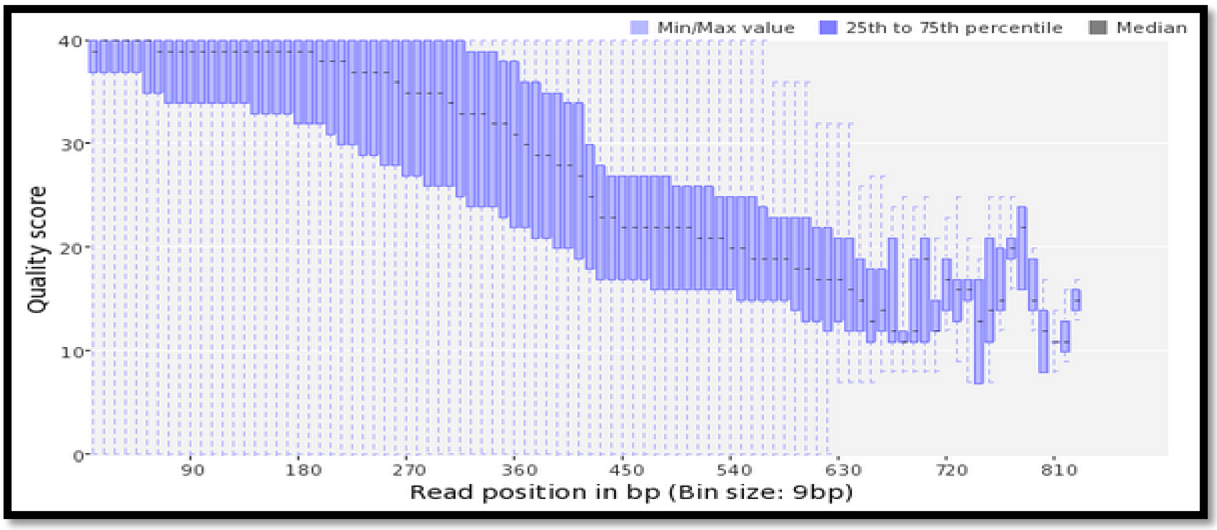


Figure #42 PRINSEQ Report of Base-Quality Distribution for quality processed Mulberry_Parent_1 data-set

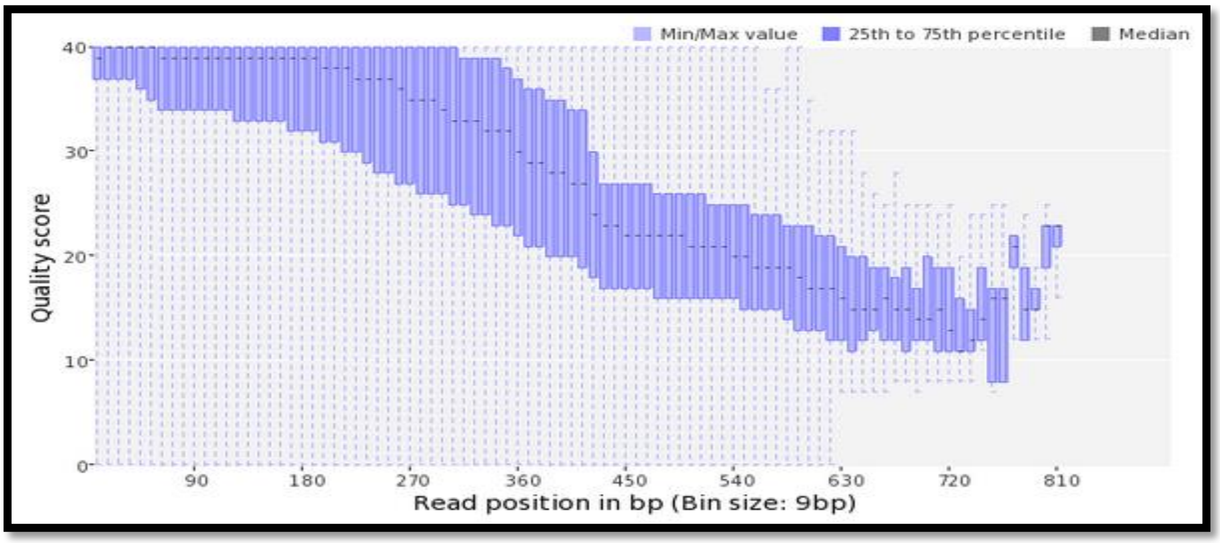


Figure #43 PRINSEQ Report of Base-Quality Distribution for quality processed Mulberry_Parent_2 data-set

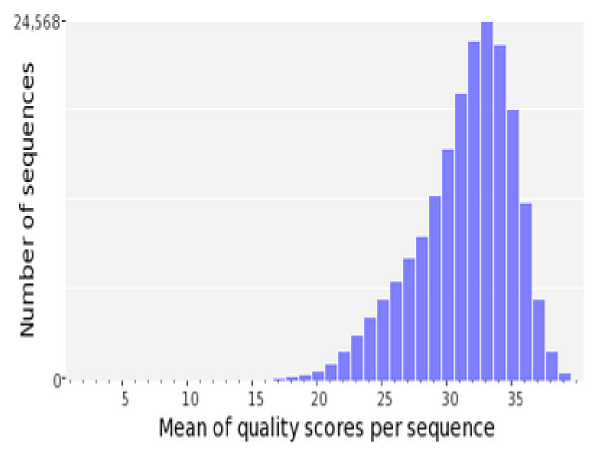
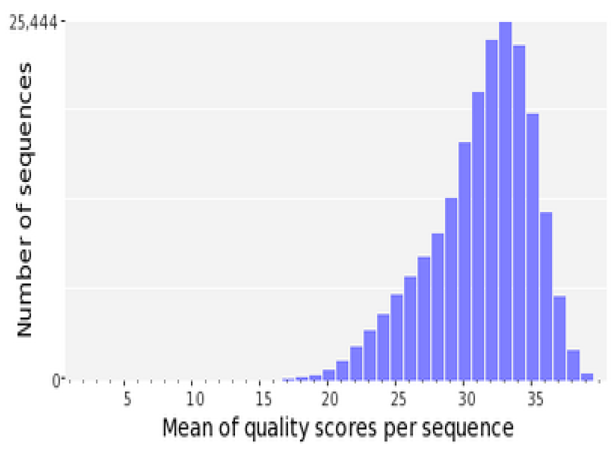
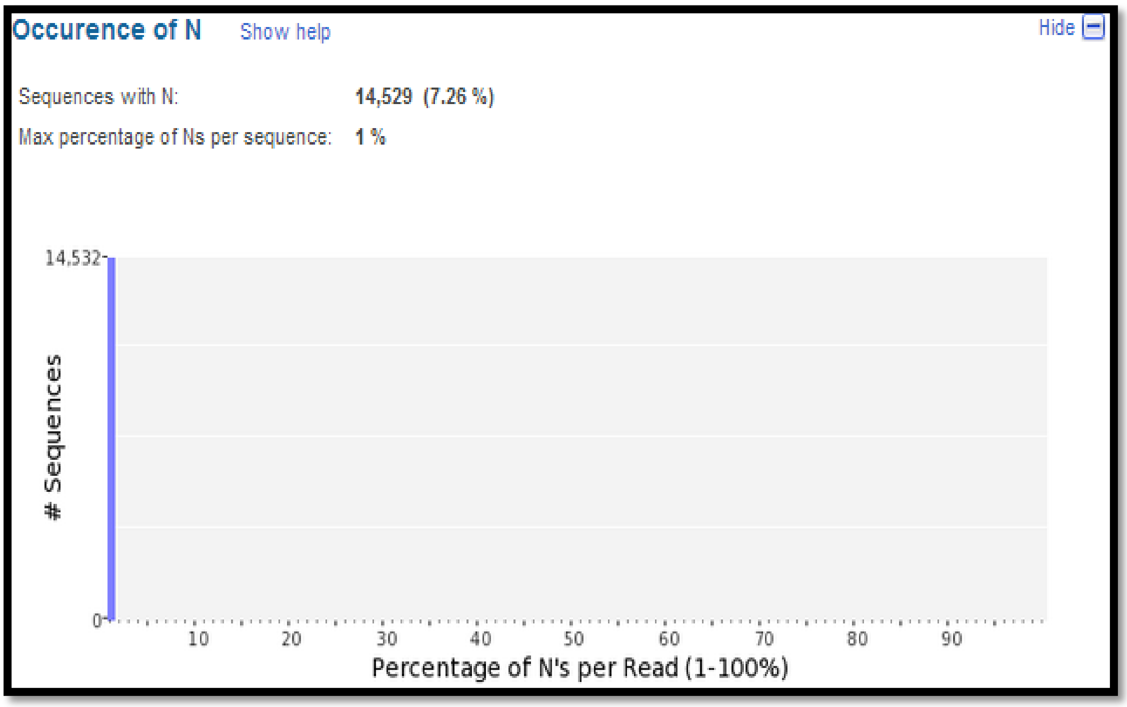
	
Mulberry_Parent_1	**Mulberry_Parent_2**

Figure #44 PRINSEQ Report of Mean Quality Distribution for Processed Mulberry_data-sets.

The mean quality of both the data-sets improved from 30 to 33 Phred quality score.

c) Ambiguous Bases:

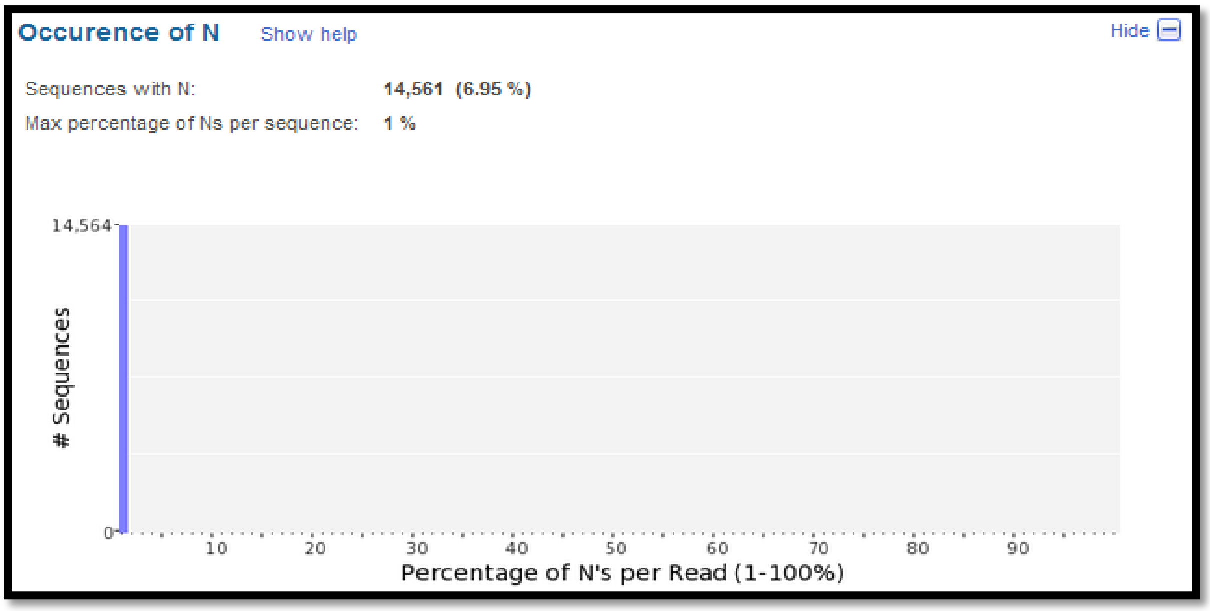
The number of reads with ambiguous bases (bases other than A/T/G/C) reduced in quality processed data-sets.


Occurence of N Show help Hide

Sequences with N: 14,529 (7.26 %)

Max percentage of Ns per sequence: 1 %

Figure #45 PRINSEQ Report of Ambiguous bases for Processed Mulberry_Parent_1


Occurence of N   Show help   Hide

Sequences with N: 14,561 (6.95 %)
Max percentage of Ns per sequence: 1 %

Figure #46 PRINSEQ Report of Ambiguous bases for Processed Mulberry_Parent_2

d) Poly A/T tails:

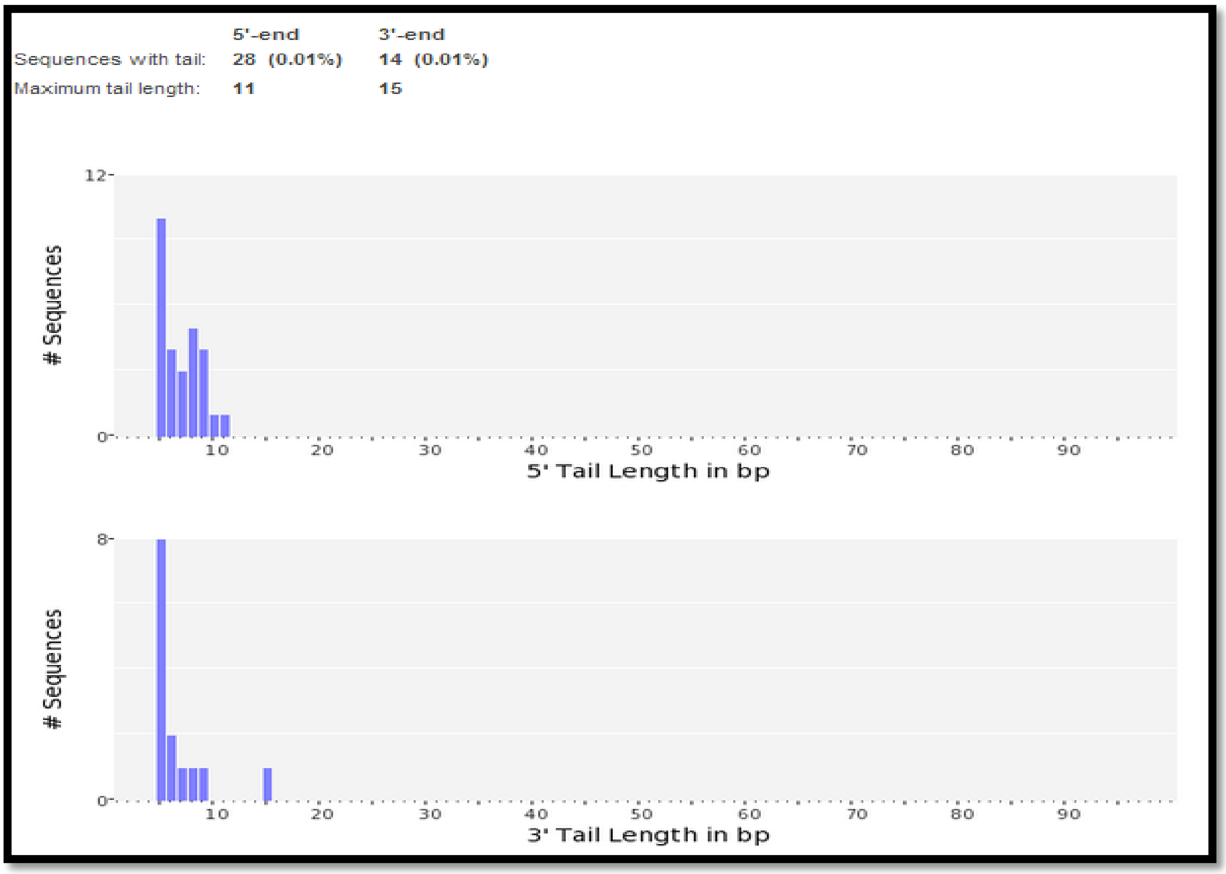

	5'-end	3'-end
Sequences with tail:	28 (0.01%)	14 (0.01%)
Maximum tail length:	11	15

Figure #47 PRINSEQ Report of Poly A/T tails for Processed Mulberry_Parent_1 data-set

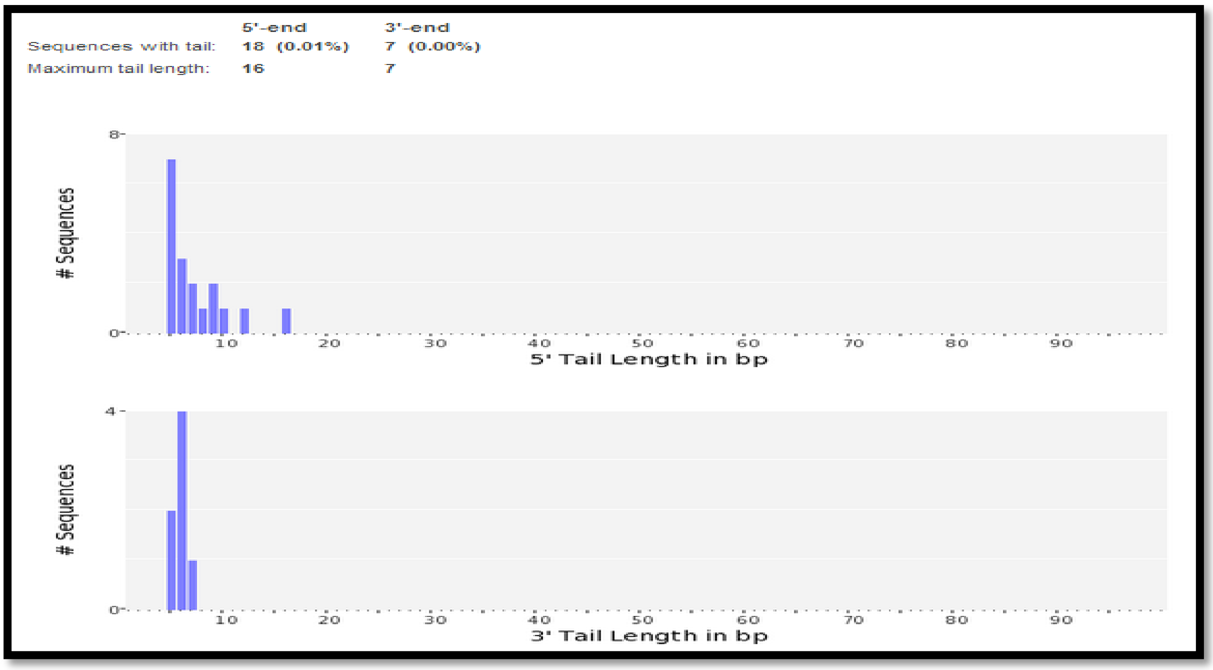


Figure #48 PRINSEQ Report of Poly A/T tails for Processed Mulberry_Parent_2 data-set

The reads with Poly A/T tails reduced to 0.02 % and 0.01 % for parent_1 and parent_2 respectively.

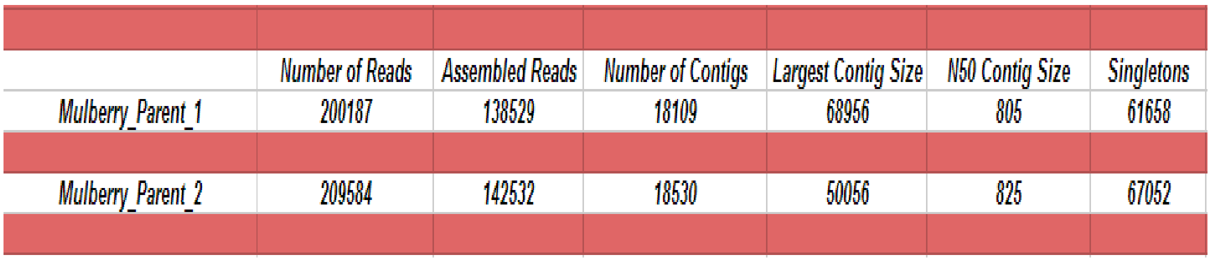
Table #6 provides a comparative statistical summary for the two data-sets before and after quality pre-processing.

	Mul_Parent_1	***Mul_P1_QP***	***Mul_Parent_2***	***Mul_P2_QP***
No. of Reads	***574,312***	***200,187***	***591,507***	***209,584***
Min_Read_len	***40***	***60***	***40***	***60***
Max_Read_len	***1120***	***823***	***1131***	***807***
Mean_Read_len	***398.65***	***419.58***	***402.53***	***417.05***
Percentage_N	***13.46%(77319)***	***7.26%(14529)***	***13.11%(77561)***	***6.95%(14561)***
Quality Avg	***30***	***33***	***30***	***33***
Poly A/T reads	***2.76% (15842)***	***0.02% (42)***	***2.81% (16622)***	***0.01% (25)***

Table #6 Comparative Summary Statistics of raw reads and quality processed reads for Mulberry_Parent_1 and Mulberry_Parent_2. Mul_Parent_1: Mulberry_Parent_1, Mul_P1_QP: Mulberry_Parent_1_Quality_Processed_data_set, Mul_Parent_2: Mulberry_Parent_2, Mul_P2_QP: Mulberry_Parent_2_Quality_Processed_data_set, Percentage_N: Percentage of Ambiguous bases, Quality Avg: Phred Base Quality Average of all the reads.

5.3 Assembly of raw reads of Mulberry_Parent_1 and Mulberry_Parent_2

Genome Assembly of raw reads was done by Newbler. Newbler uses De-Bruijn graph to assemble overlapping raw reads in contigs and unique reads into singletons.



	Number of Reads	Assembled Reads	Number of Contigs	Largest Contig Size	N50 Contig Size	Singletons
Mulberry_Parent_1	200187	138529	18109	68956	805	61658
Mulberry_Parent_2	209584	142532	18530	50056	825	67052

Table #7 Assembly Statistics of Mulberry_Parent_1 and Mulberry_Parent_2 generated by Newbler's from 454NewblerMetrics.txt by script called 454NewblerMetrics.pl

The reads assembled into 18109 and 18530 contigs of mulberry_parent_1 and mulberry_parent_2 respectively. There is approximately 10Kb difference between largest contigs of both the sets. This maybe due to the data coming different cultivars or due to the fact that reads overlapping to form the largest contig maybe from different regions of the genome of the two parents. The N50 contig size are 805 and 825 in mulberry_parent_1 and mulberry_parent_2 respectively. This indicates that in mulberry_parent_1 50% of all the contigs have length greater or equal to 805 and in mulberry_parent_2 50 % of all the contigs have length greater than 829. Theoretically, the longer the N50 value, the better the assembly.

The overall assembly is moderately reliable. The data obtained from Roche 454 run was from low coverage single-end library. This means that while library preparation for 454 pyro-sequencing, the genomic DNA was sheared randomly and was sequenced from single end. The high number of contigs and singletons (unique reads with overlaps) provide an evidence that during the library preparation the genomic DNA was highly sheared and reads specific to different genomic regions have thus been obtained. Although, the reads for organellar genomes should be present in higher proportions than nuclear counterparts, a fact which was further explored for mitochondrial genome assembly of mulberry (Wang *et al*, 2013).

5.4 Identification of Mitochondrial like reads from Mulberry_Parent_1 contigs and Mulberry_Parent_2 contigs.

Mitochondrial like reads in the two data-sets were identified by blasting (BLAST) a local database of 78 sequenced plant mitochondrial genomes with contigs of both the data-aets. This local database was created by the command makeblastdb (ncbi_blast_toolkit). The database for plant mitochondrial genome was downloaded from the following URL. http://www.ncbi.nlm.nih.gov/genomes/GenomesGroup.cgi?taxid=33090&opt=organelle.

The plant mitochondrial genome was blasted with contigs rather than with raw reads of both the parents because contigs provide information about a contiguous stretch of genome, hence contigs similar to mitochondrial genomes of plants will provide basis for identification of mitochondrial genome of mulberry. Reads assembling into mitochondrial like contigs identified by BLAST search were extracted from the raw read files of both the parents from an in-house developed shell script extract_reads_from_mt_like_contigs.sh. The script identified a total of 23,600 mitochondrial like reads in Mulberry_Parent_1 and 21,548 mitochondrial like reads in Mulberry_Parent_2. The script also created Standard Flow-gram format (sff) files of the mitochondrial like reads for both the data-sets- mul_1_mt_like_reads.sff and mul_2_mt_like_reads.sff.

	Total Reads	***Mt-like Reads***
Mulberry_Parent_1	***200,187***	***23,600***
Mulberry_Parent_2	***209,584***	***21,548***

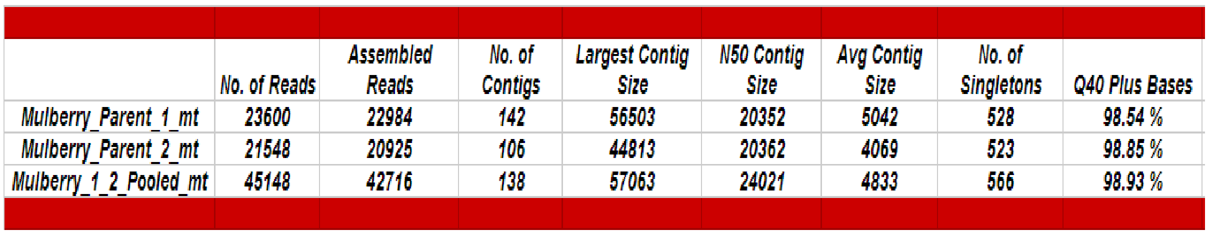
Table #8 Total Mitochondrial like reads in Mulberry_Parent_1 and Mulberry_Parent_2

5.5 Assembly of Mulberry Mitochondrial Genome

For assembling the mulberry mitochondrial genome, the sff files of both the data-sets were assembled individually by a 454 Roche specific assembler Newbler (gsAssembler). Besides that, a third assembly, by pooling the data of the two sff files was done. The raw sff files, mul_1_mt_like_reads.sff and mul_2_mt_like_reads.sff were joined by a command called sfffile. This command is in-built command of Newbler.

The new sff file created by joining the mul_1_mt_like_reads.sff and mul_2_mt_like_reads.sff was named mul_1_2_mt_like_reads.sff. This file was further used for the third pooled assembly. As the mitochondrial genome is conserved across different species of plants, pooled assembly should provide information which has been missed during genome sequencing in both the data-sets and should provide confidence about the regions being sequenced commonly in both the data-sets. Mitochondria of most higher plants are prone to house sequences of plastid and nuclear origin. This occurs due to a phenomenon called horizontal gene transfer (Mackenzie *et al*, 1999). To prevent reads of plastid origin from being assembled along with mitochondrial genome of mulberry, a screening database of plant chloroplast genomes was provided.

Newbler v 2.8 with stringent parameters (percent identity 98% and minimum overlap 40) was used for assembly.



	No. of Reads	Assembled Reads	No. of Contigs	Largest Contig Size	N50 Contig Size	Avg Contig Size	No. of Singletons	Q40 Plus Bases
Mulberry_Parent_1_mt	23600	22984	142	56503	20352	5042	528	98.54 %
Mulberry_Parent_2_mt	21548	20925	106	44813	20362	4069	523	98.85 %
Mulberry_1_2_Pooled_mt	45148	42716	138	57063	24021	4833	566	98.93 %

Table #9 Summary Statistics for Mulberry_mitochondrial_assembly.

Table #9 shows summary statistics for mulberry_mitochondrial_assembly. As expected, the assembly for pooled set is better than the assembly for individual data-sets of mulberry. This is evident by the size of largest contig formed in the assembly of pooled data-set, the total number of contigs and the Q40 Plus bases. The total number of contigs decreased as compared to Mulberry_Parent_1 which suggests that reads common to both the data-sets are overlapping to form a single contig rather than forming different contigs in individual assemblies. The N50 contig size increased in the pooled assembly suggesting that the pooled assembly is better than the individual assemblies. The largest contig size also increased owing to more available information for that particular contig in the pooled read data-set. Increased percentage of Q40 bases (bases having Phred Quality score of 40 or above) in the pooled assembly suggests that better quality of bases are being used in the assembly.

The contigs obtained in the three assembled data-sets were then screened for the contigs of nuclear origin in the mitochondrial genome. The reads coming from nuclear genomic regions should ideally be present in lower numbers as compared to their organellar counterparts.The fact that a cell contains many copies of organelles was exploited for the screening of contigs of nuclear origin. This led to the identification of read-coverage of all the contigs present in the three data-sets. Read-coverage is the number of overlapping reads resulting in the formation of contigs. So, contigs of nuclear origin would have lower read-coverage as compared to the mitochondrial contigs. Simply put, all things being equal, sequence with more coverage will be represented in larger contigs of higher quality than sequence with a lower degree of coverage (Elaine *et al*, 2002). The following table depicts the total number of high-quality, long (>2KB), high-coverage contigs and the estimated size of mitochondrial genome of mulberry_parent_1, of mulberry_parent_2 and of the pooled data-set.

	High-quality, High-coverage Contigs (>2KB)	***Estimated Size of Mt-Genome (bp)***
Mulberry_Parent_1	***26***	***376,145***
Mulberry_Parent_2	***28***	***369,949***
Mulberry_1_2_Pooled	***27***	***380,529***

Table #10 High-quality and high read-coverage contigs of three assembled data-sets with their estimated genome sizes.

The pooled assembly was definitely more informative as evident by the increased genome size. In the high-quality pooled set of contigs, we don't expect the presence of over-represented sequences. This fact has been further validated by the annotation of all three assemblies. We find better gene coverage in the overlapping/pooled data-set.

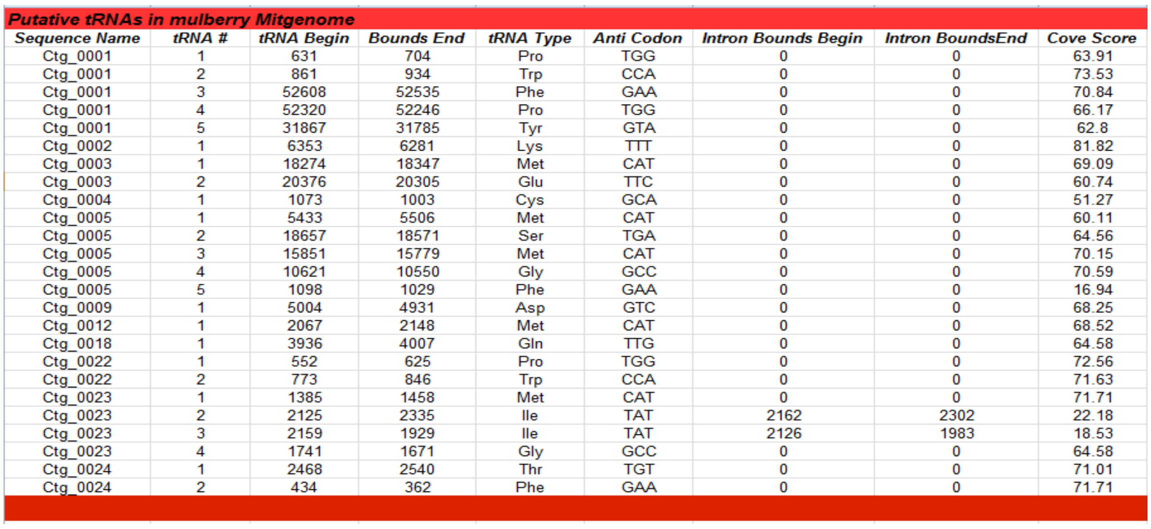
5.6 Annotation of Mulberry Mitochondrial Genome

The annotation of the high-coverage, high-quality contigs was done by a standalone software Mitofy and a in-built perl script separate_contigs_for_mitofy.pl. Mitofy is perl based program which identifies the functional Open Reading Frame (ORF) and the functional gene associated with it. The Contigs from all three data-sets were used for annotation. This was done to substantiate the fact that the pooled assembly of the data is better than the individual assemblies. By annotating the 3 genomes, we were able to recover nearly complete mitochondrial genome of mulberry in terms of functional gene-content.

Table #11 shows the mitochondrial genome annotation in all the three genomes.

A total of 70 genes were found by mitofy software which is 93 % of the functional mitochondrial genome of plants. *Rps10* and *rpl10* was not found in Mulberry_Parent_2 but was found in Mulberry_Parent_1 and the pooled data-set. Since these genes are found in pooled data-set, these are the true-positives found in the data-set. The gene *nad3* spans two contigs in the two individual assemblies – Contig 4,5 in Mulberry_Parent_1 and Contig 10,11 in Mulberry_Parent_2 but is present in a single Contig 6 in the pooled assembly. This means the two connected contigs in the individual assemblies are being assembled into a single contig in pooled assembly. Similarly, *cox2* and *cox1* are spanning two contigs in individual assemblies and are also spanning two contigs in pooled assembly. This shows a connection between two between two contigs: Contig 18,19 and Contig 19,20. The Contig connections need to be further validated by designing primers and doing some wet-lab experiments. A tRNA gene 'Leu-cp' was not found in individual assemblies but was found while annotating the pooled mt-genome assembly. Hence, the pooled assembly was able to provide missing information (genes) and substantiate some contig connections found by the genes spanning two or more contigs in the individual assemblies.

A total of 25 tRNAs were found using tRNAscan program, which are shown in the Table # 12.



Putative tRNAs in mulberry Mitgenome

Sequence Name	tRNA #	tRNA Begin	Bounds End	tRNA Type	Anti Codon	Intron Bounds Begin	Intron BoundsEnd	Cove Score
Ctg_0001	1	631	704	Pro	TGG	0	0	63.91
Ctg_0001	2	861	934	Trp	CCA	0	0	73.53
Ctg_0001	3	52608	52535	Phe	GAA	0	0	70.84
Ctg_0001	4	52320	52246	Pro	TGG	0	0	66.17
Ctg_0001	5	31867	31785	Tyr	GTA	0	0	62.8
Ctg_0002	1	6353	6281	Lys	TTT	0	0	81.82
Ctg_0003	1	18274	18347	Met	CAT	0	0	69.09
Ctg_0003	2	20376	20305	Glu	TTC	0	0	60.74
Ctg_0004	1	1073	1003	Cys	GCA	0	0	51.27
Ctg_0005	1	5433	5506	Met	CAT	0	0	60.11
Ctg_0005	2	18657	18571	Ser	TGA	0	0	64.56
Ctg_0005	3	15851	15779	Met	CAT	0	0	70.15
Ctg_0005	4	10621	10550	Gly	GCC	0	0	70.59
Ctg_0005	5	1098	1029	Phe	GAA	0	0	16.94
Ctg_0009	1	5004	4931	Asp	GTC	0	0	68.25
Ctg_0012	1	2067	2148	Met	CAT	0	0	68.52
Ctg_0018	1	3936	4007	Gln	TTG	0	0	64.58
Ctg_0022	1	552	625	Pro	TGG	0	0	72.56
Ctg_0022	2	773	846	Trp	CCA	0	0	71.63
Ctg_0023	1	1385	1458	Met	CAT	0	0	71.71
Ctg_0023	2	2125	2335	Ile	TAT	2162	2302	22.18
Ctg_0023	3	2159	1929	Ile	TAT	2126	1983	18.53
Ctg_0023	4	1741	1671	Gly	GCC	0	0	64.58
Ctg_0024	1	2468	2540	Thr	TGT	0	0	71.01
Ctg_0024	2	434	362	Phe	GAA	0	0	71.71

Table #12 tRNAs as predicted by tRNAscan in Mulberry_1_2_Pooled_Mit_Assembly.

Ctg:Contig, tRNA Begin: Start Coordinate of tRNA sequence on the Contig, tRNA Bounds End: Stop Coordinate of tRNA sequence on the Contig.

Contig 23 has tRNAs which are intron bound.

A gff (General Feature File) for the pooled assembly covering the genes and tRNAs is presented below:

SeqName	*Source*	*Feature*	*Start*	*End*	*Strand*
Pro	Contig_1 Roche 454 NGS data	tRNA	631	704	+
Trp	Contig_1 Roche 454 NGS data	tRNA	861	934	+
nad5	Contig_1 Roche 454 NGS data	gene	11679	12893	+
nad1	Contig_1 Roche 454 NGS data	gene	15844	15458	-
nad2	Contig_1 Roche 454 NGS data	gene	28584	27886	-
Tyr	Contig_1 Roche 454 NGS data	tRNA	31867	31785	-
rps7	Contig_1 Roche 454 NGS data	gene	51641	51108	-
Pro	Contig_1 Roche 454 NGS data	tRNA	52320	52246	-
Phe	Contig_1 Roche 454 NGS data	tRNA	52608	52538	-
sdh4	Contig_1 Roche 454 NGS data	gene	55340	54978	-
cox3	Contig_1 Roche 454 NGS data	gene	56074	55271	-
Lys	Contig_2 Roche 454 NGS data	tRNA	6353	6281	-
ccmB	Contig_2 Roche 454 NGS data	gene	8827	9456	+
rps10	Contig_2 Roche 454 NGS data	gene	10979	10857	-
cob	Contig_2 Roche 454 NGS data	gene	20968	22146	+
matR	Contig_2 Roche 454 NGS data	gene	22271	24265	+
ccmFn	Contig_3 Roche 454 NGS data	gene	761	2590	+
mttB	Contig_3 Roche 454 NGS data	gene	3314	4149	+
ccmc	Contig_3 Roche 454 NGS data	gene	17534	18283	+
Ile-cp	Contig_3 Roche 454 NGS data	tRNA	18274	18347	+
Glu	Contig_3 Roche 454 NGS data	tRNA	20376	20305	-
rps4	Contig_3 Roche 454 NGS data	gene	25298	26263	+
nad6	Contig_3 Roche 454 NGS data	gene	26914	27585	+
Cys-mt	Contig_4 Roche 454 NGS data	tRNA	1073	1003	-
rpl10	Contig_4 Roche 454 NGS data	gene	6139	6252	+
atp9	Contig_4 Roche 454 NGS data	gene	8263	8027	-
rps13	Contig_4 Roche 454 NGS data	gene	16663	16316	-
nad7	Contig_4 Roche 454 NGS data	gene	20741	20277	-
Phe	Contig_5 Roche 454 NGS data	tRNA	1098	1029	-
rrnL	Contig_5 Roche 454 NGS data	gene	1639	3388	+
Met-f	Contig_5 Roche 454 NGS data	tRNA	5433	5506	+
Gly	Contig_5 Roche 454 NGS data	tRNA	10623	10550	-
Met-cp	Contig_5 Roche 454 NGS data	tRNA	15851	15779	-
Ser	Contig_5 Roche 454 NGS data	tRNA	18658	18571	-
rps3	Contig_6 Roche 454 NGS data	gene	1	1553	+
rpl16	Contig_6 Roche 454 NGS data	gene	1540	1986	+
rrnS	Contig_6 Roche 454 NGS data	gene	15004	16938	+
rrn5	Contig_6 Roche 454 NGS data	gene	17512	17631	+
rps12	Contig_6 Roche 454 NGS data	gene	23817	23443	-
nad3	Contig_6 Roche 454 NGS data	gene	24022	23870	-
nad5_ex3	Contig_7 Roche 454 NGS data	gene	4446	4467	+
rpl2	Contig_7 Roche 454 NGS data	gene	5287	5556	+
rps19	Contig_7 Roche 454 NGS data	gene	5566	5760	+
atp1	Contig_8 Roche 454 NGS data	gene	11710	13236	+
rps2	Contig_8 Roche 454 NGS data	gene	12097	12237	+

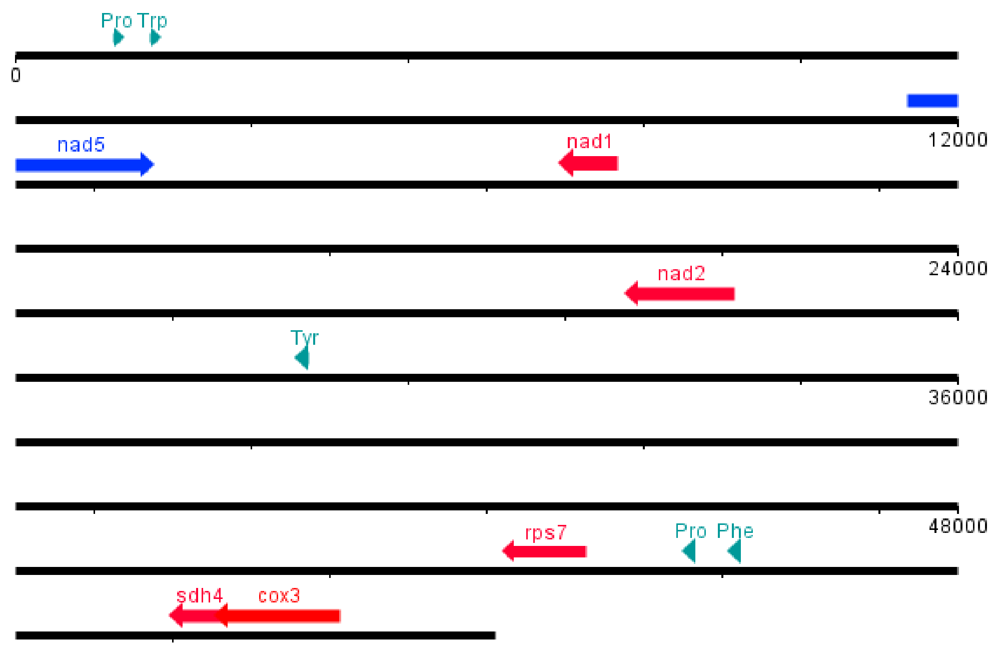
Asp	Contig_9 Roche 454 NGS data	tRNA	4994	4921	-
rps1	Contig_9 Roche 454 NGS data	gene	5784	5840	+
nad4	Contig_9 Roche 454 NGS data	gene	10211	10965	+
Ile	Contig_12 Roche 454 NGS data	tRNA	2067	2148	+
atp4	Contig_12 Roche 454 NGS data	gene	7669	7073	-
nad4L	Contig_12 Roche 454 NGS data	gene	8097	7798	-
sdh3	Contig_13 Roche 454 NGS data	gene	168	254	+
atp8	Contig_13 Roche 454 NGS data	gene	8215	8691	+
atp6	Contig_14 Roche 454 NGS data	gene	4225	5025	+
nad9	Contig_17 Roche 454 NGS data	gene	5781	5209	-
ccmFc	Contig_18 Roche 454 NGS data	gene	150	911	+
Gln	Contig_18 Roche 454 NGS data	tRNA	3936	4007	+
cox2	Contig_18 Roche 454 NGS data	gene	4914	4923	+
cox2	Contig_19 Roche 454 NGS data	gene	1	692	+
cox1	Contig_19 Roche 454 NGS data	gene	3221	4448	+
cox1	Contig_20 Roche 454 NGS data	gene	1	5	+
Pro	Contig_22 Roche 454 NGS data	tRNA	552	625	+
Trp-cp	Contig_22 Roche 454 NGS data	tRNA	773	846	+
rps14	Contig_23 Roche 454 NGS data	gene	12	1046	+
Ile	Contig_23 Roche 454 NGS data	tRNA	2159	1929	-
Ile	Contig_23 Roche 454 NGS data	Intron	2126	1983	-
Ile	Contig_23 Roche 454 NGS data	tRNA	2125	2335	+
Ile	Contig_23 Roche 454 NGS data	Intron	2162	2302	+
Phe	Contig_24 Roche 454 NGS data	tRNA	434	362	-
Thr	Contig_24 Roche 454 NGS data	tRNA	2468	2540	-
Leu-cp	Contig_26 Roche 454 NGS data	tRNA	2431	2477	+

Table #13 General Feature File of Mulberry Mit-Genome

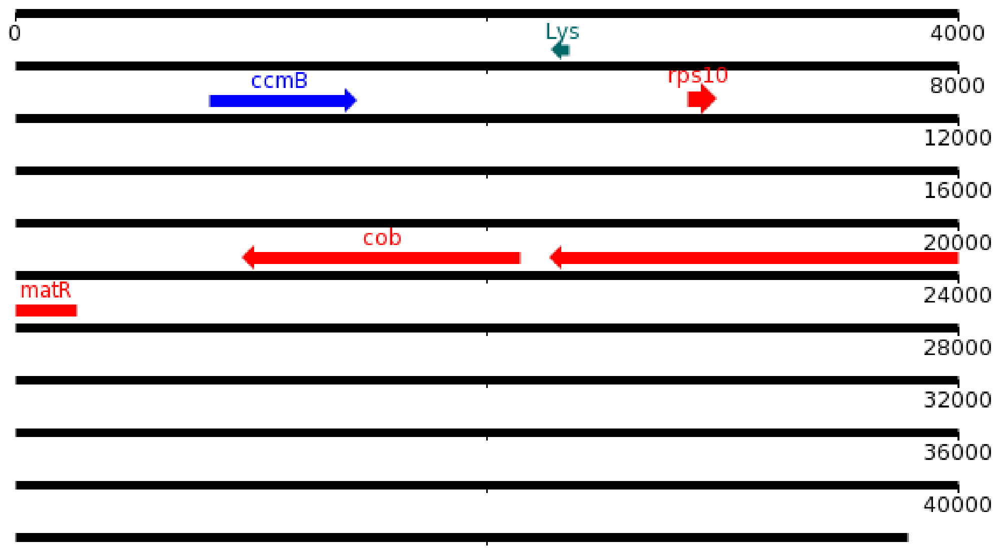
According to the gff file of Mulberry mit-genome, Feature Blocks of all the Contigs were Constructed using DNAPlotter. Feature Blocks are blocks of gene fragments, tRNA and Introns spanning a particular Contig.

The Feature Blocks of Mulberry Mit-Genome are as follows:

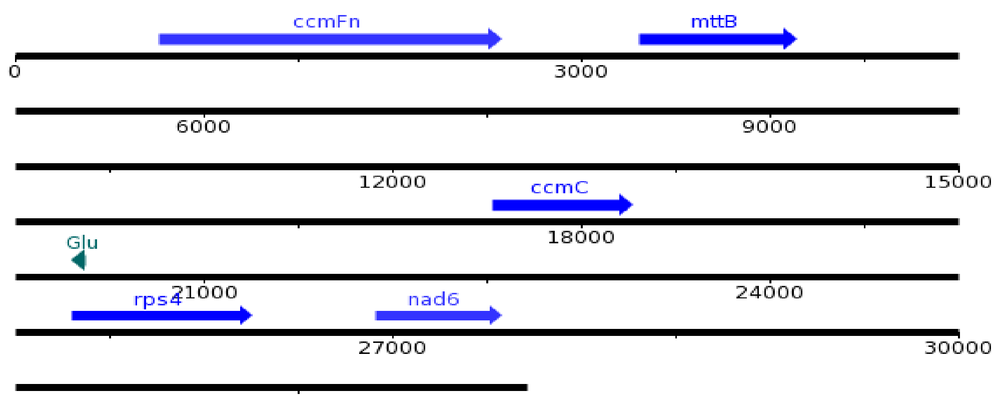
Contig 1



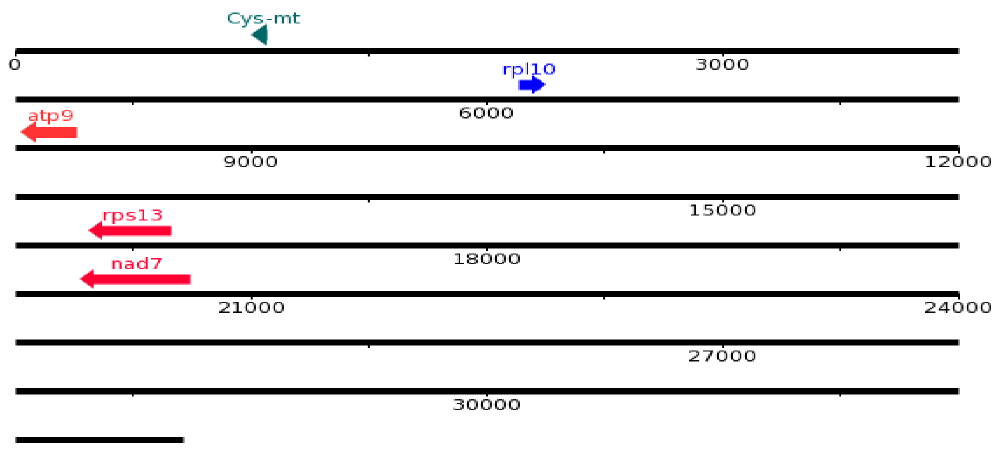
Contig 2



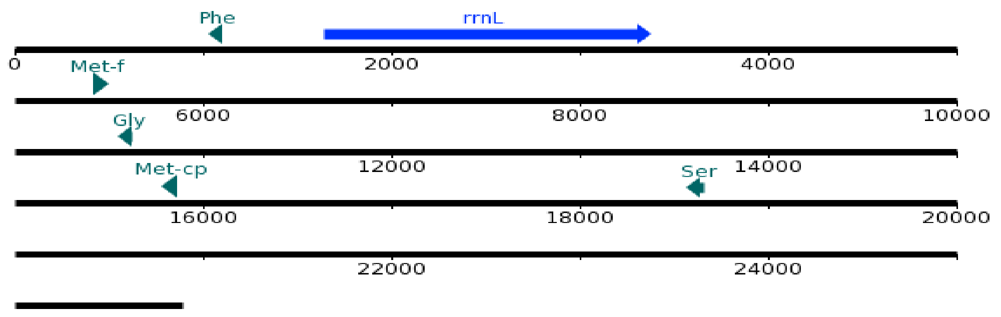
Contig 3



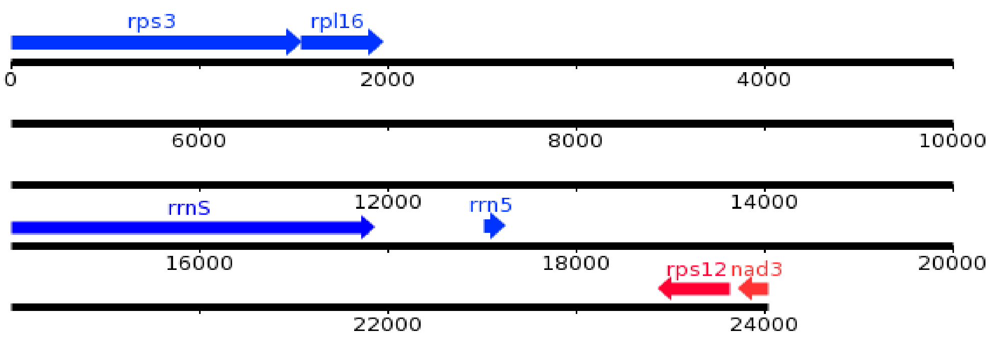
Contig 4



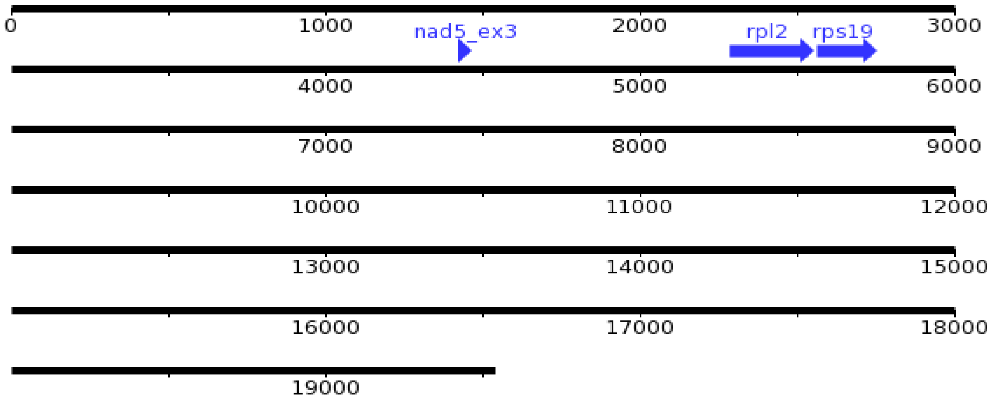
Contig 5



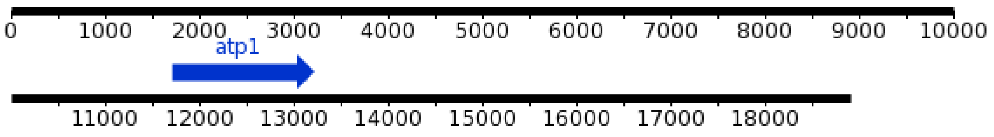
Contig 6



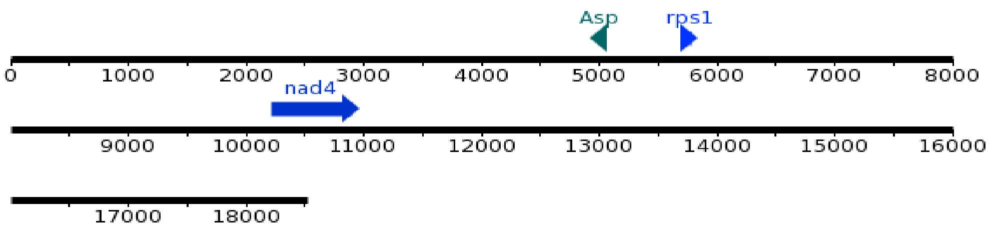
Contig 7



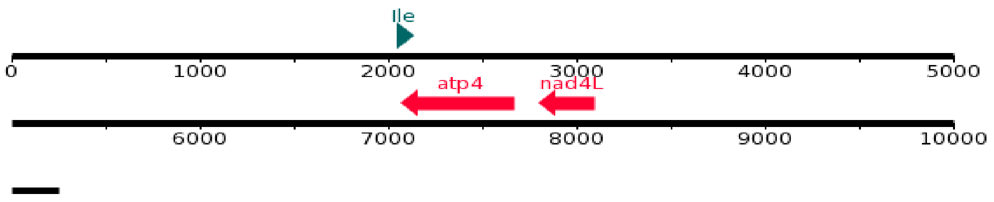
Contig 8



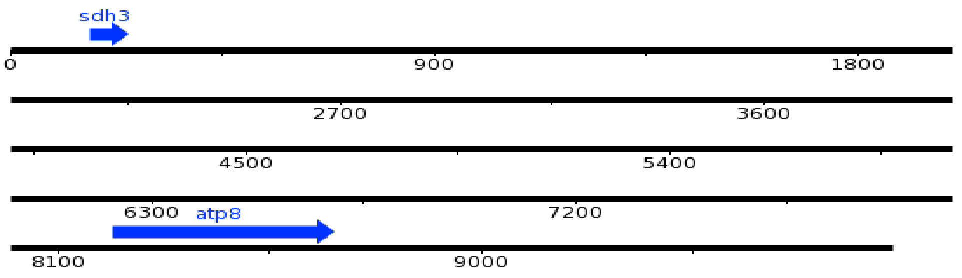
Contig 9



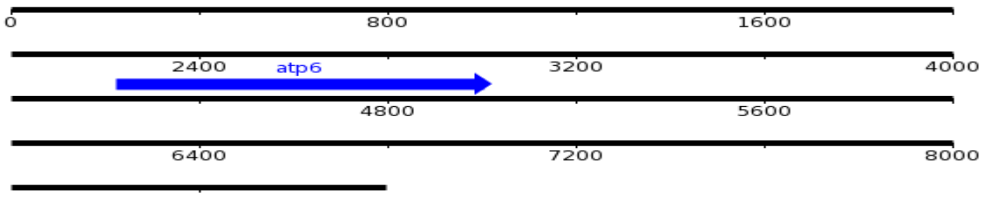
Contig 12



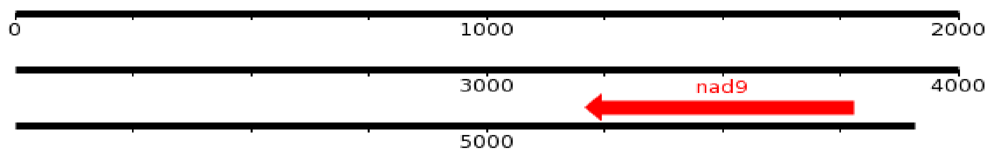
Contig 13



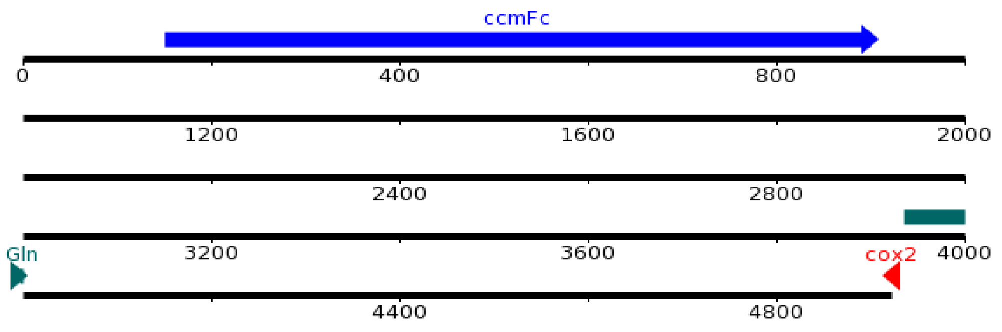
Contig 14



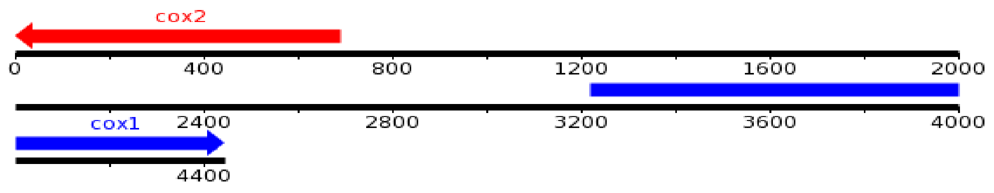
Contig 17



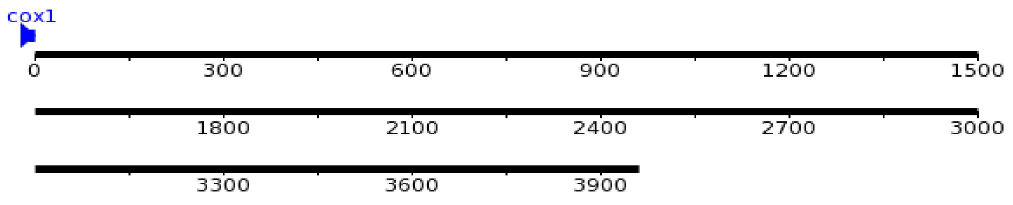
Contig 18



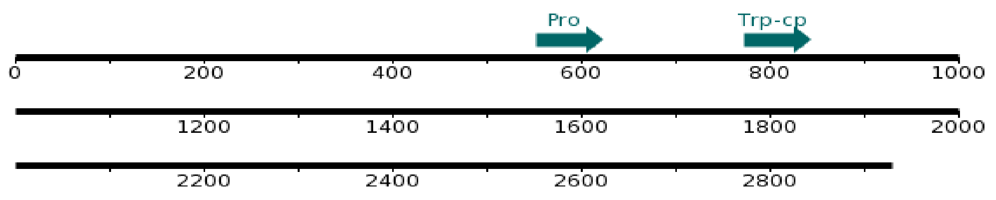
Contig 19



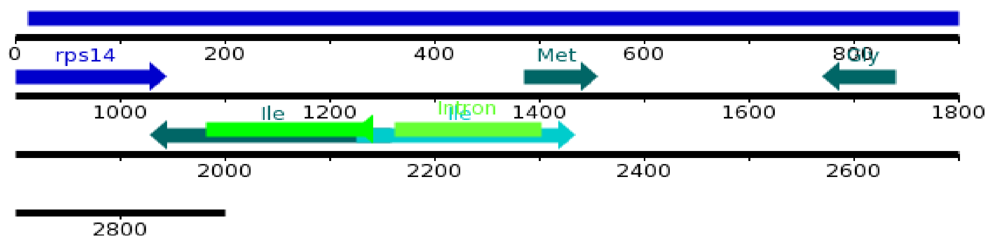
Contig 20



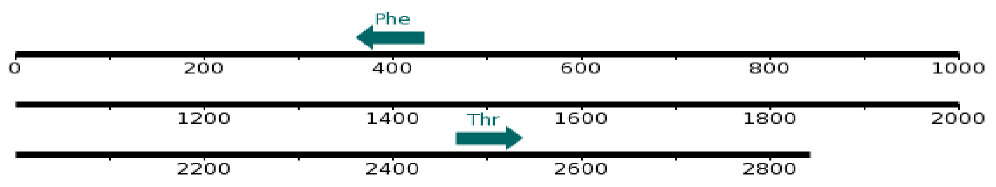
Contig 22



Contig 23



Contig 24



Contig 26

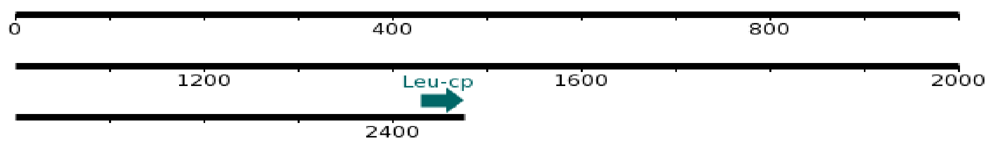


Figure #49 Feature-Blocks for Mulberry-Mit Genome. Color Code : Blue – Genes in forward strand, Red – Genes in Reverse Strand, Green – tRNAs, Fluorescent Green - Introns

No putative conserved domain was found in Contig 10, Contig 11, Contig 15, Contig 16, Contig 21, Contig 25, and Contig 27. NCBI's BLAST (Basic Local Alignment Tool blastn and blastx) was used to annotate the un-annotated contigs.

SeqName	***Genomic Feature (identified by BLAST)***	***Co-ordinates on Sequence***
Contig 10	***ATP synthase F0 subunit 9***	***10234..10380 (Plus)***
Contig 11	***hypothetical protein (mitochondrion)***	***7076..5574 (Minus)***
Contig 15	***hypothetical chloroplast RF2***	***3..488 (Plus)***
Contig 16	***cytochrome c maturation protein CcmC***	***4611..6397 (Plus)***
Contig 21	***hypothetical protein MTR_5g050970***	***3007..2300 (Minus)***
Contig 25	***repeat_type=inverted***	***475..751 (Plus)***
Contig 27	***uncharacterized RNA-binding protein C660***	***1761..2225 (Plus)***

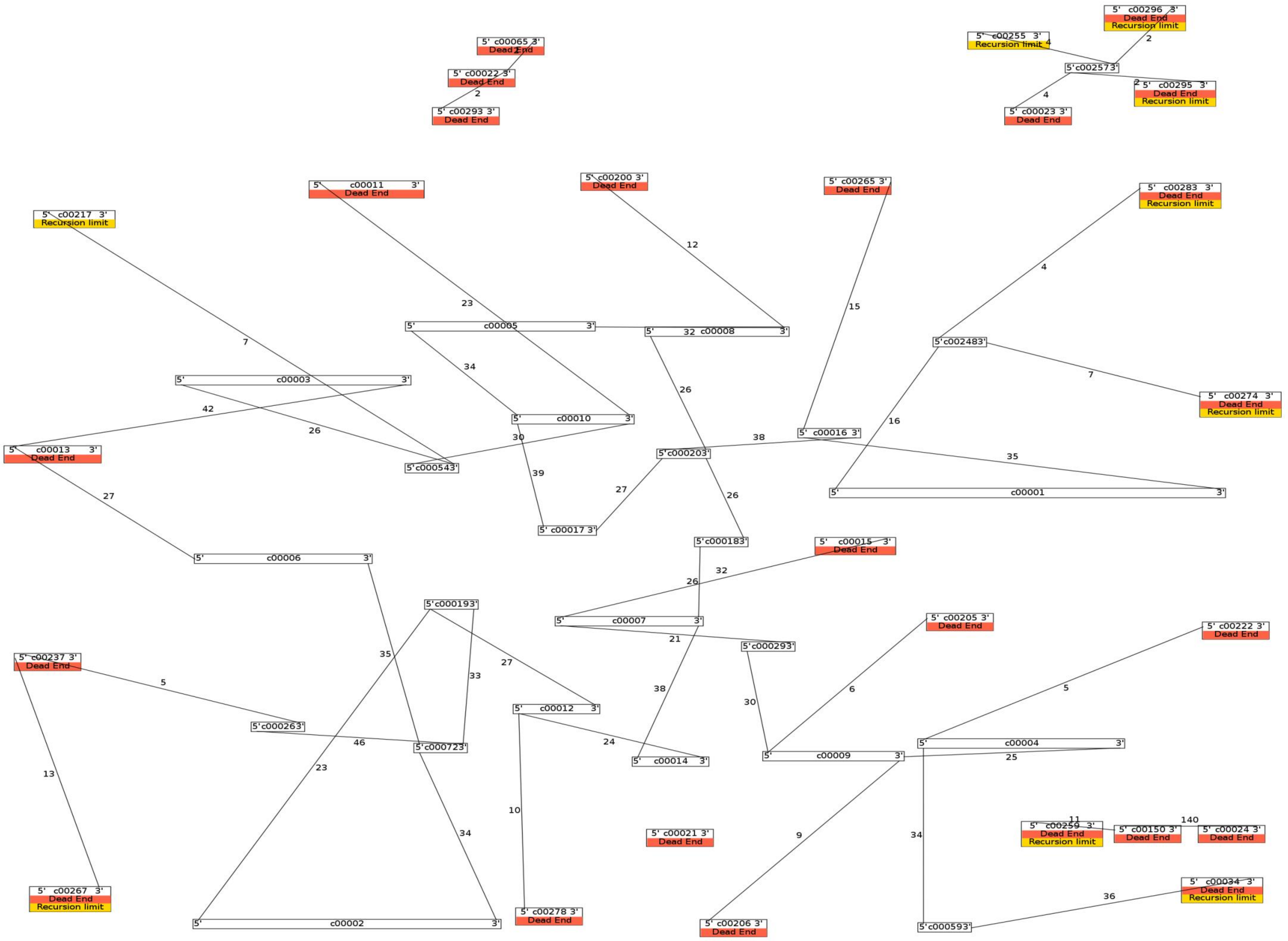
Table #14 Annotation of Un-annotated contigs revealing the presence of nuclear copies of mitochondrial DNA, and DNA from plastid origin.

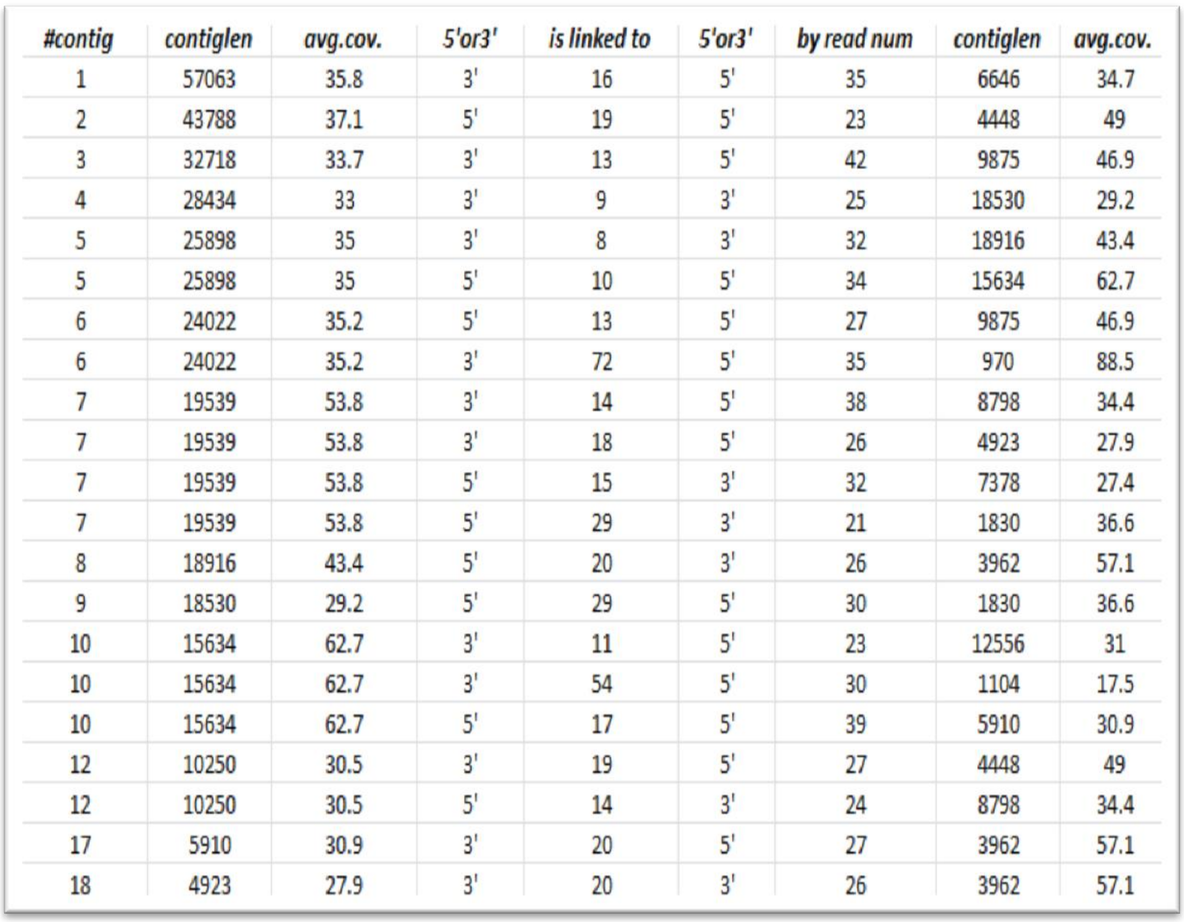
Annotation of Contig 15 revealed a non-funtional 454 bp fragment of plastid origin. Repeats and non-functional genes were also identified.

5.6 Mulberry Mitochondrial Genome Finishing

Genome finishing is a process in which contiguous segments of sequence are ordered and linked to one another and any ambiguities or discrepancies among the individual reads are resolved (Elaine *et al*, 2002). A finishing stage is critical to the usefulness of the final data.

Genome Finishing refers to the proper contig order and the forward and reverse strand information. This was deciphered by a perl script called bb.454ContigNet.pl which derives information from 454ContigGraph.txt file and joins the contigs together with the help of overlapping reads between them. It basically gives a De-bruijn graph of the contigs. The contigs are the nodes and the overlapping reads between them act as edges for the nodes. The De-Bruijn Graph of the Contigs derived from bb.454ContigNet.pl gives the Contig Connection and thus the Contig Order.


5' c00296 3' Dead End Recursion limit
5' c00255 3' Recursion limit
5'c002573'
5' c00295 3' Dead End Recursion limit
5' c00023 3' Dead End
5' c00065 3' Dead End
5' c00022 3' Dead End
5' c00293 3' Dead End
5' c00283 3' Dead End Recursion limit
5' c00274 3' Dead End Recursion limit
5'c002483'
5' c00265 3' Dead End
5' c00016 3'
5' c00001 3'
5' c00200 3' Dead End
5' c00008 3'
5' c00011 3' Dead End
5' c00005 3'
5' c00010 3'
5'c000203'
5' c00017 3'
5'c000183'
5' c00015 3' Dead End
5' c00217 3' Recursion limit
5' c00003 3'
5' c00013 3' Dead End
5'c000543'
5' c00006 3'
5'c000193'
5' c00007 3'
5'c000293'
5' c00205 3' Dead End
5' c00222 3' Dead End
5' c00004 3'
5' c00009 3'
5' c00012 3'
5' c00014 3'
5' c00237 3' Dead End
5'c000263'
5'c000723'
5' c00002 3'
5' c00278 3' Dead End
5' c00021 3' Dead End
5' c00206 3' Dead End
5'c000593'
5' c00259 3' Dead End Recursion limit
5' c00150 3' Dead End
5' c00024 3' Dead End
5' c00034 3' Dead End Recursion limit
5' c00267 3' Dead End Recursion limit
2
4
2
4
2
2
12
15
4
7
16
35
38
23
34
26
42
26
30
39
27
26
7
27
32
26
21
38
30
6
5
25
34
9
36
11
140
35
27
33
5
46
24
13
23
34
10



#contig	contiglen	avg.cov.	5'or3'	is linked to	5'or3'	by read num	contiglen	avg.cov.
1	57063	35.8	3'	16	5'	35	6646	34.7
2	43788	37.1	5'	19	5'	23	4448	49
3	32718	33.7	3'	13	5'	42	9875	46.9
4	28434	33	3'	9	3'	25	18530	29.2
5	25898	35	3'	8	3'	32	18916	43.4
5	25898	35	5'	10	5'	34	15634	62.7
6	24022	35.2	5'	13	5'	27	9875	46.9
6	24022	35.2	3'	72	5'	35	970	88.5
7	19539	53.8	3'	14	5'	38	8798	34.4
7	19539	53.8	3'	18	5'	26	4923	27.9
7	19539	53.8	5'	15	3'	32	7378	27.4
7	19539	53.8	5'	29	3'	21	1830	36.6
8	18916	43.4	5'	20	3'	26	3962	57.1
9	18530	29.2	5'	29	5'	30	1830	36.6
10	15634	62.7	3'	11	5'	23	12556	31
10	15634	62.7	3'	54	5'	30	1104	17.5
10	15634	62.7	5'	17	5'	39	5910	30.9
12	10250	30.5	3'	19	5'	27	4448	49
12	10250	30.5	5'	14	3'	24	8798	34.4
17	5910	30.9	3'	20	5'	27	3962	57.1
18	4923	27.9	3'	20	3'	26	3962	57.1

Table #15 Contig-Connections generated according to the De-Bruihn graph. These Connections are neat and are to be validated further by designing primers and PCR experiments.

The De-Bruijn Graph for the Contigs is provided in the Figure #50. Some Connections between the contigs are quite clean. According to 454NewblerMetrices.txt file, 11.3 % of the contigs are neatly connected to other contigs. As evident in table # 15 the number of overlapping reads between two connecting contigs are mostly >30 . So the neat connections between two contigs are strong. Some edges are ambiguous (a node having more than two edges). These ambiguous edges could be repeats. For resolving these repeats, CodonCode aligner was used to align contigs with each other. If a sequence or a contig is a repeat in the genome, it will align with its repetitive counter-part. Codon-code aligner identified 4 pairs of repetitive contigs in the assembly. The Coordinates of the repetitive contigs were blasted against NCBI nr/nt database and the genomic features present as repeats in the mitochondrial genome were identifed.

Repetitive Regions	***Co-ordinates***	***Strand***	***Genomic Feature***
Contig 14 (func)	***4464..4638***	***Plus***	***Gene: atp6***
Contig 26 (repeat)	***107..281***	***Minus***	

Contig 5 (func)	***11751..12058***	***Minus***	***tRNA: Gly***
Contig 25 (repeat)	***1757..2036***	***Minus***	
Contig 1 (repeat)	***1053..1649***	***Plus***	***Gene: atp4***
Contig 12 (func)	***7669..7073***	***Minus***	
Contig 8 (repeat)	***5781..6186***	***Plus***	***Gene: nad9***
Contig 17 (func)	***5415..5820***	***Minus***	

Table #16 Repeats in Mulberry Mit-Genome.

The repeats in the mit-genome assembly are false links/forks present in assembly graph and they normally belong to different genomes. The mitochondrial genome of higher plants is loaded with genes of nuclear origin. These regions are called numts (nuclear mitochondrial DNA) . Numts are products of horizontal gene-transfer and are usually non-functional in the mitochondrial genome (Mishmar D *et al*, 2004). Our procedure found 4 pairs of repetitive regions in the assembled mit-genome of mulberry. Three numts were identified owing to their non-functionality or lack of functional ORF. Some more numts were found from BLAST results of initially un-annotated contigs.

NuMts	***Coordinates on the mit-genome***	***Genomic Feature***
Contig 1	***1053..1649***	***Atp 4***
Contig 26	***107..281***	***Atp 6***
Contig 8	***5781..6186***	***Nad 9***
Contig 10	***10234..10380***	***Atp 9***
Contig 16	***4611..6397***	***ccmC***
Contig 25	***475..751***	***repeat_type=inverted***

Table #17 Numts and Repeats identified in the Mulberry Mit-genome.

CHAPTER #6

CONCLUSION

The Mulberry Mitochondrial genome assembly using 454 Roche NGS data provides a mit-genome of 380,529 bp with 45 (functional) genes, 25tRNA genes, 2 rRNA genes, 3 numts and a read-coverage of 66x. A 454 bp fragment of plastid DNA was also incorporated in the Mulberry Mit-genome. A General feature file (gff) and Feature-Blocks for all the contigs were created. We were able to retreive near complete mit-genome of Mulberry in terms of functional gene content. Only one functional genes, *rps11* was found missing in the mit-genome of mulberry.

We thus report the first-ever highly annotated mitochondrial genome of *Morus indica* L. which can act as a reference for the assembling other closely related plant mitochondrial genomes. The assembled Mulberry mit-genome's sequence dataset can be a pivotal resource for plant molecular breeders, biologists, geneticists and plant scientists.

We also provide a new, rapid procedure for plant mitochondrial genome sequencing and assembly using the Roche/454 GS FLX platform. Plant cells can contain multiple copies of the organellar genomes, and there is a significant correlation between the depth of sequence reads in contigs and the number of copies of the genome. Without isolating organellar DNA from the mixture of nuclear and organellar DNA for sequencing, we retrospectively extracted assembled contigs of mitochondrial sequences from the whole genome Roche 454 data. Moreover, the contig connection graph property of Newbler (a platform-specific sequence assembler) ensures an efficient final assembly. Using this procedure, we assembled a near complete draft mitochondrial genome *Morus indica*, with high fidelity.

The copy number difference between organellar and nuclear DNA is independent of the sequencing platform. Therefore, this procedure can be extended to other platforms with low coverage genome sequencing, such as the Illumina HiSeq platform.

In addition, our strategy is also very useful for plant sequencing projects when an adequate coverage has not been reached, but a data quality assessment is required. For example, our procedure could be extended to cost-efficient 454 sequencing data from a single lane or less. The methodology used will provide an unambiguous way to assemble mitochondria/choloroplast genome from a single lane 454 data which is rich in organelles. This will significantly reduce the cost of data-acquisition for the assembly of organellar genomes. Therefore, we are confident that our efficient and direct procedure will prove useful for further organellar genome sequencing and assembly.

CHAPTER #7

FUTURE PROSPECTS

We present the near-complete draft mitochondrial genome of mulberry *Morus indica* L. The sequencing data was single-end 2 lane data of mulberry_parent_1 and mulberry_parent_2 derived from 454 GS FLX Platform. The 27 Contigs assembled from the pooled_Mulberry data-set are not ordered or linked to form a contiguous scaffold. The genome-finishing of the draft mit-genome of mulberry can be done by designing primers for the contigs which are neatly connected to each other in the De-Bruijn graph. Primers can also be designed for Contigs connected by spanning gene-features. The validated contig connection will provide contiguous contigs or the scaffolds. The scaffolds could be used as a reference for completing the master circle of mulberry mitochondrial genome.

To validate the final assembly we need to incorporate other types of data or experiments to ensure contig connections among the contigs. Scaffolding can be made easier, if we add mate-pair NGS data from the same or different platform. Once, the mate-pair data is available, it can be reassembled into scaffolds and PCR reactions can be done to fill the gaps between the scaffolds. The Contigs initially assembled by our approach can act as a reference sequence for the re-assembling of the mate-pair NGS data of mulberry. Since a reference draft mit-genome has been produced by our procedure, the mate-pair sequencing of mulberry doesn’t require the isolation of mitocchondrial genome. The mitochondrial scaffolds can be assembled with the help of reference mitochondrial contigs available. Closure of gaps can be followed by PCR and a master circle of mulberry mitochondrial genome can be obtained.

The feature-blocks of the reference contigs contain information about the gene-content, tRNA and junk DNA present in the contigs. These can serve as minimal genome survey sequences (GSS) for the detection of SNP and SSR markers. Once, the contig order is known, the feature-blocks could be used to estimate the functional gene order in the mit-genome of mulberry.

CHAPTER #8

REFERENCES

Mulberry Silk. (2012, October). Retrieved from Central Silk Board-Govt of India: http://www.csb.gov.in/

Alverson. (2010). Insights into the evolution of mitochondrial genome size. *Molecular Biology Evolution*, 27, 1436-1448.

Alverson AJ, W. X. (2010). Insights into the evolution of mitochondrial genome size from complete sequences of Citrullus lanatus and Cucurbita pepo (Cucurbitaceae). *Molecular Biological Evolution*, 27: 1436–1448.

Alverson AJ, W. X. (2010). Insights into the Evolution of Mitochondrial Genome Size from Complete Sequences of Citrullus lanatus and Cucurbita pepo (Cucurbitaceae). *Molecular Biology Evolution*, 27:1436-1448.

Archibald J, R. T. (2010). Gene transfer: anything goes in plant mitochondria. *BMC Biology*, 8:147.

Baker, M. (2012). De novo genome assembly: what every biologist should know. *Nature Methods*, 9, 333–337.

Balzer, S. (2010). Characteristics of 454 pyrosequencing data—enabling realistic simulation with flowsim. *Bioinformatics*, 26 (18): i420-i425.

Beuf KD, S. J. (2012). Improved base-calling and quality scores for 454 sequencing based on a Hurdle Poisson model. *BMC Bioinformatics*, 13:303.

Bondada Andallua, V. S. (2001). Effect of mulberry (Morus indica L.) therapy on plasma and erythrocyte membrane lipids in patients with type 2 diabetes. *Clinica Chimica Acta*, 314: 47-53.

Chang S, Y. T. (2011). Mitochondrial genome sequencing helps show the evolutionary mechanism of mitochondrial genome formation in Brassica. *BMC Genomics*, 12:497.

Cock, P. J. (2009). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acid Research*, 38(6): 1767–1771.

Cui P, L. H. (2009). A complete mitochondrial genome of wheat (Triticum aestivum cv. Chinese Yumai), and fast evolving mitochondrial genes in higher plants. *Journal of Genetics*, 88(3):299-307.

Cui Z, L. Y. (2009). The complete mitochondrial genome of the large yellow croaker, Larimichthys crocea (Perciformes, Sciaenidae): Unusual features of its control region and the phylogenetic position of the Sciaenidae. *Gene*, 432:33-43.

Datta, R. (2012). Mulberry cultivation and utilization in India. Srirampura, Mysore, India.

DC, L. (2006). The mitochondrial compartment. *Journal of Experimental Botany*, 57: 1225–1243.

DX, H. (2003). Potential mechanisms of cancer chemoprevention by anthocyanins. *Current Molecular medicine*, 3 (2):149–59.

Elaine Mardis, J. M. (2002). What is Finished, and Why Does it Matter. *Genome Research*, 12: 669-671.

Eren AM, V. J. (2013). A Filtering Method to Generate High Quality Short Reads Using Illumina Paired-End Technology. *Plos One*, 8(6):e66643.

Ewing B, G. P. (1998). Base-calling of automated sequencer traces using phred. II. Error probabilities. *Genome Research*, 8 (3): 186–194.

Flxlex. (2010). *How Newbler works.* Retrieved from wordpress.com: http://contig.wordpress.com/2010/02/09/how-newbler-works/

Gomez-Alvarez V, T. T. (2009). Systematic artifacts in metagenomes from complex microbial communities. *International Society for microbial ecology*, 3:1314-1317.

Góngora-Castillo E, B. C. (2013). Bioinformatics challenges in de novo transcriptome assembly using short read sequences in the absence of a reference genome sequence. *Nature Product Reports*, 30(4):490-500.

Gunnarsdóttir ED, L. M. (2011). High-throughput sequencing of complete human mtDNA genomes from the Philippines. *Genome Research*, 21:1-11.

Gupta, N., & Stephen Tanner, N. J. (2009). "Whole proteome analysis of post-translational modifications: applications of mass-spectrometry for proteogenomic annotation". *Genome Research*, 17 (9): 1362–1377.

Jex AR, H. R. (2010). An integrated pipeline for next-generation sequencing and annotation of mitochondrial genomes. *Nucleic Acid Research*, 38:522-533.

JM, S. (2012). Morus alba L. . *Plant Protection and Production* , 33: 67-72.

Jun Zhang, R. C. (2011). The impact of next-generation sequencing on genomics. *Journal of Genetics and Genomics*, 38(3): 95–109.

Keeling PJ, P. J. (2008). Horizontal gene transfer in eukaryotic evolution. *Nature Review Genetics*, 9: 605–618.

Kleine T, M. U. (2009). DNA transfer from organelles to the nucleus: the idiosyncratic genetics of endosymbiosis. *Annual Review Plant Biology*, 60:115-138.

Krasileva KV, B. V. (2013). Separating homeologs by phasing in the tetraploid wheat transcriptome. *Genome Biology*, 14(6):R66.

Kubo T, M. T. (2007). Organization and variation of angiosperm mitochondrial genome. *Physiology Plantarum*, 129:6-13.

Kubo T, N. K. (2008). Angiosperm mitochondrial genomes and mutations. *Mitochondrion*, 8:5-14.

Legkari. (2010). Next Generation Sequencing Technologies: 454 Pyro-Sequencing. *Biotech Articles*, 30:11.

Li, R., & Hongmei Zhu, J. R. (2010). De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20 (2): 265–272.

Lima J, C. L. (2012). A Scheduling Algorithm for Computational Grids that Minimizes Centralized Processing in Genome Assembly of Next-Generation Sequencing Data. *Frontiers in Genetics*, 10.3389.

Liu X, X. G. (2004). Quantification and Purification of Mulberry Anthocyanins with Macroporous Resins. *Journal of Biomedicine and Biotechnology*, 5: 326-331.

Mackenzie S, M. L. (1999). Higher plant mitochondria. *Plant Cell*, 11: 571–586.

Massimo Iorizzo, D. S. (2012). De novo assembly of the carrot mitochondrial genome using next generation sequencing of whole genomic DNA provides first evidence of DNA transfer into an angiosperm plastid genome. *BMC Plant Biology*, 12:61.

Matus JT, L. R. (2009). Post-veraison sunlight exposure induces MYB-mediated transcriptional regulation of anthocyanin and flavonol synthesis in berry skins of Vitis vinifera. *Journal of experimental botany*, 60 (3): 853–67.

Michalovova M, V. B. (2013). Analysis of plastid and mitochondrial DNA insertions in the nucleus (NUPTs and NUMTs) of six plant species: size, relative age and chromosomal localization. *Heredity*, 10:67-72.

Ogihara Y, Y. Y. (2005). Structural dynamics of cereal mitochondrial genomes as revealed by complete nucleotide sequencing of the wheat mitochondrial genome. *Nucleic Acid Research*, 33:6235-6250.

Ombrello. (2012). *The mulberry tree and its silkworm connection.* Cranford, NJ: Department of Biology, Union County College.

Pettersson E, L. J. (2009). Generations of Sequencing Technologies. *Genomics*, 93:105-111.

Pevsner, J. (2009). Bioinformatics and functional genomics. NJ: Wiley-Blackwell.

Phillip E C Compeau, P. A. (2011). How to apply de Bruijn graphs to genome assembly. *Nature Biotechnology*, 28(11): 45-59.

Reis-Filho, J. S. (2009). Next Generation Sequencing. *Breast Cancer Research*, 11(Suppl 3):S12.

Richardson AO, P. J. (2007). Horizontal gene transfer in plants. *Journal of Experimental Biology*, 58:1-9.

S, H. (2007). Accuracy and quality of massively parallel DNA pyrosequencing. *Genome Biology*, 8:R143.

Schmieder, R. (2011). Quality control and preprocessing of metagenomic datasets. *Bioinformatics Advance Access*, 23(1): 5-7.

Schuster W, B. A. (1994). The Plant Mitochondrial Genome - Physical Structure, Information-Content, RNA Editing, and Gene Migration to the Nucleus. *Annual Review Plant Physiology and Plant Molecular Biology*, 45:61-78.

Sciences, 4. L. (n.d.). How is Genome Sequencing done? Branford, CT: 454 Life Sciences Corporation.

Shedko SV, M. I. (2013). Complete mitochondrial genome of the endangered Sakhalin taimen Parahucho perryi (Salmoniformes, Salmonidae). *Mitochondrial DNA*.

Sloan DB, A. A. (2010). Extensive loss of translational genes in the structurally dynamic mitochondrial genome of the angiosperm Silene latifolia. *BMC Evolutionary Biology*, 10: 274.

Stein, L. (2001). Genome Annotation: From Sequence to Biology. *Nature Reviews Genetics.*, 2 (7): 493–503.

T, L. (2002). A review of DNA sequencing techniques. *Quarterly Reviews of Biophysics*, 169-200.

Wang XC, Z. Q. (2013). Global transcriptome profiles of Camellia sinensis during cold acclimation. *BMC Genomics*, 14(1):415.

Wang, W. (2012). The Mitochondrial Genome of an Aquatic Plant, Spirodela polyrhiza. *Plos One*, 7: 10-18.

Woodson JD, C. J. (2008). Coordination of gene expression between organellar and nuclear genomes. *Nature Reviews Genetics*, 9:383-395.

Yang H, T. Y. (2013). Draft Genome Sequence, and a Sequence-Defined Genetic Linkage Map of the Legume Crop Species Lupinus angustifolius L. *Plos One*, 8(5):e64799.

Yang M, Z. X.-M. (2010). The complete chloroplast genome sequence of date palm (Phoenix dactylifera L.). *Plos One*, 5:e12762.

CHAPTER #9

APPENDIX

Fasta_qual_fastq.pl: A perl script which combines FASTA file and QUAL file of raw reads to make a file of FASTQ file.

```
#!/usr/bin/perl

use warnings;
use strict;
use File::Basename;

my $inFasta = $ARGV[0];
my $baseName = basename($inFasta, qw/.fasta .fna/);
my $inQual = $baseName . ".qual";
my $outFastq = $baseName . ".fastq";

my %seqs;

$/ = ">";

open (FASTA, "<$inFasta");
my $junk = (<FASTA>);

while (my $frecord = <FASTA>) {
	chomp $frecord;
	my ($fdef, @seqLines) = split /\n/, $frecord;
	my $seq = join '', @seqLines;
	$seqs{$fdef} = $seq;
}

close FASTA;

open (QUAL, "<$inQual");
$junk = <QUAL>;
open (FASTQ, ">$outFastq");

while (my $qrecord = <QUAL>) {
	chomp $qrecord;
	my ($qdef, @qualLines) = split /\n/, $qrecord;
	my $qualString = join ' ', @qualLines;
	my @quals = split / /, $qualString;
	print FASTQ "@","$qdef\n";
	print FASTQ "$seqs{$qdef}\n";
	print FASTQ "+\n";
	foreach my $qual (@quals) {
		print FASTQ chr($qual + 33);
	}
	print FASTQ "\n";
}

close QUAL;
close FASTQ;
```

Trim_homopolymer.pl: An in-house perl script to trim homo-polymer errors.

```
#!/usr/bin/perl

use warnings;
use strict;
use File::Basename;
```

```
open(input_file,”>>mul_1.fasta”);    /Open the FASTA file to be processed

qx (awk '/^>/{$0=(NR>1)?RS $0:$0;ORS=RS}!/>/{ORS=""}END{printf "\n"}1' input_file);

/ A shell command to remove new_lines from FASTA sequences/

Open(output_file,”>>processed_file.fna”);

qx(awk ‘{if(length>=700) print}’ input_file > output_file);

qx(sed -n '/AAAAAAA/!p'|sed -n '/TTTTTTTTT/!p'|sed -n '/CCCCCCCCC/!p'|sed -n
'/GGGGGGG/!p'|sed -n '/NNNNNNNN/!p' output_file);
```

NewblerMetrices.pl: A perl script for extracting metrices of Newbler Assembly Run

```
#! /usr/bin/perl

# Makes a tab-separated file from
# the 454NewblerMetrics.txt file
# from a newbler assembly
# tested on newbler v 2.3 and 2.5.3
# on both shotgun, shotgun + paired end and transcriptome assemblies
# by Lex Nederbragt, lex.nederbragt@bio.uio.no
# Release notes:
# version 1, May 2011
#	first release
# Version 1.1, September 2011
#	fixed change from pairDistanceAvg to computedPairDistanceAvg in newbler 2.6
# current version:
# version 1.2, September 2012
#	fixed a small erroneous tab in the output

# run as
# newblermetrics.pl 454Newblermetrics.txt
# newblermetrics.pl /path/to/454Newblermetrics.txt

# or
# perl newblermetrics.pl 454Newblermetrics.txt
# perl newblermetrics.pl /path/to/454Newblermetrics.txt



use strict;
use warnings;

#################################
# variables
#################################

my $metrics;		# holds the entire 454NewblerMetrics.txt file
my $section = "";	# section of the file, e.g. rundata
my $level2;			# all lines with a single tab
my $level3;			# all lines with two tabs
my %metrics = ();	# hash with extracted results
my @lib_names;			# names for paired end libraries

#################################
# test inputfile
#################################

# file given?
if (!$ARGV[0]){
	print STDERR "Please add a 454Newblermetrics.txt file on the command line...\n";
	exit[0];
}

# file exists and is a file?
unless (-e $ARGV[0] && -f $ARGV[0]){
	print STDERR "File '$ARGV[0]' does not exist or is not a file...\n";
	exit[0];
}
```

```
# file can be opened?
open METRICS , "<$ARGV[0]" or die "File '$ARGV[0]' can't be opened:\n$!";


# read in the file
$/=undef; # set the record to 'slurp' the file
$metrics = <METRICS>;

# correct file type?
unless ($metrics =~ /454 Life Sciences Corporation/ && $metrics =~ /Newbler Metrics
Results/ ){
        print STDERR "File '$ARGV[0]' does not appear to be a 454NewblerMetrics
file...\n";
        exit[0];
}

if ($metrics =~ /Date of Mapping: /){
        print STDERR "The script currently only works on 454NewblerMetrics.txt files
from newbler assemblies,
not from mappings (gsMapper, runmapping)...\n";
        exit[0];
}


################################
# process inputfile
################################

foreach (split /\n/, $metrics){
        $section = $_ if /^\w/;

        # runData/pairedReadData
        if ($section eq "runData" || $section eq "pairedReadData"){
                if( /(numberOf.+) = (\d+), (\d+);/){
                        $metrics{'reads'}{"$1Raw"}+=$2;
                        $metrics{'reads'}{"$1Trimmed"}+=$3;
                }
        next;
        }

        # consensusResults section
        if ($section eq "consensusResults"){
                # type of metric/status is on level 2
                $level2 = $1 if /^\t(\w+)/;

                # pairedReadStatus
                # have to take care of both
                # newbler 2.5.3: pairDistanceAvg (or ...Dev)
                # newbler 2.6: computedPairDistanceAvg (or ...Dev)
                push @lib_names, $1 if/libraryName\s+= "(.+)";/;
```

```
			$metrics{$lib_names[-1]}{$1}=$2 if /(airDistance...)\s+= ([0-9\.]+);/;

			# other metrics
			$metrics{$level2}{$1}=$2 if /^\t\t(\w+)\s+= ([0-9\.]+)/;
			next;
		}
}
$/="\n"; # reset the record separator


#################################
# between versions fixes
#################################

# fix spelling mistake 'bug' from newbler 2.3
if ($metrics{'isotigMetrics'}{'numberWithOneConitg'}){
	$metrics{'isotigMetrics'}{'numberWithOneContig'} =
$metrics{'isotigMetrics'}{'numberWithOneConitg'}
}

#################################
# output
#################################

print "Input\n";
print "Number of reads\t", $metrics{'reads'}{'numberOfReadsRaw'}, "\n";
print "Number of bases\t", $metrics{'reads'}{'numberOfBasesRaw'}, "\n";
print "Number of reads trimmed\t", $metrics{'reads'}{'numberOfReadsTrimmed'}, "\t",
	sprintf ("%.1f",	100*$metrics{'reads'}{'numberOfReadsTrimmed'}/
						$metrics{'reads'}{'numberOfReadsRaw'}),
"%\n";
print "Number of bases trimmed\t", $metrics{'reads'}{'numberOfBasesTrimmed'}, "\t",
	sprintf ("%.1f",	100*$metrics{'reads'}{'numberOfBasesTrimmed'}/
						$metrics{'reads'}{'numberOfBasesRaw'}),
"%\n";
print "\n";

print "Consensus results\n";
print "Number of reads assembled\t", $metrics{'readStatus'}{'numberAssembled'},"\t",
	(sprintf "%.1f",	100*$metrics{'readStatus'}{'numberAssembled'}/

	$metrics{'reads'}{'numberOfReadsTrimmed'})."%\n";
print "Number partial\t", $metrics{'readStatus'}{'numberPartial'},"\t",
	(sprintf "%.1f",	100*$metrics{'readStatus'}{'numberPartial'}/

	$metrics{'reads'}{'numberOfReadsTrimmed'})."%\n";
print "Number singleton\t", $metrics{'readStatus'}{'numberSingleton'},"\t",
	(sprintf "%.1f",	100*$metrics{'readStatus'}{'numberSingleton'}/

	$metrics{'reads'}{'numberOfReadsTrimmed'})."%\n";
```

```
print "Number repeat\t", $metrics{'readStatus'}{'numberRepeat'},"\t",
        (sprintf "%.1f",        100*$metrics{'readStatus'}{'numberRepeat'}/

        $metrics{'reads'}{'numberOfReadsTrimmed'})."%\n";
print "Number outlier\t", $metrics{'readStatus'}{'numberOutlier'},"\t",
        (sprintf "%.1f",        100*$metrics{'readStatus'}{'numberOutlier'}/

        $metrics{'reads'}{'numberOfReadsTrimmed'})."%\n";
print "Number too short\t", $metrics{'readStatus'}{'numberTooShort'},"\t",
        (sprintf "%.1f",        100*$metrics{'readStatus'}{'numberTooShort'}/

        $metrics{'reads'}{'numberOfReadsTrimmed'})."%\n";

print "\n";


if (exists $metrics{'scaffoldMetrics'}{'numberOfScaffolds'}){
        print "Scaffold Metrics\n";
        print "Number of scaffolds\t", $metrics{'scaffoldMetrics'}{'numberOfScaffolds'},
"\n";
        print "Number of bases\t", $metrics{'scaffoldMetrics'}{'numberOfBases'}, "\n";
        print "Average scaffold size\t", $metrics{'scaffoldMetrics'}{'avgScaffoldSize'}, "\n";
        print "N50 scaffold size\t", $metrics{'scaffoldMetrics'}{'N50ScaffoldSize'}, "\n";
        print "Largest scaffold size\t", $metrics{'scaffoldMetrics'}{'largestScaffoldSize'},
"\n";
        print "\n";
}

if (exists $metrics{'isogroupMetrics'}{'numberOfIsogroups'}){
        print "Isogroup Metrics\n";
        print "Number of isogroups\t", $metrics{'isogroupMetrics'}{'numberOfIsogroups'},
"\n";
        print "Average contig count\t", $metrics{'isogroupMetrics'}{'avgContigCnt'}, "\n";
        print "Largest contig count\t", $metrics{'isogroupMetrics'}{'largestContigCnt'}, "\n";
        print "Number with one contig\t",
$metrics{'isogroupMetrics'}{'numberWithOneContig'}, "\n\n";
        print "Average isotig count\t", $metrics{'isogroupMetrics'}{'avgIsotigCnt'}, "\n";
        print "Largest isotig count\t", $metrics{'isogroupMetrics'}{'largestIsotigCnt'}, "\n";
        print "Number with one isotig\t",
$metrics{'isogroupMetrics'}{'numberWithOneIsotig'}, "\n\n";

        print "Isotig Metrics\n";
        print "Number of Isotigs\t", $metrics{'isotigMetrics'}{'numberOfIsotigs'}, "\n";
        print "Average contig count\t", $metrics{'isotigMetrics'}{'avgContigCnt'}, "\n";
        print "Largest contig count\t", $metrics{'isotigMetrics'}{'largestContigCnt'}, "\n";
        print "Number with one contig\t",
$metrics{'isotigMetrics'}{'numberWithOneContig'}, "\n\n";
        print "Number of bases\t", $metrics{'isotigMetrics'}{'numberOfBases'}, "\n";
        print "Average isotig size\t", $metrics{'isotigMetrics'}{'avgIsotigSize'}, "\n";
        print "N50 isotig size\t", $metrics{'isotigMetrics'}{'N50IsotigSize'}, "\n";
```

```
	print "Largest isotig\t", $metrics{'isotigMetrics'}{'largestIsotigSize'}, "\n\n";
}

print "Large Contig Metrics\n";
print "Number of contigs\t", $metrics{'largeContigMetrics'}{'numberOfContigs'}, "\n";
print "Number of bases\t", $metrics{'largeContigMetrics'}{'numberOfBases'}, "\n";
print "Average contig size\t", $metrics{'largeContigMetrics'}{'avgContigSize'}, "\n";
print "N50 contig size\t", $metrics{'largeContigMetrics'}{'N50ContigSize'}, "\n";
print "Largest contig size\t", $metrics{'largeContigMetrics'}{'largestContigSize'}, "\n";
print "Q40 plus bases\t", $metrics{'largeContigMetrics'}{'Q40PlusBases'}, "\t",
	(sprintf "%.2f",	(100*$metrics{'largeContigMetrics'}{'Q40PlusBases'}/

	$metrics{'largeContigMetrics'}{'numberOfBases'})),"%\n";
print "\n";
print "All Contig Metrics\n";
print "Number of contigs\t", $metrics{'allContigMetrics'}{'numberOfContigs'}, "\n";
print "Number of bases\t", $metrics{'allContigMetrics'}{'numberOfBases'}, "\n";
print "Average contig size\t",
	(sprintf "%.0f",	$metrics{'allContigMetrics'}{'numberOfBases'}/

	$metrics{'allContigMetrics'}{'numberOfContigs'})."\n";
print "\n";

if (exists $metrics{'scaffoldMetrics'}{'numberOfScaffolds'}){
	print "Library Pair distance average (bp)\n";
	foreach my $lib_name (sort @lib_names){
		print "$lib_name\t",$metrics{$lib_name}{'airDistanceAvg'},"\n";}}
```

bb.454Contignet.pl: A perl script for establishing Contig-Connections.

```
#!/usr/bin/perl
#-----------------------------------------------------------------------#
#		Author: Douglas Senalik dsenalik@wisc.edu		#
#  http://www.vcru.wisc.edu/simonlab/sdata/software/index.html#contignet  #
#	Modified by: Simon Gladman simon.gladman@csiro.au		#
#				2011				#
#-----------------------------------------------------------------------#
# "Black Box" program series
=bb
Create a network of all interconnected 454 contigs
=cut bb
use strict;
use warnings;
use Getopt::Long;	# for getting command line parameters

# 1.0.1 - Dec 29, 2010
# 1.0.2 - Feb 15, 2011 - removed extra line that prevented count of nodes from working
# 1.0.3 - Mar 12, 2011 - Support paired end data output files, add "@" prefix for list files,
#			add --nospline and --overlapmode parameters
# 1.0.3-Simon - May 18, 2011 - Added pseudo-links formed by paired end information
instead of just
#						newbler links
```

```
# 1.0.4 - September 2, 2011
#       Allow "+" or "-" in contig numbers, but it is ignored
#       Add parameter to specify output image type
#       Correct error in --tag help description
#       Add --label as a synonym for --tag
#       Fix bug in dead end and recursion limit labeling
# 1.0.5 - October 18, 2011
#       Allow keeping graphviz command file
#       change default overlap mode to "false" (it was "none" before)
#       to have same default output with newest version of neato
#       make scaffold connections optional, add --scaffold to use scaffold connections
# 1.0.6 - November 4, 2011
#       Filter out null contigs, excludes, etc. from command line
#       Optional ABySS-Explorer output file added
# 1.0.7 - May 4, 2012
#       Text output data file is now sorted by contig number
#       change the --scaffold parameter to --pairlinks,
#         since I may use --scaffold in the future for the actual scaffold section
#       Show paired end links and labels ( --pairlinks ) in a different color
#       Add support for flowthrough links ( --flowthrough ) in a third color
#       Add support for flowbetween links ( --flowbetween ) in a fourth color
#       Add paired end support to ABySS-Explorer ouput, and use exclusively the
#         new ABySS-Explorer 1.3.0 .dot format
my $version          = "1.0.7";



###############################################################
# configuration variables
###############################################################
my $bpabbreviation      = "nt";    # set to b.p., bp, nt, or even a null string, as you prefer
my $defaultmaxlevel     = 2;
my $debuglimit          = 1000;    # how long print debug messages while extending
my $numdigits           = 5;       # contig numbers filled out with leading zeroes to this length
my $scalefactor         = 0.01;    # convert b.p. to graphviz length by multiplying by this value
my $defaultoverlapmode  = "false"; # neato overlap mode in graph section, "true" to disable
my $defaultouttype      = "png";   # default image type ( passed to neato )
my $dotheaderid         = "adj";   # ABySS-Explorer .dot file header id
my $contiggraphtxt      = "454ContigGraph.txt";  # this is defined by gsAssembler/Newbler
my $allcontigsfna       = "454AllContigs.fna";   # this is defined by gsAssembler/Newbler
my $defaultminflowbetween = 1;

# default color definitions
my $deadendcolor        = "tomato";
my $recursionlimitcolor  = "gold";
my $normallinkcolor     = "";          # leave blank for the default color of black
my $normalfontcolor     = $normallinkcolor;
my $forcedlinkcolor     = "red";       # links from --force parameter
my $forcedfontcolor     = $forcedlinkcolor;
my $pairedendlinkcolor   = "dodgerblue";  # links from paired end information
my $pairedendfontcolor   = $pairedendlinkcolor;
my $flowbetweenlinkcolor = "purple";      # links from flowthrough "F" information
my $flowbetweenfontcolor = $flowbetweenlinkcolor;
my $flowthroughlinkcolor = "forestgreen"; # links from flowthrough "I" information
my $flowthroughfontcolor = $flowthroughlinkcolor;
my $abyssk = 1;       # we don't use kmers, so set this to 1 so coverage will be direct
conversion
my $abyssedge = 0;    # negative kmer minus 1, thus zero
my $abyssevalue = 0;  # we don't have a value for this, so just use zero always



#################################################################
# global variables
#################################################################
my $ansiup      = "\033[1A";  # terminal control
(my $prognopath = $0) =~ s/^.*[\/\\]//;
my @contiglen   = ();   # contig length
my @contigcov   = ();   # contig average coverage
my %ends        = ();   # key is contig . "3'" or "5'" or "0'", with optional "p", "f", or "i",
                        # values are @[ contig, 3'|5', #reads, flag ]
my %pairs                  = ();     #used for temporary storage of paired end information until
link ends can be sorted.. [Simon Gladman - 2011]
my %deadends    = ();   # key is contig, value is # of ends with reads extending ( so 1 = dead
end )
my %minrl       = ();   # key is contig, value is lowest recursion level seen for this contig
my %seen        = ();   # key is contig, value is >=1 if we already traversed, undefined if not
                        #   actual value reflects recursion level when seen
my %edgeseen    = ();   # key is contig 3'or5' contig 3'or5', value = 1 or undefined
my %taghash     = ();   # key is contig, value is array of tags
my %colorhash   = ();   # key is contig, value is array of colors
my %excludehash = ();   # key is contig, value is 1 to exclude
my %inverthash  = ();   # key is contig, value is 1 to exclude
my %data        = ();   # subset of %ends that will end up in graph
my %flowdata    = ();   # store flow through read data here
my $extensions  = 0;    # number of auto-extensions so far (for --extend)
my @extarr      = ();   # extensions will apply only to supplied contigs, not auto-added ones
my $infilename  = "";   # 454ContigGraph.txt path and name
my @listofexcl  = ();   # list of excluded contigs, --listexcluded turns this on
my $deletecmdfile = 1;  # becomes 0 if a command file name is explicitly specified
my $returncode  = 0;    # return code of this program



#################################################################
# command line parameters
#################################################################
my $indirname   = "";   # input file name
my $outfilename = "";   # output file name
my $outtype     = $defaultouttype;  # output image file format
my $cmdfilename = "";   # graphviz .dot language command file name
my $imgfilename = "";   # image file name
my $outfastaname = "";  # create FASTA file of contigs in output
my $level       = $defaultmaxlevel;    # maximum number of levels of recursion
my $boldabove   = 0;
my @contig      = ();   # starting contig(s)
my @tag         = ();   # tag certain contigs
my @color       = ();   # color certain contigs
my @exclude     = ();   # never go into these contigs (e.g. repeat regions)
my @invert      = ();   # contigs to plot backwards
my $len         = 1;    # neato len parameter
my $listexcluded = 0;   # list contigs that have been excluded
my $extend      = 0;    # auto extend for best contig
my @forcelink   = ();   # force links where none may exist
my $showbp      = 0;    # include length in b.p. in graph
my $showcov     = 0;    # include contig average coverage in graph
my $lowlimit    = 0;    # ignore links with read limit < this
my $highlimit   = 0;    # ignore links with read number > this
my $nolabel     = 0;    # disable dead end and recursion limit labelling
my $overlapmode = $defaultoverlapmode;
my $nospline    = 0;    # to disable splines
my $scaffold    = 0;    # to enable scaffold connection information
my $pairlinks   = 0;    # to enable paired end read connection information
my $flowthrough = 0;    # to enable flowthrough connection information
my $flowbetween;        # to enable flow between connection information, has optional value
also
my $alllinks    = 0;    # sets --pairlinks --flowthrough --flowbetween and --scaffold
my $abyssdotfile;       # generate file for use with ABySS-Explorer
my $help        = 0;    # print help and exit
my $quiet       = 0;    # only show errors
my $debug       = 0;    # print extra debugging information
GetOptions (
        "indir=s"        => \$indirname,        # string
        "outfile=s"      => \$outfilename,      # string
        "type=s"         => \$outtype,          # string
        "cmdfile=s"      => \$cmdfilename,      # string
        "imgfile=s"      => \$imgfilename,      # string
        "fastaout=s"     => \$outfastaname,     # string
        "abyssexplorer=s"=> \$abyssdotfile,     # string
        "level=i"        => \$level,            # integer
        "boldabove=i"    => \$boldabove,        # integer
        "contig=s"       => \@contig,           # string array
        "exclude=s"      => \@exclude,          # string array
        "invert=s"       => \@invert,           # string array
        "tag|label=s"    => \@tag,              # string array
        "color=s"        => \@color,            # string array
        "forcelink=s"    => \@forcelink,        # string array
        "len=s"          => \$len,              # real
        "extend=i"       => \$extend,           # integer
        "lowlimit=i"     => \$lowlimit,         # integer
        "highlimit=i"    => \$highlimit,        # integer
        "nolabel"        => \$nolabel,          # flag
        "nospline"       => \$nospline,         # flag
        "pairlinks"      => \$pairlinks,        # flag
        "scaffold"       => \$scaffold,         # flag
        "flowthrough"    => \$flowthrough,      # flag
        "flowbetween:s"  => \$flowbetween,      # flag/string
        "alllinks"       => \$alllinks,         # flag
        "overlapmode=s"  => \$overlapmode,      # string
        "listexcluded"   => \$listexcluded,     # flag
        "showbp|shownt"  => \$showbp,           # flag
        "showcoverage"   => \$showcov,          # flag
        "help"           => \$help,             # flag
        "quiet"          => \$quiet,            # flag
        "debug"          => \$debug);           # flag
# debug implies not quiet
if ( $debug ) { $quiet = 0; }
unless ( ( $indirname ) and ( $outfilename ) and ( scalar @contig ) ) { $help = 1; }

# changing meaning of --scaffold for future use
if ( $scaffold )
  { die "--scaffold has been changed to --pairlinks\n"; }

# $flowbetween is a flag with an optional value, set default value if no value was specified
if ( ( defined $flowbetween ) and ( $flowbetween eq "" ) ) { $flowbetween =
$defaultminflowbetween; }

if ( $alllinks )
  {
    $pairlinks = 1;
    $flowthrough = 1;
    $flowbetween = $defaultminflowbetween;
    $scaffold = 1;  # future use
  }

# allow specification of only the directory
unless ( ( -d $indirname ) or ( $help ) )
  {
    print "Error, specified input directory \"$indirname\" does not exist or is not a directory\n";
    $help = 1;
  }
$infilename = $indirname;
unless ( $infilename =~ m/\/$/ ) { $infilename .= "/"; }
$infilename .= $contiggraphtxt;

# make sure input file exists
unless ( ( -e $infilename ) or ( $help ) )
  {
    print "Error, input file \"$infilename\" does not exist\n";
    $help = 1;
  }

# if no --imgfile, create name based on --outfile
unless ( $imgfilename ) { $imgfilename = $outfilename . "." . $outtype; }
# if no --cmdfile, create name based on --outfile
if ( $cmdfilename )
  { $deletecmdfile = 0; }
else
  { $cmdfilename = $outfilename . ".graphviz"; }
if ( $outfilename =~ m/png$/i ) { warn "You probably do not want to append a .png extension
on --outfile\n"; }

# OBSOLETE
# ABySS-Explorer idiosyncracy
#if ( ( $abyssexplorer ) and ( $abyssexplorer !~ m/-4.adj/ ) )
#  { print "WARNING: ABySS-Explorer versions <= 1.0.1 require that the input file name
ends in \"-4.adj\"\n"; }



###############################################################
# print help screen
###############################################################
if ( $help )
  {
    print "$prognopath  version $version
Required parameters:
  --indir=xxx       path to 454 assembly directory
  --outfile=xxx     output text file of results
  --contig=xxx[,xxx]...
                    one or more starting contig numbers,
                    separated by comma, or multiple --contig
                    parameters may be used. Use just the
                    numeric portion of the contig
Optional parameters:
  --type=xxx        output file format, default is \"$defaultouttype\"
                    ( anything besides \"png\" is experimental )
  --cmdfile=xxx     graphviz command file in .dot language will be created
                    using this name. If not specified, a temporary command
                    file will be created, and it will be deleted when done
  --imgfile=xxx     graph image file will be created with
                    this name. If not specified, will be
                    --outfile with .${outtype} extension added
  --fastaout=xxx    create a FASTA file of all contigs in
                    the output, save in this file
  --abyssexplorer=xxx  Generate a .dot file that can be used for
                    visualization with ABySS-Explorer 1.3.0,
                    http://www.bcgsc.ca/platform/bioinfo/software/abyss-explorer";
#                     for compatibility, the file name must end in \"-4.adj\"
#                     If paired end information is available, and the
#                   --pairlinks parameter is used, corresponding
#                   \"-3.dist\" and \"-contigs.fa\" files will also be created
print "
  --flowthrough     include connection information derived from
                    reads that flow through more than two contigs
  --flowbetween[=x] include connection information derived from
                    reads that flow from one contig into another
                    by default, if the distance value is zero, it will not be
                    shown, the optional value for this parameter is a minimum
                    distance, defaulting to $defaultminflowbetween, set to --flowbetween=0 to show
                    these links also
  --pairlinks       include connection information derived
                    from paired end reads, only applicable for assemblies
                    containing paired end reads
  --alllinks        sets --flowthrough, --flowbetween, and --pairlinks
  --tag=tagname,contig[,contig]...
                    list of 1 or more contigs will be given
                    this tag. Multiple --tag allowed.
                    tagname is a text label that will be shown
                    in the final image, e.g. --tag=\"ATP1,14,34\"
  --label           a synonym for --tag
  --showbp          show length in b.p. in graph
  --shownt          a synonym for --showbp
  --showcoverage    show average contig read coverage in graph
  --color=colorname,contig[,contig]...
                    like --tag, but color the contig.
                    for list of valid color names see
                    http://www.graphviz.org/doc/info/colors.html
  --forcelink=xxx-5:yyy-3   force a link where none exists
                    between specified ends, xxx and yyy are
                    contig numbers
  --level=xxx       maximum recursion level, default=$level
  --boldabove=xxx   lines with read coverage >= this value
                    will be drawn in bold. no default value
  --exclude=xxx[,xxx]...
                    one or contigs to never traverse past,
                    for example a repeated region contig
  --listexcluded    print out a list of which excluded contigs
                    are being ignored
  --invert=xxx[,xxx]...
                    one or more contigs to plot backwards on
                    the graph, i.e. 3' to 5' direction
  --extend=xxx      auto extension for the single best
                    path, value is maximum steps, default=$extend
  --lowlimit=xxx    ignore connections < this number of reads
  --highlimit=xxx   ignore connections > this number of reads
  --len=xxx         len parameter to neato, default=$len
  --nolabel         disable highlighting of dead ends, and limit
                    of recursion contigs
  --overlapmode     neato paramter, default is $overlapmode, one of
```

```
                  none, true, scale
  --nospline        disable spline when edges would overlap
  --help           print this screen
  --quiet          only print error messages
  --debug          print extra debugging information

In place of lists of contigs, you can use \@filename to read in
values for that parameter from a file, e.g. --exclude=\@excl.txt

This program requires that the graphviz program \"neato\" be
available in the default PATH. The graphviz web site is
http://www.graphviz.org/
";
    exit 1;
  } # if ( $help )



##################################################################
# expand --contig lists separated by commas into single array
##################################################################
{
my @tmp = ();
foreach my $acontig (@contig)
  {
    $acontig = expandatprefix ( $acontig );
    push ( @tmp, split ( /\s*[,;]\s*/, $acontig ) );
  }
# cleaning and validation
@contig = ();
foreach my $item ( @tmp )
  {
    $item = expandatprefix ( $item );
    $item =~ s/^0+//;   # remove leading zeroes
    $item =~ s/\s//g;   # remove any white space
    $item =~ s/[\+\-]//g;  # allow "+" or "-" in contig numbers, but it is ignored
    if ( $item =~ m/[^\d]/ ) { die "Error, non-numeric character used for --contig \"$item\"\n";
}
    unless ( $item =~ m/^$/ )  # skip null items
      { push ( @contig, $item ); }
  }
debugmsg ( "Supplied contig list of ".scalar(@contig)." contigs = \"".join ("\" \"",
@contig)."\"" );
}



##################################################################
# expand --tag lists separated by commas into hash of arrays
##################################################################
```

```
{
my $ntags = 0;
foreach my $atag (@tag)
  {
    my @tmp = split ( /\s*[,;]\s*/, $atag );
    my $taglabel = shift ( @tmp );
    foreach my $item ( @tmp )
      {
        $item = expandatprefix ( $item );
        $item =~ s/^0+//;  # remove leading zeroes
        unless ( $item =~ m/^$/ ) # skip null items
          {
            push ( @{$taghash{$item}}, $taglabel );
            $ntags++;
          }
      }
  } # foreach (@tag)
debugmsg ( "Stored ".commify($ntags)." tags" );
}



############################################################
# expand --color lists separated by commas into hash of arrays
############################################################
{
my $ncolors = 0;
foreach my $acolor (@color)
  {
    my @tmp = split ( /\s*[,;]\s*/, $acolor );
    my $colorlabel = shift ( @tmp );
    foreach my $item ( @tmp )
      {
        $item = expandatprefix ( $item );
        $item =~ s/^0+//;  # remove leading zeroes
        $item =~ s/[\-\+]//g;  # remove plus or minus - has no meaning, but this is a convenience
to be compatible with bb.fastareorder
        unless ( $item =~ m/^$/ ) # skip null items
          {
            push ( @{$colorhash{$item}}, $colorlabel );
            $ncolors++;
          }
      }
  } # foreach (@color)
debugmsg ( "Stored ".commify($ncolors)." colors" );
}



############################################################
```

```
# expand --forcelink lists separated by commas into hash of arrays
################################################################
{
my $nforce = 0;
foreach my $aforce (@forcelink)
  {
    my @tmp = split ( /\s*[,;]\s*/, $aforce );
    foreach my $link ( @tmp )
      {
        $link = expandatprefix ( $link );
        my @parts = split ( /\s*[:-]\s*/, $link );
        unless ( scalar @parts == 4 ) { die "Invalid format for --forcelink \"$link\"\n"; }
        foreach ( @parts )
          {
            s/^0+//;   # remove leading zeroes
            s/'//g;    # remove primes (they are optional)
          }
        # add this artificial link to the list

        push ( @{$ends{$parts[0].$parts[1]."'"}}, [ $parts[2], $parts[3]."'", 0 ] );
        push ( @{$ends{$parts[2].$parts[3]."'"}}, [ $parts[0], $parts[1]."'", 0 ] );

      }
    $nforce++;
  } # foreach (@forcelink)
debugmsg ( "Stored ".commify($nforce)." forced links" );
}



################################################################
# convert --exclude lists to a simple hash
################################################################
{
foreach my $aexclude (@exclude)
  {
    foreach my $item ( split ( /\s*[,;]\s*/, $aexclude ) )
      {
        $item = expandatprefix ( $item );
        # cleaning and validation
        $item =~ s/^0+//;   # remove leading zeroes
        $item =~ s/\s//g;   # remove any white space
        unless ( $item =~ m/^$/ ) # skip null items
          {
            if ( $item =~ m/[^\d]/ ) { die "Error, non-numeric character used for --exclude
\"$item\"\n"; }
            $excludehash{$item} = 1;
          }
      }
  }
debugmsg ( "Stored ".commify(scalar keys %excludehash)." exclude contigs" );
}



###############################################################
# convert --invert lists to a simple hash
###############################################################
{
foreach my $ainvert (@invert)
  {
    $ainvert = expandatprefix ( $ainvert );
    foreach my $item ( split ( /\s*[,;]\s*/, $ainvert ) )
      {
        $item = expandatprefix ( $item );
        # cleaning and validation
        $item =~ s/^0+//;   # remove leading zeroes
        $item =~ s/\s//g;   # remove any white space
        unless ( $item =~ m/^$/ ) # skip null items
          {
            if ( $item =~ m/[^\d]/ ) { die "Error, non-numeric character used for --invert
\"$item\"\n"; }
            $inverthash{$item} = 1;
          }
      }
  }
debugmsg ( "Stored ".commify(scalar keys %inverthash)." invert contigs" );
}



###############################################################
# parse 454ContigGraph.txt
###############################################################
# sample content: refer to this excellent description for more info:
# http://contig.wordpress.com/2010/04/13/newbler-output-iii-the-454contiggraph-txt-file
#1       contig00001    588     2.6
#2       contig00002    1072    6.8
#3       contig00003    644     4.1
#...
#C      7      3'     14770   5'     3
#C      12     3'     14824   5'     5
#C      12     3'     52148   5'     4
#...
#S      1      84148   1:+;2:+;gapOneNoEdges:186;3:+;4:+;5:+;6:+
#S      2      17530   7:+
#S      3      25222   8:+;9:-;10:-
;11:+;12:+;14:+;gapMultiEdges:4733;15:+;gapMultiEdges:4241;16:+;17:+;19:+
#...
```

```
#I      7
aCAACaTTATCATTGtATTTatATTCcTGTTtGAGATACGTGTGGACAGAGAATGTTG
GTTTTTTGGACTAGAATCGGATTTATCATTATTATAATGT...
#I      48      AGTTCGTCCTGGACGACTTGAGTT        11:19543-5'..16159-5';6:19543-
5'..60104-5'
#I      50
GGgTAATAGTTGACCGTCTTACGAAATtGGCACATTTTCTTCCAATTAACGAGAAA
TCTTCggtAGACAGACTAGTTCATATGTATGTGCGtGAAATC...
#...
#F      7       -       14770/3/0.0
#F      8       56895/6/0.0;54006/2/231.5       -
#F      12      -       14824/5/0.0;52148/4/0.0
#...
#P      1
10130/2/0.0;33848/3/0.0;34537/2/397.0;25104/2/679.5;15/170/698.4;6/2/1600.0;209/175/235
2.0;9364/2/3929.5;19/89/4351.2;16/128/5345.5;17/25/5380.7;14/15/603...
#P      2       1/284/927.3;20341/2/7324.0;23108/2/8174.5
4/210/2787.7;3/22/3918.7;39/2/4345.5
#P      3       4/183/1037.9;24637/4/6940.0
1/156/3004.0;2/22/3918.7;1675/2/4906.0;29922/3/5867.0;97/2/6139.5;7360/2/7464.0
{
my $lines = 0;
my $clines = 0;
my $ilines = 0;
my $flines = 0;
my $slines = 0;
my $plines = 0;
my $endsstored = 0;
open ( my $INF, "<", $infilename ) or die ( "Error opening input file \"$infilename\": $!\n" );
while ( my $aline = <$INF> )
  {
    $lines++;

    # progress indicator
    unless ( $quiet ) { if ( ( $lines % 1000 ) == 0 ) { print commify($lines), "\n", $ansiup; } }

    $aline =~ s/[\r\n]//g;
    my @cols = split ( /\t/, $aline );


    if ( $cols[0] eq "C" )
      {
        $clines++;
        # store both orientations in hash
        my $keep = 1;
        if ( $cols[5] < $lowlimit ) { $keep = 0; }
        if ( ( $highlimit ) and ( $cols[5] > $highlimit ) ) { $keep = 0; }
        if ( $keep )
          {
```

```
          # store data from each end, the three columns are [0]=nextcontig
[1]=5'|3'|5'p|3'p|5'f|3'f|5'i... [2]=readnum
          push ( @{$ends{$cols[1].$cols[2]}}, [ $cols[3], $cols[4], $cols[5] ] );
          # and store reciprocal end
          push ( @{$ends{$cols[3].$cols[4]}}, [ $cols[1], $cols[2], $cols[5] ] );
          $endsstored+=2;
        } # if $keep
      if ( $endsstored < $debuglimit )
        {
          my $txt = $keep?"    Storing":"Not storing";
          debugmsg ( "$txt ends: \$ends{$cols[1]$cols[2]}  [ $cols[3], $cols[4], $cols[5] ];" );
          debugmsg ( "                : \$ends{$cols[3]$cols[4]}  [ $cols[1], $cols[2], $cols[5] ];" );
        }
    } # "C"


  elsif ( $cols[0] eq "I" )  # flowthrough information
    {
      $ilines++;
      if ( $flowthrough )
        {
          # cols[1] is contig number
          # cols[2] is contig sequence ( if <= 256 b.p.)
          # cols[3] is the through-flow information, a ";" delimited list of
          #   the format 15:1805-3'..207-3'
          # cols[3] will sometimes be null
          if ( $cols[3] )
            {
              my @parts = split ( /;/, $cols[3] );
              foreach my $apart ( @parts )
                {
                  my @subparts = split ( /[:\-\.]/, $apart );
                  # @subparts columns become [0]=15 [1]=1805 [2]=3' [3]=null [4]=207 [5]=3'
                  push ( @{$ends{$cols[1]."0'"."i"}}, [ $subparts[1], $subparts[2], $subparts[0] ]
);
                  push ( @{$ends{$cols[1]."0'"."i"}}, [ $subparts[3], $subparts[4], $subparts[0] ]
);
                } # foreach @leftparts
            } # if ( $cols[3] )
        } # if ( $flowthrough )
    } # "I"


  elsif ( $cols[0] eq "F" )  # flowbetween information
    {
      $flines++;
      if ( defined $flowbetween )
        {
          # cols[1] is contig number
          # cols[2] is flow information for reads flowing from the 5' end of the contig
          # cols[3] is flow information for reads flowing from the 3' end of the contig
          my @leftparts = split ( /;/, $cols[2] );
          my @rightparts = split ( /;/, $cols[3] );
          # each part is of the format xx/yy/z.z where xx=contig yy=number of reads
z.z=distance in b.p.
          if ( $cols[2] ne "-" )  # "-" is the indicator for null entry
            {
              foreach my $apart ( @leftparts )
                {
                  my @subparts = split ( /\//, $apart );
                  # this, and paired links is the only case where we save a fourth column in
                  # the %ends hash, a distance in b.p. value
                  # $flowbetween is also used as a minimum distance cutoff for filtering, so
                  # filter by this distance value, and ignore if too short
                  if ( $subparts[2] >= $flowbetween )
                    { push ( @{$ends{$cols[1]."5'f"}}, [ $subparts[0], "0'", $subparts[1],
$subparts[2] ] ); }
                } # foreach @leftparts
            } # if ne "-"
          if ( $cols[3] ne "-" )  # "-" is the indicator for null entry
            {
              foreach my $apart ( @rightparts )
                {
                  my @subparts = split ( /\//, $apart );
                  if ( $subparts[2] >= $flowbetween )
                    { push ( @{$ends{$cols[1]."3'f"}}, [ $subparts[0], "0'", $subparts[1],
$subparts[2] ] ); }
                } # foreach @leftparts
            } # if ne "-"
        } # if ( defined $flowbetween )
      } # "F"


    elsif ( $cols[0] eq "S" )
      {
        $slines++;
      } # "S"


    elsif ( $cols[0] eq "P" )
      {
                    $plines++;
                    if ( $pairlinks ) {
                              my $con_number = $cols[1];
                              my $fiveprime_connects = $cols[2];
                              my $threeprime_connects = $cols[3];
                              my @tmp = split ";", $fiveprime_connects;
                              foreach my $connection (@tmp){
                                        #now split up the connection.
                                        next if $connection eq "-";
```

```
                                my @x = split "/", $connection;
                                my $termcontig = $x[0];
                                my $num_connects = $x[1];
                                my $distance = $x[2];
                                if($num_connects >= $lowlimit){
                                        #store in temp pairs var and search through later for
termcontig details.
                                        push( @{$pairs{$con_number}}, [ $termcontig, "5'",
$num_connects, $distance ] );
                                        $endsstored += 2;
                                        if ( $endsstored < $debuglimit )
                                                {
                                                        debugmsg ( "Storing ends:
\$ends{".$con_number."3'p} [ $termcontig, 5', $num_connects];" );
                                                        debugmsg ( "              :
\$ends{".$termcontig."5'p} [ $con_number, 3', $num_connects];" );
                                }
                                }
                        }
                        @tmp = split ";", $threeprime_connects;
                        foreach my $connection (@tmp){
                                #now split up the connection
                                next if $connection eq "-";
                                my @x = split "/", $connection;
                                my $termcontig = $x[0];
                                my $num_connects = $x[1];
                                my $distance = $x[2];
                                if($num_connects >= $lowlimit){
                                        #store in temp pairs var and search through later for
termcontig details.
                                        push( @{$pairs{$con_number}}, [ $termcontig, "3'",
$num_connects, $distance ] );
                                        $endsstored += 2;
                                        if ( $endsstored < $debuglimit )
                                                {
                                                        debugmsg ( "Storing ends:
\$ends{".$con_number."3'p} [ $termcontig, 5', $num_connects];" );
                                                        debugmsg ( "              :
\$ends{".$termcontig."5'p} [ $con_number, 3', $num_connects];" );
                              }
                                }
                        }
                } # if ( $pairlinks )
   } # "P"


  elsif ( $cols[0] =~ m/^\d+$/ )  # first section of file
   {
     $contiglen[$cols[0]] = $cols[2];
     $contigcov[$cols[0]] = $cols[3];
```

```
      } # section 1


    else
      {
        die ( "Error on line $lines of file \"$infilename\", unknown type of content:\n$aline\n" );
      }

  } # while <$INF>
close $INF;


debugmsg ( commify($lines) . " lines read from input file \"$infilename\"" );
debugmsg ( commify($#contiglen) . " contig lengths were stored" );
debugmsg ( commify($clines) . " \"C\" lines were found" );
debugmsg ( commify($endsstored) . " ends were stored" );
debugmsg ( commify($ilines) . " \"I\" lines were found" );
debugmsg ( commify($flines) . " \"F\" lines were found" );
debugmsg ( commify($plines) . " \"P\" lines were found" );
debugmsg ( commify($slines) . " \"S\" lines were found (and ignored)" );
}



###############################################################
# make paired end information links
###############################################################
# Added by Simon Gladman - CSIRO - 2011
# Adds paired end links to the link data variable "%ends"
foreach my $key (keys %pairs){
	my @contig = @{$pairs{$key}};
	foreach my $tmp (@contig){
		my @x = @{$tmp};
		my $term_contig = $x[0];

		my @tcontig;  # 10/11/2011 is this a bug? occassional not defined state here
		if ( $pairs{$term_contig} ) { @tcontig = @{$pairs{$term_contig}}; } # end
of fix
		#my @tcontig = @{$pairs{$term_contig}};

		foreach my $ttmp (@tcontig){
			my @y = @{$ttmp};
			if($y[0] == $key){

				push( @{$ends{$key."$x[1]"."p"}}, [ $term_contig, "$y[1]",
$x[2], $x[3] ] );
				push( @{$ends{$term_contig."$y[1]"."p"}}, [ $key, "$x[1]",
$y[2], $y[3] ] );
				last;
			}
```

```
                    }
            }
    }



############################################################
# sort data
############################################################
# sort data so that higher read coverage links come first
# sorting is only needed if we will automatically extend network
if ( $extend )
  {
    # extend applies only to command line contigs and not auto-generated ones
    foreach ( @contig )
      { push ( @extarr, $extend ) }  # but the $extend value just evaluates to "true" later
    debugmsg ( "--extend=$extend, Sorting data" );
    foreach my $key ( keys %ends )
      {
        @{$ends{$key}} = sort {$b->[2] <=> $a->[2]} @{$ends{$key}};
      } # foreach my $key ( keys %ends )
    debugmsg ( "Extend contig list of ".scalar(@extarr)." contigs" );
  } # if ( $extend )



############################################################
# dead end detection
############################################################
# dead ends are contigs which might have reads extending to another contig
# from either the 5' or 3' end, but not both. Both ends could be dead ends, too.
# %deadends{contigid} is # of ends extending, so 1 = dead end,
# 2 = continues both ends, 0=isolated contig without any connections
unless ( $nolabel )
  {
    debugmsg ( "Dead end detection" );
    foreach my $key ( keys %ends )
      {
        ( my $contig = $key ) =~ s/[530]'[pfis]?$//;  # remove 5' or 3' or 5'p etc at end of string
        # note that this version of $contig does not have leading zeroes
        $deadends{$contig}++;
        @{$ends{$key}} = sort {$b->[2] <=> $a->[2]} @{$ends{$key}};
      } # foreach my $key ( keys %ends )
  } # unless ( $nolabel )



############################################################
# construct network from starting point(s)
############################################################
```

```
my $totalnodes = 0;
my $index = 0;
foreach my $acontig (@contig)
  {
    my $dbgtxt = (defined $extarr[$index])?"user-defined":"auto-extend";

    debugmsg ( "Contig #".($index+1)."=$dbgtxt, Recursion starting at \"$acontig\"" );
    # when hit end of our specified contigs
    unless ( defined $extarr[$index] )
      {
        if ( $extend ) { debugmsg ( "At end of specified contigs, turning off extend" ); }
        $extend = 0;
      }
    recurse ( $acontig, "", "", 0, $extend );
    $index++;
  } # foreach my $acontig (@contig)
unless ( $quiet ) { print commify($totalnodes), " nodes present in output\n"; }



############################################################
sub recurse { my ( $startcontig, $camefrom, $fromend, $recurselevel, $followlevel ) = @_;
############################################################
  unless ( $totalnodes > $debuglimit ) { debugmsg ( "recurse \"$startcontig\",
camefrom=\"$camefrom\" end=\"$fromend\" recurselevel=$recurselevel" ); }

  # data for this contig is in @ { %ends{contig . 5'|3'|5'p|3'p|5'f|3'f|5'i... } } [0]=nextcontig
[1]=5'|3'|5'p|3'p"5'f|3'f|5'i... [2]=readnum

  # store lowest recursion level seen for this contig
  unless ( $nolabel )
    {
      if ( ( ! defined $minrl{$startcontig} ) or ( $recurselevel < $minrl{$startcontig} ) )
        { $minrl{$startcontig} = $recurselevel; }
    } # unless ( $nolabel )

  # here is the limit to recursion
  my $stophere = 0;
  if ( $recurselevel > $level )
    {
      unless ( $totalnodes > $debuglimit ) { debugmsg ( "recursion for \"$startcontig\" at limit
level=$level, returning" ); }
      $stophere = 1;
    }

  # return if this contig is on the exclude list
  elsif ( $excludehash{$startcontig} )
    {
      unless ( $totalnodes > $debuglimit ) { debugmsg ( "contig \"$startcontig\" on exclude list,
returning" ); }
```

```
      # save information for list of excluded option
      my @row = ( $startcontig, $camefrom, $fromend );
      push ( @listofexcl, \@row );
      return 0;
## was this wrong? always return if on exclude list     $stophere = 1;
    }

  # skip return if follow allows it
  if ( ( $stophere ) and ( $followlevel < 1 ) ) { return 0; }

  # count nodes
  unless ( defined $seen{$startcontig} )
    {
      if ( $startcontig =~ m/^\s*$/ ) { die "Error, null contig in sub recurse($startcontig,
$camefrom, $fromend, $recurselevel, $followlevel)\n"; }
      $seen{$startcontig} = 1;
      $totalnodes++;
    }

  foreach my $end ( "5'", "3'", "5'p", "3'p", "5'f", "3'f", "5'i", "3'i", "0'" )
    {
      my $first = 1;
      foreach my $contigref ( @{$ends{$startcontig.$end}} )
        {
          unless ( $totalnodes > $debuglimit ) { debugmsg ( "from \"$startcontig\" end \"$end\"
find linked contig ".join(";",@$contigref) ); }

          # always skip links back to where we just came from ( and don't clear $first flag )
          if ( $contigref->[0] eq $camefrom )
            {
              unless ( $totalnodes > $debuglimit ) { debugmsg ( "\"$startcontig\": skipping link
back to source contig \"$camefrom\"" ); }
              next;
            }

          # skip data storing if this edge was already stored
          unless ( defined $edgeseen{$startcontig.$end.$contigref->[0].$contigref->[1]} )
            {
              debugmsg ( "Storing in \@data at key \"$startcontig$end\": [ \"".join ( "\", \"",
@$contigref )."\" ]" );
              push ( @{$data{$startcontig.$end}}, $contigref );  # $contigref is array reference

              if ( ( $first ) and ( $followlevel ) )
                {
                  unless ( $totalnodes > $debuglimit ) { debugmsg ( "auto extension following
contig \"$contigref->[0]\", \$extensions=$extensions" ); }
                  $extensions++;
                  $edgeseen{$startcontig.$end.$contigref->[0].$contigref->[1]} = $recurselevel;
                  recurse ( $contigref->[0], $startcontig, $end, $recurselevel, $followlevel-1 );
                } # if
            } # unless $edgeseen
          else
            { debugmsg ( "Seen, not storing in \@data at key \"$startcontig$end\": [ \"".join ( "\",
\"", @$contigref )."\" ]" ); }


          # recurse if edge seen level is higher than current level or not seen before
          if ( ( ! defined $edgeseen{$startcontig.$end.$contigref->[0].$contigref->[1]} ) or
               ( $edgeseen{$startcontig.$end.$contigref->[0].$contigref->[1]} > $recurselevel ) )
            {
              $edgeseen{$startcontig.$end.$contigref->[0].$contigref->[1]} = $recurselevel;
              recurse ( $contigref->[0], $startcontig, $end, $recurselevel+1, 0 );
            }
          $first = 0;
        } # foreach $contigref
    } # foreach $end

} # sub recurse



###############################################################
# consolidate pairs of flow between links
###############################################################
collapseflowbetween();



###############################################################
# create output table text file
###############################################################
createoutputtable();



###############################################################
# generate optional ABySS-Explorer .dot file
###############################################################
if ( $abyssdotfile ) { createabyssfile(); }



###############################################################
# generate optional FASTA file
###############################################################
if ( $outfastaname ) { createfastafile(); }



###############################################################
```

```
# create graphviz command file (.dot file)
################################################################
my $savednodes = 0;
my $savededges = 0;
debugmsg ( "Creating graphviz command file \"$cmdfilename\"" );
open ( my $OUTF, ">", $cmdfilename ) or die ( "Error creating graphviz command file
\"$cmdfilename\": $!\n" );
print $OUTF "graph G\n";
print $OUTF "  {\n";
print $OUTF "    edge [len=$len];\n";
my $splinemode = $nospline?"false":"true";
print $OUTF "    graph [overlap=$overlapmode,splines=$splinemode];\n";
print $OUTF "    node [shape=plaintext];\n";

# table of nodes
print $OUTF "\n    // Nodes\n";
foreach my $seencontig ( keys %seen )
  {
    my $contigid = sprintf ("%0${numdigits}d", $seencontig);

    # box is proportional to size, except if contig is small,
    # the box is still as large as the labels it contains
    my $scaledcontiglen = sprintf ( "%1d", $contiglen[$seencontig] * $scalefactor );

    # define the box representing the contig, implemented as a table in HTML
    print $OUTF "    c$contigid [label=< <TABLE BORDER=\"1\" CELLBORDER=\"0\"
CELLSPACING=\"0\" CELLPADDING=\"0\"";
    if ( $colorhash{$seencontig}->[0] ) { print $OUTF "
BGCOLOR=\"$colorhash{$seencontig}->[0]\""; }

    # top row is always present
    print $OUTF "><TR>";
    if ( $inverthash{$seencontig} )
      { print $OUTF "<TD PORT=\"R\">3'</TD>"; }
    else
      { print $OUTF "<TD PORT=\"L\">5'</TD>"; }
    print $OUTF "<TD PORT=\"C\" WIDTH=\"$scaledcontiglen\"";
    print $OUTF ">c$contigid</TD>";
    if ( $inverthash{$seencontig} )
      { print $OUTF "<TD PORT=\"L\">5'</TD>"; }
    else
      { print $OUTF "<TD PORT=\"R\">3'</TD>"; }
    print $OUTF "</TR>";

    # optional table row for contig size
    if ( $showbp )
      { print $OUTF "<TR><TD COLSPAN=\"3\">",commify($contiglen[$contigid]),"
$bpabbreviation</TD></TR>"; }

    # optional table row for coverage
    if ( $showcov )
      { print $OUTF "<TR><TD
COLSPAN=\"3\">cov=",commify($contigcov[$contigid]),"</TD></TR>"; }

    # one or more optional rows for labels specified on the command line
    if ( $taghash{$seencontig}->[0] )
      {
        foreach my $tag ( @{$taghash{$seencontig}} )
          { print $OUTF "<TR><TD COLSPAN=\"3\">$tag</TD></TR>"; }
      }

    # optional final rows for dead end contigs, or recursion limit contigs
    unless ( $nolabel )
      {
        if ( ( ! defined $deadends{$seencontig} ) or ( $deadends{$seencontig} <= 1 ) )
          { print $OUTF "<TR><TD COLSPAN=\"3\" BGCOLOR=\"$deadendcolor\">Dead
End</TD></TR>"; }
        if ( ( defined $minrl{$seencontig} ) and ( $minrl{$seencontig} >= $level ) )
          { print $OUTF "<TR><TD COLSPAN=\"3\"
BGCOLOR=\"$recursionlimitcolor\">Recursion limit</TD></TR>"; }
      } # unless ( $nolabel )

    # and the end of this huge mess of HTML
    print $OUTF "</TABLE> >];\n";

  } # foreach my $seencontig ( keys %seen )
debugmsg ( "Saved ".commify($savednodes)." nodes" );

# table of edges ( connections )
print $OUTF "\n    // Adjacency Edges\n";
my %alreadydrawn = ();
foreach my $key ( keys %data )
  {
    $key =~ m/^(.*)([530]'[pfis]?)$/;
    my $srcend = $2;
    my $srccontig = sprintf ("%0${numdigits}d", $1);
    my $srclr = "C";
    if ( $srcend =~ m/5'/ ) { $srclr = "L"; }
    if ( $srcend =~ m/3'/ ) { $srclr = "R"; }
    unless ( $srclr ) { die "Error, no valid end at \"$key\"\n"; }
    if ( $savededges < $debuglimit ) { debugmsg ( "Key=\"$key\" Contig=\"$srccontig\"
End=\"$srcend\" Port=\"$srclr\"" ); }
    foreach my $edgeref (@{$data{$key}})  # edgeref elements: [0]=contig(no leading 0)
[1]=5'|3'|5'p|3'p|5'f|3'f|5'i... [2]=readnumber
      {
        # skip those final edges not leading to a node in the %seen list,
        # or those we have deleted by marking with "X"
        if ( ( $seen{$edgeref->[0]} ) and ( $edgeref->[1] ne "X" ) )
          {
```

```
        if ( $savededges < $debuglimit ) { debugmsg ( "Edge=[ \"$edgeref->[0]\" \"$edgeref-
>[1]\" \"$edgeref->[2]\" ]" ); }

        my $contigid = sprintf ("%0${numdigits}d", $edgeref->[0]);  # format for graph has
leading zeroes
        my $lr = "C";
        if ( $edgeref->[1] =~ m/5'/ ) { $lr = "L"; }
        if ( $edgeref->[1] =~ m/3'/ ) { $lr = "R"; }
        unless ( $lr ) { die "Error no valid end at \"$edgeref->[0]\"\n"; }

        unless ( $alreadydrawn{$srccontig.$srclr."-".$contigid.$lr} )
          {
            my $linklabel = $edgeref->[2];
            if ( $edgeref->[3] )
              {
                # if the distance is 10 b.p. or greater, remove the decimal places to eliminate
clutter
                if ( $edgeref->[3] >= 10 ) { $edgeref->[3] =~ s/\..*$//; }
                $linklabel .= "/" . $edgeref->[3] . $bpabbreviation;
              }
            print $OUTF "    \"c$srccontig\":$srclr -- \"c$contigid\":$lr [label=\"$linklabel\"";

            # bold for high coverage links
            if ( ( $boldabove ) and ( $edgeref->[2] >= $boldabove ) ) { print $OUTF "
style=bold"; }

            # specify colors of lines and labels connecting contigs
            if ( $edgeref->[2] == 0 )  # forced link
              { print $OUTF " color=$forcedlinkcolor fontcolor=$forcedfontcolor"; }
            elsif ( ( $srcend =~ m/p/ ) or ( $edgeref->[1] =~ m/p/ ) )  # paired end link
              { print $OUTF " color=$pairedendlinkcolor fontcolor=$pairedendfontcolor"; }
            elsif ( ( $srcend =~ m/f/ ) or ( $edgeref->[1] =~ m/f/ ) )  # flowbetween link
              { print $OUTF " color=$flowbetweenlinkcolor
fontcolor=$flowbetweenfontcolor"; }
            elsif ( ( $srcend =~ m/i/ ) or ( $edgeref->[1] =~ m/i/ ) )  # flowthrough link
              { print $OUTF " color=$flowthroughlinkcolor fontcolor=$flowthroughfontcolor";
}
            else  # if $normallinkcolor is a null string, don't specify any color, use default of
black
              {
                if ( $normallinkcolor ) { print $OUTF " color=$normallinkcolor"; }
                if ( $normalfontcolor ) { print $OUTF " fontcolor=$normalfontcolor"; }
              }

            print $OUTF "];\n";
            $alreadydrawn{$contigid.$lr."-".$srccontig.$srclr} = 1;  # note we store in reverse
orientation here
            $savededges++;
          }
      }
      } # foreach my $edgeref (@$arrayref)
  } # foreach my $arrayref ( keys %data )
debugmsg ( "Saved ".commify($savededges)." edges" );
print $OUTF "  }\n";
close $OUTF;



#################################################################
# run graphviz program neato
#################################################################
my $cmd = "neato -T${outtype} -o\"$imgfilename\" \"$cmdfilename\"";
debugmsg ( "running command \"$cmd\"" );
my $result = system ( $cmd );
if ( $result )
  {
    print "Problem creating image. Error code $result returned from command \"$cmd\"\n";
    $returncode = $result;
  }
else
  {
    unless ( $quiet )
      { print "Success\n"; }
  }

# remove graphviz command file
if ( $deletecmdfile ) { unlink $cmdfilename; }



#################################################################
# print out list of excluded contigs
#################################################################
if ( $listexcluded )
  {
    print "List of excluded contigs\n";
    print "Contig\tLinked from\tFrom End\n";
    foreach my $rowref ( @listofexcl )
      { print join ( "\t", @{$rowref} ), "\n"; }
  } # if ( $listexcluded )



#################################################################
# end of program
#################################################################
exit $returncode;
```

```
#############################################################
sub collapseflowbetween {
#############################################################
# flowbetween links do not have a ending end, so are designated 0' there,
# but since they come in pairs, we can figure out the ends that way
# and consolidate the two links into one to eliminate clutter
# global variables (command line parameters) used:
#
# other global variables uses
#   %data

my %finder;

# first step is to create an index
foreach my $key ( keys %data )
  {
    next unless ( $key =~ m/f/ );
    unless ( $key =~ m/^(.*)([530]'[pfis]?)$/ ) { die "Program bug parsing key \"$key\"\n"; }
    my $srcend = $2;
    my $srccontig = $1; #sprintf ("%0${numdigits}d", $1);
    foreach my $edgeref (@{$data{$key}})  # [0]contig(no leading 0)
[1]5'|3'|5'p|3'p|5'f|3'f|5'i... [2]readnumber [3]distance
      {
        if ( $edgeref->[1] =~ m/0'/ )
          {
            my $bothkey = $srccontig . ":" . $edgeref->[0] . ":" . $edgeref->[3]; # omit source end
in this key
            debugmsg ( "Save collapsible reference \"$bothkey\" from key \"$key\"" );
            # save the source end as array element [4]
            $edgeref->[4] = $srcend;
            push ( @{$finder{$bothkey}}, $edgeref );
          }
      } # foreach $edgeref
  } # foreach my $key ( keys %data )

# second step is to check the index for unique reciprocal links
foreach my $key ( keys %data )
  {
    next unless ( $key =~ m/f/ );
    unless ( $key =~ m/^(.*)([530]'[pfis]?)$/ ) { die "Program bug parsing key \"$key\"\n"; }
    my $srcend = $2;
    my $srccontig = $1;
    foreach my $edgeref (@{$data{$key}})  # [0]contig(no leading 0)
[1]5'|3'|5'p|3'p|5'f|3'f|5'i... [2]readnumber [3]distance
      {
        my $bothkey = $srccontig . ":" . $edgeref->[0] . ":" . $edgeref->[3];
        my $reversekey = $edgeref->[0] . ":" . $srccontig . ":" . $edgeref->[3];

        if ( ( $finder{$bothkey} ) and ( $finder{$reversekey} ) )
          {
```

```
          # if by chance multiple matches, do not merge links
          if ( ( scalar @{$finder{$bothkey}} == 1 ) and ( scalar @{$finder{$reversekey}} == 1
) )
            {
              debugmsg ( "Collapsing link \"$bothkey\" <=> \"$reversekey\"" );
              my @parts = split ( /:/, $bothkey );
              # look up the correct other end of this link from the other member of the reciprocal
pair
              my $correctotherend = $finder{$reversekey}->[0]->[4];
              # save new link
              push @{$data{$key}}, [ $edgeref->[0], $correctotherend, $edgeref->[2], $edgeref-
>[3] ];
              # set flag on old links to indicate that they should be ignored later
              # the value from %finder is a reference back to the $edgeref from %data,
              # so here we are modifying the %data hash indirectly
              $finder{$bothkey}->[0]->[1] = "X";
              $finder{$reversekey}->[0]->[1] = "X";

              # finished processing, remove both original links from %finder hash
              # to avoid hitting the same link again in the reverse orientation
              undef ( $finder{$bothkey} );
              undef ( $finder{$reversekey} );
            }
          else
              { debugmsg ( "Ignoring multiple collapsible links for \"$bothkey\" or
\"$reversekey\"" ); }
        }
      } # foreach $edgeref
  } # foreach my $key ( keys %data )
} # sub collapseflowbetween



################################################################
sub createoutputtable {
################################################################
# global variables (command line parameters) used:
#   $outfilename, $pairlinks, $flowthrough, $flowbetween, $scaffold
# other global variables uses
#   %data
my %alreadyprinted = ();
my $printededges = 0;
open ( my $OUTF, ">", $outfilename ) or die ( "Error opening output file \"$outfilename\":
$!\n" );
print $OUTF join ( "\t", "#contig", "contiglen", "avg.cov.", "5'or3'", "is linked to", "5'or3'",
"by read num", "contiglen", "avg.cov." );
if ( ( $pairlinks ) or ( $flowthrough ) or ( defined $flowbetween ) or ( $scaffold ) )
  { print $OUTF "\tlink type"; }
print $OUTF "\n";
```

```
foreach my $key ( sort { ($a=~m/^(\d+)[530]'/)[0] <=> ($b=~m/^(\d+)[530]'/)[0] } keys
%data )
  {
    unless ( $key =~ m/^(.*)([530]'[pfis]?)$/ ) { die "Program bug, unparsable key \"$key\"\n";
}
    my $srccontig = $1;
    my $srcend = $2;
    foreach my $edgeref (@{$data{$key}})  # [0]contig(no leading 0)
[1]5'|3'|5'p|3'p|5'f|3'f|5'i... [2]readnumber
      {
        unless ( ( $alreadyprinted{$srccontig.$srcend."-".$edgeref->[0].$edgeref->[1]} ) or (
$edgeref->[1] eq "X" ) )
          {
            print $OUTF join ( "\t", $srccontig, $contiglen[$srccontig], $contigcov[$srccontig],
                               $srcend, $edgeref->[0], $edgeref->[1], $edgeref->[2],
                               $contiglen[$edgeref->[0]], $contigcov[$edgeref->[0]] );
            if ( ( $pairlinks ) or ( $flowthrough ) or ( defined $flowbetween ) or ( $scaffold ) )
              {
                my $t = "direct";
                if ( $srcend =~ m/p/ ) { $t = "paired-end"; }
                if ( $srcend =~ m/f/ ) { $t = "flowbetween"; }
                if ( $srcend =~ m/i/ ) { $t = "flowthrough"; }
                if ( $srcend =~ m/s/ ) { $t = "scaffold"; }
                print $OUTF "\t", $t;
              }
            print $OUTF "\n";
            $alreadyprinted{$edgeref->[0].$edgeref->[1]."-".$srccontig.$srcend} = 1;  # note we
store in reverse orientation here
            $printededges++;
          }
      } # foreach my $edgeref (@{$data{$key}})
  } # foreach my $arrayref ( keys %data )
debugmsg ( "Printed ".commify($printededges)." edges to output file \"$outfilename\"" );
close $OUTF;
} # sub createoutputtable



############################################################
sub createabyssfile {
############################################################
    # global variables (command line parameters) used:
    #   $abyssdotfile, $pairlinks, $flowthrough, $flowbetween, $scaffold
    # other global variables uses
    #   %data

    # collect connection data
    my %aedata;   # direct links
    my %aepdata;  # paired end links
    foreach my $key ( keys %data )
```

```
      {
       unless ( $key =~ m/^(.*)([530]'[pfis]?)$/ ) { die "Program bug parsing key \"$key\"\n"; }
       my $srccontig = $1;
       my $srcend = $2;
       debugmsg ( "AE: key=\"$key\" becomes srccontig=\"$srccontig\" srcend=\"$srcend\"" );
       foreach my $edgeref (@{$data{$key}})  # edgeref elements: [0]=contig(no leading 0)
[1]=5'|3'|5'p|3'p|5'i... [2]=readnumber [3]=distance
         {
          if ( $edgeref->[1] eq "X" )  # masked entry, ignore it
            {
             debugmsg ( "AE: masked edgeref \"$edgeref->[0]\" \"$edgeref->[1]\" \"$edgeref-
>[2]\"" );
             next;
            }
          if ( $key !~ m/[pfis]/ )  # direct connection
            {
             my $nreads = $edgeref->[2];
             my $distance = 0;
             debugmsg ( "AE: direct edgeref \"$edgeref->[0]\" \"$edgeref->[1]\" \"$edgeref-
>[2]\"" );
             # values assigned here are not currently used, just the state of being defined
             $aedata{$srccontig}->{$srcend}->{$edgeref->[0]}->{$edgeref->[1]} = [
$distance, $nreads ];  # $nreads would be zero for forced links
             # if no links otherwise show up, still need a dummy line in the .adj file
             $aedata{$edgeref->[0]}->{used} = 1;
            }
          elsif ( $key =~ m/p/ )  # paired end connection
            {
             my $nreads = $edgeref->[2];
             my $distance = $edgeref->[3];
             debugmsg ( "AE: paired-end edgeref \"$edgeref->[0]\" \"$edgeref->[1]\"
\"$edgeref->[2]\" \"$edgeref->[3]\"" );
             unless ( $distance ) { die "Program bug distance = \"$distance\" key $key\n"; }
             $aepdata{$srccontig}->{$srcend}->{$edgeref->[0]}->{$edgeref->[1]} = [
$distance, $nreads ];
             $aepdata{$edgeref->[0]}->{used} = 1;
            }
         } # foreach my $edgeref (@{$data{$key}})
      } # foreach my $arrayref ( keys %data )

   ##### ABySS-Explorer 1.3.0 .dot output file example lines
   # parts of the example file SRP000220-6.dot
   #digraph adj {
   #graph [k=32]
   #edge [d=-31]
   #"18+" [l=334 C=32002]
   #"18-" [l=334 C=32002]
   #...
   #"1807+" -> "1811-" [d=-706]
   #"1807+" -> "1825+" [d=-706]
   #
   # from SRP000220-6.path1.dot
   # "1861+" -> "1863+" [d=35 e=1.6 n=172]
   # "1861+" -> "1886-" [d=-13 e=1.8 n=135]

   # not sure about use of this file, not loadable SRX000430-6.dist.dot
   #digraph dist {
   #graph [k=32 s=100 n=10]
   #"18+" -> "71-" [d=320 e=2.6 n=62]

   debugmsg ( "Creating ABySS-Explorer .dot file \"$abyssdotfile\"" );
   open ( my $OUTF, ">", $abyssdotfile ) or die ( "Error creating ABySS-Explorer .dot file
\"$abyssdotfile\": $!\n" );
   # start of .dot file, the header line, I don't know if other names than "adj" are valid, I didn't
check
   print $OUTF "digraph $dotheaderid \{\n";
   print $OUTF "graph [k=", $abyssk, "]\n";   # this will be 1 because we don't use kmers
   print $OUTF "edge [d=", $abyssedge, "]\n";

   ### Vertices
   {

   my %list;  # make a list of just the contig numbers, but from both shotgun and paired end
   foreach my $acontig ( keys %aedata )
     {
       if ( $aedata{$acontig}->{used} )
       #$acontig =~ s/[530]'[pfis]?//;
         { $list{$acontig} = 1; }
     }
   foreach my $acontig ( keys %aepdata )
     {
       if ( $aepdata{$acontig}->{used} )
         { $list{$acontig} = 1; }
     }

   foreach my $acontig ( sort { $a <=> $b } keys %list )
     {
       # coverage must be an integer for ABySS-Explorer, so round to nearest integer
       # multiply coverage by length because ABySS-Explorer uses kmer coverage, we need to
simulate that
       my $avgcov = sprintf( "%0d", ( $contigcov[$acontig] * $contiglen[$acontig] ) );
       # print contig here
       print $OUTF "\"" . $acontig . "+\" [l=" . $contiglen[$acontig] . " C=" . $avgcov . "]\n";
       print $OUTF "\"" . $acontig . "-\" [l=" . $contiglen[$acontig] . " C=" . $avgcov . "]\n";
     } # foreach my $acontig %data
   }

   ### adj pattern
   foreach my $acontig ( sort { $a <=> $b } keys %aedata )  # $acontig is just a contig
number
```

```
        {
          debugmsg ( "AE: write adj data for contig \"$acontig\"" );
          foreach my $fend ( sort keys %{$aedata{$acontig}} )
            {
              next if ( $fend !~ m/\d/ ); # i.e., ne 'used'
              foreach my $rcontig ( sort { $a <=> $b } keys %{$aedata{$acontig}->{$fend}} )
                {
                  foreach my $rend ( sort keys %{$aedata{$acontig}->{$fend}->{$rcontig}} )
                    {
                      # currently only use defined state, values ignored for direct connections
                      # my ( $distance, $nreads ) = @{$aepdata{$acontig}->{$fend}->{$rcontig}-
>{$rend}};

                      # + and - are backwards from what looks natural, so that ABySS-Explorer
                      # defaults to showing arrows 5' to 3'
                      # 3'->5' = - -  3'->3' = - +
                      # 5'->5' = + -  5'->3' = + +
                      my $fdir = ($fend=~m/5/)?"+":"-";
                      my $rdir = ($rend=~m/3/)?"+":"-";

                      # print link here
                      print $OUTF "\"${acontig}${fdir}\" -> \"${rcontig}${rdir}\"\n";
                      debugmsg ( "AE: rcontig=$rcontig pm=$rdir link is \"${acontig}${fdir}\" ->
\"${rcontig}${rdir}\"" );
                    } # foreach $rend
                } # foreach $rcontig
            } # foreach $fend
        } # foreach my $acontig %aedata

      ### dist pattern
      foreach my $acontig ( sort { $a <=> $b } keys %aepdata )
        {
          debugmsg ( "AE: write dist data for contig \"$acontig\"" );
          foreach my $fend ( sort keys %{$aepdata{$acontig}} )
            {
              next if ( $fend !~ m/\d/ ); # i.e., ne 'used'
              foreach my $rcontig ( sort { $a <=> $b } keys %{$aepdata{$acontig}->{$fend}} )
                {
                  if ( $fend !~ m/p/ ) { die "Program bug: \%aepdata key for $acontig has no \"p\":
\"$fend\"\n"; }
                  foreach my $rend ( sort keys %{$aepdata{$acontig}->{$fend}->{$rcontig}} )
                    {
                      my ( $distance, $nreads ) = @{$aepdata{$acontig}->{$fend}->{$rcontig}-
>{$rend}};
                      unless ( $distance ) { die "Program bug, paired end distance not defined contig
$acontig end $fend to end $rcontig end $rend\n"; }

                      # 3'->5' = - -  3'->3' = - +
                      # 5'->5' = + -  5'->3' = + +
                      my $fdir = ($fend=~m/5/)?"+":"-";
                my $rdir = ($rend=~m/3/)?"+":"-";

                # special precise formatting needed by ABySS-Explorer
                $distance = int ( $distance + 0.5 );  # must be integer
                my $e = sprintf ( "%0.1f", $abyssevalue ); # units are b.p., required 1 decimal
place
                my $n = int($nreads); # n is number of mates, must be integer, here we use the
number of reads, same thing

                # print link here
                print $OUTF "\"${acontig}${fdir}\" -> \"${rcontig}${rdir}\" [d=$distance e=$e
n=$n]\n";
                debugmsg ( "AE: write pe link \"${acontig}${fdir}\" -> \"${rcontig}${rdir}\"
[d=$distance e=$e n=$n]" );
               } # foreach $rend
            } # foreach $rcontig
         } # foreach $fend
      } # foreach my $acontig %aepdata

   # end of file
   print $OUTF "\}\n";
   close $OUTF;

} # sub createabyssfile



################################################################
sub createfastafile {
################################################################
   # global variables (command line parameters) used:
   #   $indirname, $outfastaname
   # other global variables uses
   #   %seen, $bpabbreviation
   my %fasta = ();  # store all contigs in memory
   my $fastainfilename = $indirname;
   unless ( $fastainfilename =~ m/\/$/ ) { $fastainfilename .= "/"; }
   $fastainfilename .= $allcontigsfna;
   my $lines = 0;
   my $sequences = 0;
   my $seqsaved = 0;
   my $bpsaved = 0;
   my $saveflag = 0;
   my $id = "";
   open ( my $INF, "<", $fastainfilename ) or die ( "Error opening input file
\"$fastainfilename\": $!\n" );
   while ( my $aline = <$INF> )
     {
      $lines++;
      $aline =~ s/[\r\n]//g;
```

```
        if ( $aline =~ m/^>([^\s]*)/ )
          {
            $id = $1;  # up to first white space
            $id =~ s/contig0*//;  # remove "contig" and any leading zeroes
            if ( $id =~ m/^\s*$/ ) { die "Error, null contig name from line $lines of file
\"$fastainfilename\"\n"; }
            $sequences++;
            $saveflag = $seen{$id};
          } # if
        if ( $saveflag )
          {
            $fasta{$id} .= $aline . "\n";
            unless ( $aline =~ m/^>/ )
              {
                $aline =~ s/[^AaCcTtGgMmRrYyKkVvHhDdBb]//g;
                $bpsaved += length ( $aline );
              }
          } # if
      } # while
    close $INF;
    unless ( $quiet )
      { print "Input FASTA file contained ", commify($lines), " lines and ",
commify($sequences), " sequences\n"; }

    open ( my $OUTF, ">", $outfastaname ) or die ( "Error opening output file
\"$outfastaname\": $!\n" );
    foreach my $acontig ( sort { $a <=> $b } keys %seen )
      {
        my $seq = exists($fasta{$acontig}) ? $fasta{$acontig} : "";
        if ( $seq )
          { print $OUTF $seq; }
        else
          { print "Warning, empty sequence for contig \"$acontig\"\n"; }
        $seqsaved++;
      }
    close $OUTF;
    unless ( $quiet )
      { print commify($seqsaved), " sequences, ", commify($bpsaved), " $bpabbreviation
saved in \"$outfastaname\"\n"; }
  } # sub createfastafile



############################################################
sub debugmsg { my ( $text, $noreturn, $nolinenum ) = @_;
############################################################
  if ( $debug )
    {
      my ($package, $filename, $line, $sub) = caller(0);
      unless ( $nolinenum ) { $text = "Line $line: " . $text; }
      if ( ! ( $noreturn ) ) { $text .= "\n"; }
      print $text;
    } # if ( $debug )
} # sub debugmsg



###################################################################
sub expandatprefix { my ( $string ) = @_;
###################################################################
# some parameters with contigs can have "@xxx" used instead,
# the text following "@" is a filename.
# This file contains the parameters, which will be
# substituted in. Otherwise return the string unmodified
  if ( $string =~ m/^\@(.*)$/ )
    {
      my $filename = $1;
      open my $EFILE,"<",$filename or die ( "Error opening file \"$filename\": $!\n" );
      my @contents = <$EFILE>;
      close $EFILE;
      $string = join ( ",", @contents );
      $string =~ s/[\r\n\s]//g;
    } # if ( $string =~ m/^\@/ )
  return $string;
} # sub expandatprefix



###################################################################
sub timestr {
###################################################################
  @_ = localtime(shift || time);
  return(sprintf("%04d/%02d/%02d %02d:%02d", $_[5]+1900, $_[4]+1, $_[3], @_[2,1]));
} # sub timestr



###################################################################
sub commify {
###################################################################
# http://perldoc.perl.org/perlfaq5.html#How-can-I-output-my-numbers-with-commas
  local $_ = shift;
  1 while s/^([-+]?\d+)(\d{3})/$1,$2/;
  return $_;
} # commify



# eof
=pod sample
```

```
graph G
  {
    edge [len=1];
    graph [overlap=none,splines=true];
    node [shape=plaintext];
    c12345 [label=< <TABLE BORDER="1" CELLBORDER="0"
CELLSPACING="0"><TR><TD PORT="L">5'</TD><TD WIDTH="200">c12345</TD...
    c23456 [label=< <TABLE BORDER="1" CELLBORDER="0"
CELLSPACING="0"><TR><TD PORT="L">5'</TD><TD WIDTH="10">c23456</TD>...
    c34567 [label=< <TABLE BORDER="1" CELLBORDER="0"
CELLSPACING="0"><TR><TD PORT="L">5'</TD><TD WIDTH="1">c34567</TD><...
    c45678 [label=< <TABLE BORDER="1" CELLBORDER="0"
CELLSPACING="0"><TR><TD PORT="L">5'</TD><TD WIDTH="100">c45678</TD...

    "c23456":L -- "c34567":R [label="31"];
    "c34567":L -- "c45678":R [label="12"];
    "c45678":L -- "c12345":R [label="18"];
    "c34567":L -- "c12345":R [label="1"];
```

Extract_singlets_from_fasta_sff.sh – In-house Shell Script

```
echo -e "This is the script to extract singlets/repeats/outlier from the raw sff files used for
assembly \n Run this in the assembly directory of your gsAssembler run \n Make sure to copy
the sff files used by the gsAssembler run in the directory (you can find that by looking in the
sff folder of your gsAssembler run) \n"
#perl
/root/Downloads/Scripts_for_454_data_processing/copy_sff_files_to_current_assembly_dire
ctory.pl

fgrep Singleton 454ReadStatus.txt > singletons.txt
sfffile -o singletons.sff -i singletons.txt *.sff
sffinfo -s singletons.sff > singletons.fna
fgrep Outlier 454ReadStatus.txt > outliers.txt
sfffile -o outliers.sff -i outliers.txt *.sff
sffinfo -s outliers.sff > outliers.fna
```

ORFFINDER.pl: An in-house Perl script for finding functional ORF in contigs.

```
open(FILE,"gene.fna");
@file=<FILE>;
$header=splice(@file,0,1);
$file=join(' ',@file);
$file=~s/\n//g;
$file=~s/\s//g;
$DNA=~m/(ATG|GTG|TTG)(...)*(TGA|TAG|TAA)/g;
 while ($file =~ /((ATG|GTG|TTG)(...)*(TGA|TAG|TAA))/g) {
      my $orf_length = length($1);
      my $orf_end = pos($file) - 1;
      my $orf_start = pos($file) - $orf_length;
print "orf",$orf_start, "to" ,$orf_end,"\n";
$sub='';
if($sub=index($file,$DNA))
{
$file=~tr/[TTT|TTC][TCT|TCA|TCG|TCC|AGT|AGC][TAT|TAC)(TGT|TGC)(TTA|TTG|CT
T|CTC|CTA|CTG][TGG)][CCA|CCT|CCG|CCC][CAC|CAT][CGT|CGA|CGG|CGC|AGA|A
GG][CAA|CAG][ATT|ATC|ATA][ACA|ACT|ACG|ACC][AAA|AAG][ATG][GCA|GCG|G
CT|GCC][GAT|GAC][GGT|GGA|GGC|GGG][GTT|GTA|GTG|GTC][GAG][AAT|AAC]/[F]
[S][Y][C][L][W][P][H][R][Q][I][T][K][M][A][D][G][V][E][N]/;
print $file;
}
}
```

Separate_contigs_for_mitofy.pl : A perl script for formatting contigs and generating input data for MITOFY.

```
#!/usr/bin/perl
print "USAGE Enter fasta file of contigs to be annotated : ";
$input_file_name=<STDIN>;
open(file,"$input_file_name");
@input_file=<file>;
foreach $input_file(@input_file)
{
if($input_file=~/^>/)
{
@file_name=$input_file;
}
foreach $file_name(@file_name)
{
s/^>//g;
open(new_file,">>$file_name");
do
{
print new_file $input_file;
} while($input_file!=/^>/);}}
```