Chapter 2

A core bioinformatics workflow environment for ultra-high-throughput transcriptome data analysis

Chapter preface

This chapter describes the development of software tools in the form of `Galaxy` workflows to address very specific next-generation sequence analysis problems. The workflows address very specific bioinformatics steps during the analysis of uHTS transcriptome datasets. The developed workflows focus on evaluating the quality of data from an Illumina mRNA-seq run, introduce a *de novo* transcriptome assembly pipeline, describe an automated assemble pipeline, and also provides a framework for gene expression (FPKM) calculation of genes expressed from a genome where the gene models have not been defined yet.

A customised `Galaxy` server has been installed at the Bioinformatics and Computational Biology Unit (BCBU) research group, that contains a copy of tools available in the public `Galaxy` server, as well as new tools that are not available on the default server. These tools consists of either third party, open source applications in the public domain that were included in the BCBU `Galaxy` instance, or novel Python and R scripts that were developed specifically for the BCBU server.

The BCBU `Galaxy` server can be accessed at the following URL: http://zoidberg.bi.up.ac.za:8882

2.1. Introduction

The sheer volumes of data produced by high-throughput technologies are forcing the biological research community to adopt automated data analysis methodologies in order to investigate the underlying biological relevance of the data produced. These technologies have enabled relatively small research groups with moderate budgets to produce large amounts of DNA sequence data, which necessitated the bioinformatics community to develop user-friendly analysis environments geared towards data management and result sharing. The current lack of bioinformatics human capacity, technical support and computational hardware in most research institutions is generally considered the bottleneck in obtaining relevant biological answers to a hypothesis. Deploying flexible and user-friendly analysis systems which empower the laboratory scientist to assist in data analysis and interface with custom software solutions developed by the bioinformatics community will greatly relieve the demand for bioinformatics support in a research project, and will assist both the experimental biologist and bioinformaticist in interpreting experimental findings.

The field of bioinformatics is more often than not spoiled for choice when it comes to selecting the most appropriate software analysis tool to perform a specific analysis. New software tools are made available to the community on a weekly basis, and especially in a newly expanding field such as high-throughput sequencing applications, various analysis tools that perform essentially the same function, but following different methodological approaches are rapidly being developed. A good example is the wide range of short read alignment tools currently available to align results from mRNA-Seq data to a target genome (Table 1.1). Each of these software packages have been designed with specific criteria in mind, and selecting the most appropriate tool that fits an experimental design or computational environment is often a daunting task. Many research groups and consortia have developed software pipelines and automated systems which use specific tools to address the need for analysis automation (Mungall *et al.*, 2002; Durham *et al.*, 2005; Forment *et al.*, 2008). In general, these pipelines do not lend themselves to customisation in terms of the exchange of one analysis tool for another that is more suitable for an experiment, and often requires modifying various scripts in order to successfully replace a tool. The

need therefore exists for a bioinformatics workflow environment, where very complex analysis pipelines can be built *ad hoc* from a repository of tools, and these pipelines can then be executed with different datasets and parameters as input, and together with the results, shared with collaborators (Ludäscher *et al.*, 2005; Taylor *et al.*, 2007).

A successful bioinformatics analysis and workflow system needs to meet a diverse set of requirements. First, the initial development hurdle required to extend the system needs to be intuitive and relatively easy, it needs to be easily deployable and maintainable, scalable to various computational environments systems, as well as having a user-friendly interface for the users. The bioinformatics community currently employs a diverse range of tools and programming languages to develop analysis tools geared towards biological data mining. Traditionally, dynamic scripting languages, such as `Python`, `PERL`, `PHP` and `RUBY` have been used with great success in building complex analysis portals and resulted in large "Bio*" community projects developing around these languages (Chapman and Chang, 2000; Stajich *et al.*, 2002; Goto *et al.*, 2003; Holland *et al.*, 2008 and http://www.openbio.org). The aim of these communities can be summarized as providing a standard set of tools, or modules to perform common bioinformatics tasks. These tasks generally involve parsing results from popular analysis tools, connecting to the application programming interface (API) of a widely-used analysis tool, or converting between different biologically relevant file formats. The extensive use of these scripting languages in the bioinformatics community can be attributed to the lower entry level knowledge required when compared to compiled languages such as `C`, `C++` and `JAVA` when learning the language. This is evident in the popularity of these languages in many introductory courses to bioinformatics (Cohen, 2003; Boyle, 2004). Ideally, a bioinformatics analysis pipeline system should be ignorant in terms of the language a particular tool is written in, and should leverage the community expertise in term of skills and experience when new tools and features needs to be added to the workflow framework.

The modern biologist and laboratory scientist should ideally interact with an analysis workflow system in such a way that the underlying hardware requirements and nuances of running a specific tool should be obscured from the user interface, enabling the researchers to focus on interpreting the results obtained.

The `Galaxy` workflow system (Giardine *et al.*, 2005), introduced in Section 1.5.1 meets a large number of the criteria mentioned above for a successful bioinformatics pipeline system, and was therefore selected to serve as the basis of a system which caters for next generation data management and analysis. `Galaxy` has the abillity to execute scripts or analysis programs concurrently on local computational resources, and do not require the use of remote resources to execute a specific job. Workflow systems such as `Taverna` (Oinn *et al.*, 2004) and `Kepler` (Ludäscher *et al.*, 2005) makes extensive use of remote servers and protocols to construct the workflows. With the limited bandwith available in South Africa during the lifetime of this project, these workflow systems were not considered as viable contenders for a base workflow system to extend. The `Ergratis` (Orvis *et al.*, 2010) system was only published in 2010, which effectively excluded it from being used in this study.

The aim of the chapter is firstly to develop automated analysis pipelines which will perform analysis related to the quality evaluation of mRNA-Seq reads, the *de novo* assembly of a gene catalog, develop an automated functional annotation pipeline and perform expression profiling of gene transcripts using mRNA-Seq short reads. Secondly, for each of the workflows developed, some key parameters that have an effect on the output of the different tools will be investigated, and recommendations provided as to what ranges of these parameters should be considered when performing some of the analysis steps. In order to fully describe the parameters, different mRNA-Seq datasets were used as input to the workflows. The workflows developed in this chapter were used to perform a succesfull *de novo* assembly and annotation of a gene catalog described in Chapter 3.

2.2. Materials and methods

2.2.1. BCBU Galaxy: Extending the public `Galaxy` framework

The `Galaxy` framework (Giardine *et al.*, 2005) served as the base of extention for the development of the uHTS sequence analysis workflows. The public framework already contains a wide range of NGS analysis tools, and these tools were used wherever possible to construct the workflows. When a specific

analysis tool was missing from the public server, the tool was added to the BCBU `Galaxy` server. The tools added to the BCBU server either consisted of third party applications, such as the `Velvet` assembler that were developed by external authors, or custom `Python` and `R` scripts that were developed specificaly for this project. The list of third party applications added to the BCBU server is provided in Table 2.1, and the newly developed tools added to the BCBU server in Table 2.3.

2.2.2. Illumina short-read base-quality evaluation workflow

The Illumina FASTQ quality evaluation was performed with scripts and tools already present in the `Galaxy` framework. The default installation of `Galaxy` already provides uHTS data analysis functionality focussed on mRNA-Seq quality evaluation. The workflow, "Illumina QC" evaluates the quality of the bases from the forward and reverse reads from an Illumina paired-end run. The output from the workflow includes a bar chart of the distribution of base quality values for every base in the sequenced mRNA-seq dataset. The workflow also produces a summary of the `FASTQ` statistics file, which reports the number of reads in the lane, the number of bases, and the number of unknown bases in the run. The quality control tools enable users to evaluate the quality values of especially the 3' end of bases in the input dataset, and make informed decisions for trimming bases from a dataset for use in downstream analysis.

2.2.3. *De novo* transcriptome assembly workflow

The *de novo* transcriptome assembly workflow made use of the *de Bruijn* graph-based assembler `Velvet`, and a `FASTA` statistics calculation script from the `cndsrc` package[1] to guide the user towards steps needed to perform a transcriptome assembly. Transcriptome assembly is not a straigh-forward process, and during the workflow construction the effect of multiple parameters regarding the input dataset, such as sequenced read length and the effect of paired end reads, as well as the effect that different parameters provided to the assembler have on the final assembly were evaluated. A 76 bp *Eucalyptis grandis* Illumina-sequenced mRNA-seq dataset was used to illustrate the effect of these parameters. This dataset was trimmed to illustrate the effect various input data lengths (50 bp to 76 bp),

[1] http://www.biostat.wisc.edu/~cdewey/software.html, included in the `Galaxy` framework as the "`faLen`" tool

Table 2.1: Third party applications that were added to the BCBU `Galaxy` server instance. The category column indicates the location of the tool in the BCBU server, and the reference column describes the publication of the tool, or where applicable, the software package that the tool is part of.

Name	Category	Description	Reference
Exonerate alignment	Alignment	Alignment of EST or cDNA sequence to a target genome sequence	Slater and Birney (2005)
BLAST2GO pipeline	Annotation	Executes the b2gPipe command line interface of the BLAST2GO tool, requires a local installation of the BLAST2GO package and databases	Conesa *et al.* (2005)
BLASTXML2 BLAST2GO	Annotation	Re-formats BLAST results in XML format to a format required by the BLAST2GO application	Developed by lmanchon@univ-montp2.fr, open source
InterProScan	Annotation	Runs the InterProScan analysis tool, requires the installation of all the required InterPro datasets. Currently optimised to utilise 16 cores on a single server	Zdobnov and Apweiler (2001)
BLAST	BLAST	Performs a BLAST against one of the puclic databases available locally	Altschul *et al.* (1990)
BLAST two FASTA files	BLAST	Allows users to upload to fasta files, creates the BLAST databases on demand, and performs a BLAST analysis	Altschul *et al.* (1990)
Circoletto BLAST visualisation	BLAST	Makes use of the Circoletto application to view BLAST results in text format	Darzentas (2010)
faLen stats	FASTA tools	Calculates the N50, min, max, 1st and 3rd Quartile, mean and median sequence lengths from a fasta file	http://www.biostat.wisc.edu/~cdewey/software.html
FASTQ shuffleseq	FASTQ tools	Shuffles two FASTQ files into one file, required by the Velvet assembler	Zerbino and Birney (2008)
GenScan	Gene Predictors	Calls the GenScan tool on a fasta file containing protein sequences	Burge and Karlin (1997)
Velvet assembly	NGS tools	Performs a Velvet assemble on a FASTQ file	Zerbino and Birney (2008)
Multiple Velvet assemblies	NGS tools	Allows a series of Velvet assemblies with a range of parameters	Zerbino and Birney (2008)
Oases assembly	Development	Performs an Oases assembly on a FASTQ file	Zerbino *et al.*, unpublished
DEGseq	Development	Calculates differential expression between lists of genes using FPKM as the measure of expression	Wang *et al.* (2010*a*)
Muscle alignment	Development	Uses Muscle to perform multiple sequence alignments	Edgar (2004)

Table 2.3: A list of tools newly developed to complement the existing tools available in the BCBU `Galaxy` server. The tools include `R` and `Python` scripts that perform specific analysis, or convert files between different formats that serve as input to the next tool in the analysis pipeline.

Name	Category	Description
Exonerate targetgff2gff3	Alignment	Converts the gff and text output from Exonerate to the GFF3 format
InterProScan RAW format converter	Annotation	Re-formats InterProScan RAW results to either a txt or XML based format. The XML format is required by the BLAST2GO application
InterProScan2 BLAST2GO	Annotation	Converts InterProScan XML results to a directory format required by theBLAST2GO application
Parse BLAST XML	BLAST	Provides the facility to extract custom fields from a BLAST XML file
Convert gff3 to gtf	Convert formats	Produces the compact GTF format form a GFF3 file
Convert qseq to fastq	Convert formats	Converts an Illumina qseq file to a fastq file
Extract FASTA region	FASTA tools	Extract reqions from a FASTA file
Reverse fasta sequence direction	FASTA tools	Reverse all the sequences in the FASTA file
Retrieve longest transcripts	FASTA tools	Parses the OASES assembler assembly files, retrieves the longest assembled transcripts
Rename FASTA entries	FASTA tools	Rename the FASTA entries
Summary of FASTQ Summary statistics file	NGS tools	Calculates the number of usable bases, the number of A, C, G and T bases and the theoretical base yield from a FASTQ summary statistics file
SAM QC stats	NGS tools	Calculates the number of reads that map as pairs, as singles, and uniquely from a SAM file
TopHat QC stats	NGS tools	Calculates the same statistics from a TopHat generated SAM file
SNP filter	SNP tools	Filter a pileup file with more stringent constrains, such as the minimum distance between two SNPs
SNP summary	SNP tools	Generates a summary of a pileup file. Includes the average distances between SNPs

different sequencing approaches (paired *vs.* single end sequencing), and different assembly parameters (kmer, expected coverage, and coverage cutoff parameters) on the same dataset. The different assemblies obtained from running multiple iterations of the workflow were compared with each other by a robust scoring algorithm that takes the number of contigs and length distribution of the contigs into account to evaluate an assembly. The workflow is provided in the BCBU `Galaxy` server as the "Velvet assembly pipeline". The workflow also discusses ways to evaluate the contig contiguity of the assembled datasets against known transcript sequences using `BLAST` (Altschul *et al.*, 1990) and related tools.

2.2.4. Annotation of predicted protein sequences workflow

An annotation workflow that focus on the functional annotation of translated cDNA sequences by widely used tools such as `BLAST2GO` (Conesa *et al.*, 2005) and `InterProScan` (Zdobnov and Apweiler, 2001) was developed. The pipeline predicts protein sequences from the input cDNA sequence file, and assigns functional annotations such as Gene Ontology (Gene Ontology Consortium, 2001), KEGG (Ogata *et al.*, 1999) and PFAM (Finn *et al.*, 2010) to the predicted protein sequences. The workflow relies on finding homologous sequences in model organisms, on which the functional annotations is based. The workflow is made available as the "Annotation pipeline" workflow in the BCBU Galaxy server. The various components in the workflow were used to perform the functional annotation of a *de novo* assembled Eucalyptus transcriptome descibed in Chapter 3. The results from the annotation pipeline can easily be imported into a third party application database, such as the `Eucspresso` system (Chapter 4) for the visualisation of results.

2.2.5. Expression profiling using Illumina mRNA-Seq short reads workflow

One of the main uses of mRNA-Seq data is transcriptional profiling of expressed gene products across the genome. Steps involved in calculating transcript expression include mapping reads to a target genome, inferring read coverage, and calculating the number of short read fragments that map to a specific genomic position, albeit a known gene region or an unknown genomic region. The workflow makes use of the `TopHat` aligner (Trapnell *et al.*, 2009) to map short-reads to a target genome sequence,

and the CUFFLINKS (Trapnell *et al.*, 2010) program used to calculate the normalised expression value of the gene in fragments per kilobase of reads mapped per million mapped reads. The workflow describes the gene expression calculation of a genome sequence where the only resource to define the gene boundaries in the genome is a set of EST data. The EST dataset is aligned to the genome with the EST2GENOME mode of the EXONERATE (Slater and Birney, 2005) package. After the genomic positions of the putative gene models were identified, differentially expressed genes between two sets of tissues were identified with the R-package DEGseq (Wang *et al.*, 2010*a*).

2.3. Results and discussion

Several next-generation data analysis workflows were constructed and saved in the BCBU Galaxy server as re-usable workflows, specifically with the aim to evaluate the quality of initial Illumina mRNA-Seq input data, the parameters which influence the assembly of transcriptome datasets, annotation of predicted protein sequence datasets, and expression profiling of transcriptome making use of mRNA-Seq short-reads. The sections describing each of the workflows consist of an overview or aim of each workflow, a short discussion on the components of the workflow, and a description of the effect of the parameters that can serve as input to the workflow on the results from the analysis pipeline.

2.3.1. Extending the Galaxy framework

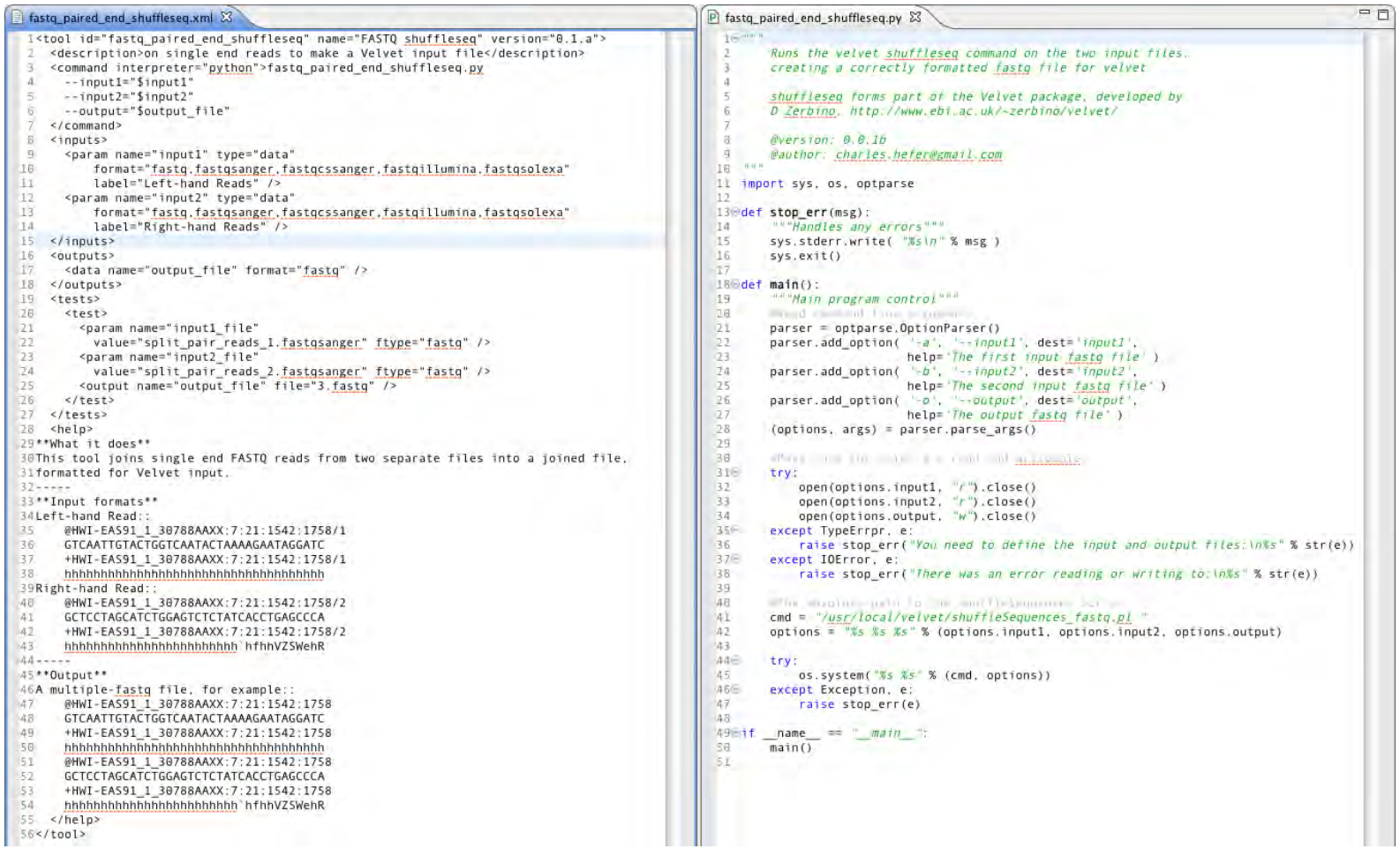
The Galaxy framework serves as a container to host data analysis tools. The framework has the ability to sequentially execute various analysis tools on specific input datasets, selected by the user. Each tool contained in the framework is represented by a XML file, which specifies the input parameters that are sent to the tool during programmatic execution. Jobs can be executed on a local server, or submitted to a job handler server, such as the Sun grid engine (SGE, http://http://wikis.sun.com/display/GridEngine/Home) that executes jobs on a cluster-based computing platform. The Galaxy server automatically keeps track of the status of the submitted jobs, and the results are displayed in the server (the "histories" pane) after the job has been completed. The server also enables the user to construct workflows, or sequential steps

that need to be performed given an input data set. The following section describes the steps required to add a very basic analysis tool to the Galaxy framework.

The results from paired-end sequencing on the Illumina platform, consist of two FASTQ quality files, one for reads sequenced in the 5' to 3' (forward reads), and one for reads oriented in the 3' to 5' direction (reverse reads). The tool, named "shuffleseq", joins two FASTQ formatted files from an Illumina file into one file, with the reads in the final file sorted in an alternate fashion of forward and reverse reads. This "shuffled" FASTQ file is a required format for the Velvet assembler, and the shuffleseq executable script forms part of the Velvet assembler distribution.

To extend the BCBU Galaxy server to contain the "shuffleseq" script, an XML file needs to be created that registers the tool in the server, and renders an interface to select the tool. The shuffleseq XML file is presented in Figure 2.1, and consists of the following sections. Lines 3-7 specify the command to be executed, and allows the definition of the names of the input parameters, as well as the required format of the input datasets (lines 9-15). The name and format of the output file to store in the database is defined in lines 16-19. Galaxy has a default interface to define automated software tests, and encourages test-driven development, which will not be discussed here. These automated tests can then be run during the development phase of the when adding a tool to ensure that pre-calculated results are obtained with a with pre-defined set of input parameters. In this example, the input parameters for the tests are defined in lines 21-24, and the expected output for a successful test in line 25. Documentation regarding the functionality of the tool is provided from lines 29 to 46 of the XML file. This XML file renders the interface shown in Figure 2.2.

The executable, in this case the Python script named "fastq_paired_end_shuffleseq.py", is presented in Figure 2.1. Lines 13-16 of the file handle error reporting, and lines 21-38 contain error handling code to ensure that the input and output files are readable and writeable. The execution of the PERL script occurs on line 45, surrounded again by some error handling code if the execution of the script fails. The crucial link between the XML file and the executable is defined in the <command> tag of the XML file, and the input parameters or options in the Python script. In effect, the Galaxy execution engine passes the



fastq_paired_end_shuffleseq.xml

```
<tool id="fastq_paired_end_shuffleseq" name="FASTQ shuffleseq" version="0.1.a">
  <description>on single end reads to make a Velvet input file</description>
  <command interpreter="python">fastq_paired_end_shuffleseq.py
    --input1="$input1"
    --input2="$input2"
    --output="$output_file"
  </command>
  <inputs>
    <param name="input1" type="data"
        format="fastq,fastqsanger,fastqcssanger,fastqillumina,fastqsolexa"
        label="Left-hand Reads" />
    <param name="input2" type="data"
        format="fastq,fastqsanger,fastqcssanger,fastqillumina,fastqsolexa"
        label="Right-hand Reads" />
  </inputs>
  <outputs>
    <data name="output_file" format="fastq" />
  </outputs>
  <tests>
    <test>
      <param name="input1_file"
        value="split_pair_reads_1.fastqsanger" ftype="fastq" />
      <param name="input2_file"
        value="split_pair_reads_2.fastqsanger" ftype="fastq" />
      <output name="output_file" file="3.fastq" />
    </test>
  </tests>
  <help>
**What it does**
This tool joins single end FASTQ reads from two separate files into a joined file,
formatted for Velvet input.
-----
**Input formats**
Left-hand Read::
    @HWI-EAS91_1_30788AAXX:7:21:1542:1758/1
    GTCAATTGTACTGGTCAATACTAAAAGAATAGGATC
    +HWI-EAS91_1_30788AAXX:7:21:1542:1758/1
    hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
Right-hand Read::
    @HWI-EAS91_1_30788AAXX:7:21:1542:1758/2
    GCTCCTAGCATCTGGAGTCTCTATCACCTGAGCCCA
    +HWI-EAS91_1_30788AAXX:7:21:1542:1758/2
    hhhhhhhhhhhhhhhhhhhhhhhh`hfhhVZSWehR
-----
**Output**
A multiple-fastq file, for example::
    @HWI-EAS91_1_30788AAXX:7:21:1542:1758
    GTCAATTGTACTGGTCAATACTAAAAGAATAGGATC
    +HWI-EAS91_1_30788AAXX:7:21:1542:1758
    hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
    @HWI-EAS91_1_30788AAXX:7:21:1542:1758
    GCTCCTAGCATCTGGAGTCTCTATCACCTGAGCCCA
    +HWI-EAS91_1_30788AAXX:7:21:1542:1758
    hhhhhhhhhhhhhhhhhhhhhhhh`hfhhVZSWehR
  </help>
</tool>
```

fastq_paired_end_shuffleseq.py

```
"""
    Runs the velvet shuffleseq command on the two input files,
    creating a correctly formatted fastq file for velvet

    shuffleseq forms part of the Velvet package, developed by
    D Zerbino, http://www.ebi.ac.uk/~zerbino/velvet/

    @version: 0.0.1b
    @author: charles.hefer@gmail.com
"""
import sys, os, optparse

def stop_err(msg):
    """Handles any errors"""
    sys.stderr.write( "%s\n" % msg )
    sys.exit()

def main():
    """Main program control"""
    parser = optparse.OptionParser()
    parser.add_option( '-a', '--input1', dest='input1',
                       help='The first input fastq file' )
    parser.add_option( '-b', '--input2', dest='input2',
                       help='The second input fastq file' )
    parser.add_option( '-o', '--output', dest='output',
                       help='The output fastq file' )
    (options, args) = parser.parse_args()

    try:
        open(options.input1, "r").close()
        open(options.input2, "r").close()
        open(options.output, "w").close()
    except TypeErrpr, e:
        raise stop_err("You need to define the input and output files:\n%s" % str(e))
    except IOError, e:
        raise stop_err("There was an error reading or writing to:\n%s" % str(e))

    cmd = "/usr/local/velvet/shuffleSequences_fastq.pl "
    options = "%s %s %s" % (options.input1, options.input2, options.output)

    try:
        os.system("%s %s" % (cmd, options))
    except Exception, e:
        raise stop_err(e)

if __name__ == "__main__":
    main()
```

Figure 2.1: An example of code developed to extend the `Galaxy` framework with the "`shuffleseq`" tool. The .xml file (left) defines the interface to the tool, and specifies the input and output format requirements. The `Python` script (.py) on the right pass the input and output parameters from the xml file to the `Perl` script, located on the file system. This example illustrates the ease of extending the `Galaxy` framework. In just over 100 lines of code, additional functionality was added to the framework.

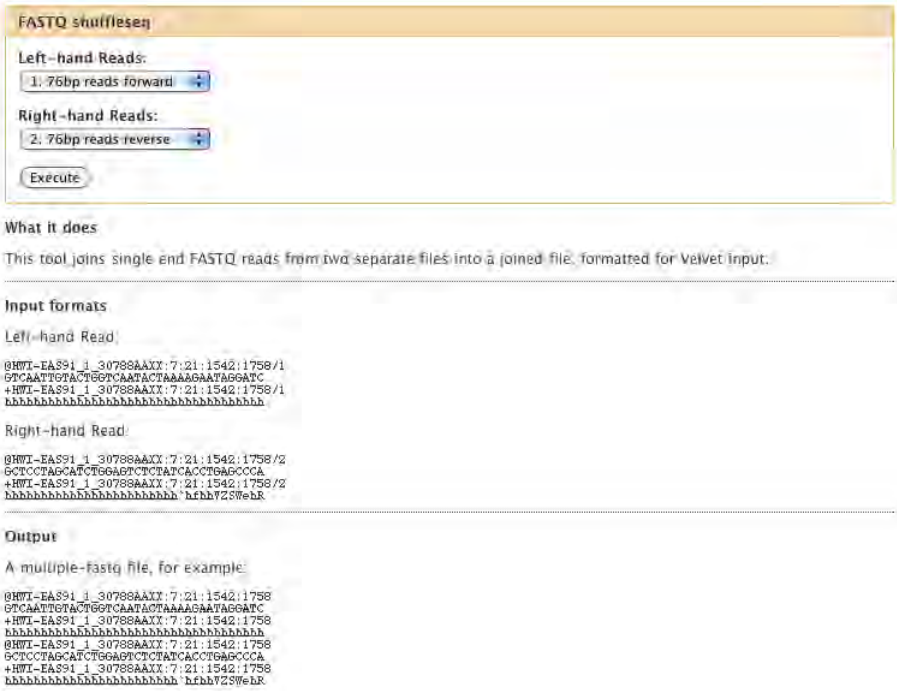


Figure 2.2: The interface of the `FASTQ` shuffleseq tool described in the fastq_shuffleseq.xml file, as rendered by `Galaxy`. The interface provides the user to select buttons to select the forward (left hand) and reverse (right-hand) reads that will be "shuffled" into a single file as output. A short description on the function of the tool, and an example of input formats is also provided.

following parameters to the Python script during execution: `python fastq_paired_end_shuffleseq.py --input1=path/to/input1/file --input2=/path/to/input2/file --output=/path/to/output/file`, and expects the result file to be present in the output file location. The PERL script could have been called directly by the XML file, but this example illustrates that any executable command can be wrapped in the Galaxy framework and executed.

2.3.2. Quality assesment of Illumina short-reads

The quality control of experimental data forms an integral part of any analysis pipeline. A workflow dedicated to calculating the average base quality, the number of usable bases and the total number of reads from an Illumina mRNA-Seq lane was developed (Figure 2.3, which is available as the Illimina QC workflow in the BCBU Galaxy server,). The typical yield in terms of bases from an Illumina GA IIx run is reported by the company to be between 37 Gbp and 45 Gbp (January 2011, http://www.illumina.com), and these ranges were observed in a recently produced dataset (Table 2.5).

The FASTQ file format stores the quality associated with every sequenced base of every read in the FASTQ file. Reads produced with the Illumina platform tend to show a drop in the quality of bases as the read length increases (Figure 2.4). In an attempt to filter erroneous sequences from dataset, it is often required to remove or trim a subset of bases from the 3' end of each read. In the case of paired-end sequencing, the reverse reads also tend to have lower quality values associated with the bases when compared to the forward reads (Table 2.5). Trimming the last few bases from the 3' end of the reads can improve the number of reads that aligns to a target sequences (read mapability), prevent the occurrence of false positives during SNP identification, and prevent misassembled contigs. The effect of read trimming will be further addressed in the sections regarding *de novo* assembly (Section 2.3.3) and read mapping to a reference genome (Section 2.3.5 on page 73). A good guideline for trimming the reads is to use an error rate of 1 in 100 bases during assembly and read mapping, which translates to a Phred quality score cutoff of 20. Several tools already exists in the public Galaxy server to trim the end of

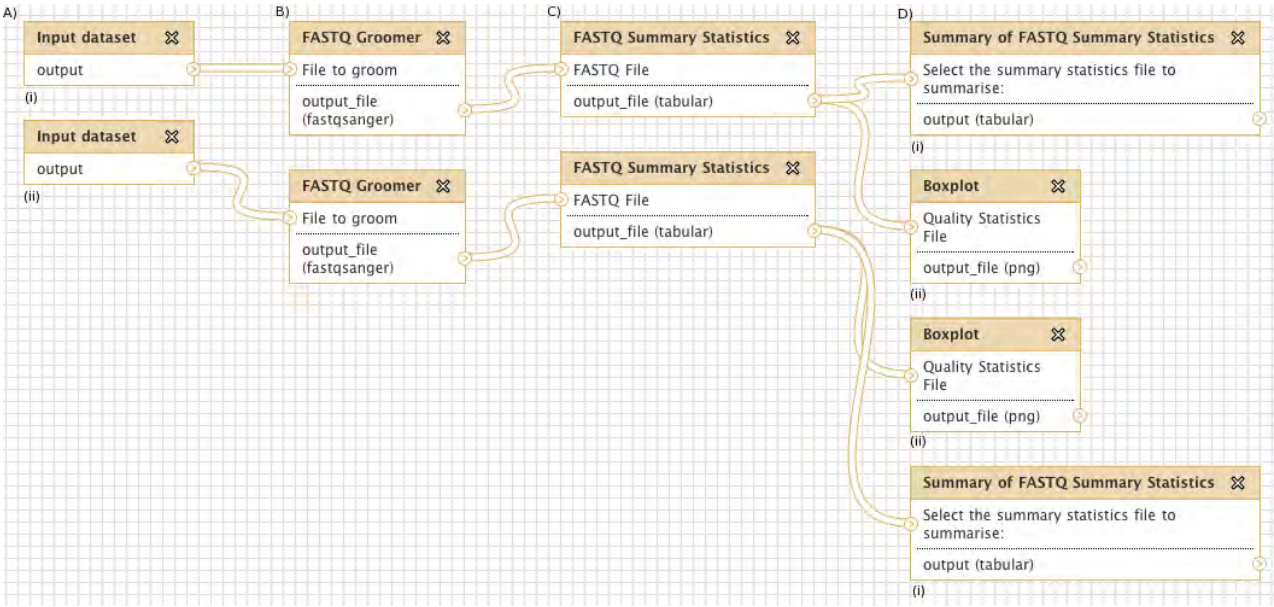


Figure 2.3: The Illumina read quality assesment pipeline. The first step after defining the input datasets (two `FASTQ` formated files (A), one that consists of the forward reads (A i) of a paired end run, and that consists of the reverse reads, A ii) is to convert the `FASTQ` values from the Illumina (1.3+) version to the `FASTQSANGER` format (`FASTQ` Groomer, B). Quality statistics per base are then calculated (`FASTQ` Summary Statistics, C), and a graphical summary of all the bases in the lanes produced (Boxplot, presented in Figure 2.4, D ii). From the `FASTQ` summary statistics, the number of reads and the number of bases present in each lane can be calculated (Summary of `FASTQ` summary statistics, Table 2.5, D i) .

Table 2.5: The theoretical and usable base (bases identified as A, G, C and T) yield for six Illumina GA IIx 76 bp paired-end lanes. The theoretical yield was calculated as the total reads per lane times the read length. On average, the forward reads yielded 97.53% of the theoretical bases to pass the internal quality control performed by the sequencing center, while 96.83% of the reverse bases were useable. If seven usable lanes are considered per flowcell, an estimated 42 Gbp would have been produced from these lanes (please note that these lanes were not produced from the same flow cell).

Tissue	Read length	Total reads	Theoretical base yield	Useable base yield	Useable Gbp
Young leaf (a)	76 bp X 76 bp	38 675 726 (X 2)	2 939 355 176 (X 2)	5 714 978 949 (97.56% fwd, 96.87% rev)	5.71
Young leaf (b)	76 bp X 76 bp	40 644 094 (X 2)	3 088 951 144 (X 2)	6 005 687 472 (97.56% fwd, 96.86% rev)	6.01
Young leaf (c)	76 bp X 76 bp	40 603 294 (X 2)	3 085 850 344 (X 2)	5 999 955 671 (97.57% fwd, 96.86% rev)	6.00
Xylem (a)	76 bp X 76 bp	40 626 119 (X 2)	3 087 585 044 (X 2)	6 001 212 765 (97.54% fwd, 96.83% rev)	6.00
Xylem (b)	76 bp X 76 bp	41 212 187 (X 2)	3 132 126 212 (X 2)	6 084 735 293 (97.50% fwd, 96.76% rev)	6.00
Xylem (c)	76 bp X 76 bp	38 363 392 (X 2)	2 915 617 792 (X 2)	5 664 669 869 (97.48% fwd, 96.81% rev)	5.66

the reads based on either read length (the `FASTQ Trimmer` by column tool in `Galaxy`), or base quality (`FASTQ Quality Trimmer` by sliding window) .

2.3.3. *De novo* transcriptome assembly using Illumina mRNA-Seq data

One of the main aims of this study was to perform a *de novo* assembly of a gene catalog from mRNA-Seq data generated from a range of primary and secondary *Eucalyptus* tissues (Chapter 3). A *de novo* assembly pipeline to achieve this goal typically consists of firstly formatting the input data to satisfy the requirements of the assembler, secondly perform the assembly, and finally evaluate the assembly (Figure 2.5, which is available as the `''Velvet assembly''` pipeline in the BCBU server). `Velvet` (Zerbino and Birney, 2008), the assembler used in this workflow, requires paired-end reads to be in a format where the first read of a fragment is directly followed by the second read of the fragment, as opposed to some other assemblers which require the reads from the same fragment to be in the same order, but in two different files. The "`shuffleseq`" tool, a script provided with the `Velvet` assembler and used to create the single file format, was wrapped in the BCBU `Galaxy` environment to allow for workflow integration (Section 2.3.1).

Input parameters of note that are specified for use during the graph-creation step of the `Velvet` assembly include the choice of kmer (Section 1.4), and the flag that specifies whether the input datasets are in paired-end format. During the graph traversal step, the expected coverage parameter and a coverage cutoff parameter is specified. The coverage cutoff parameter is used by the assembler to restrict highly connected nodes in the graph (repeat regions) from dominating the assembly. Changing each of these parameters results in differences in the properties of the final set of contigs produced from an assembly (see Section 1.4 for an overview of graph based *de novo* assemblers).

Currently no standardised protocol exists for steps needed to evaluate the success of a transcriptome assembly. Unlike the assembly of a genome sequence, where the aim is to assemble a single contig from all the reads provide, the aim of a transcriptome assembly can be viewed as the assembly of multiple, short fragments that represent mRNA molecules. The coverage of genome derived data is also distributed more

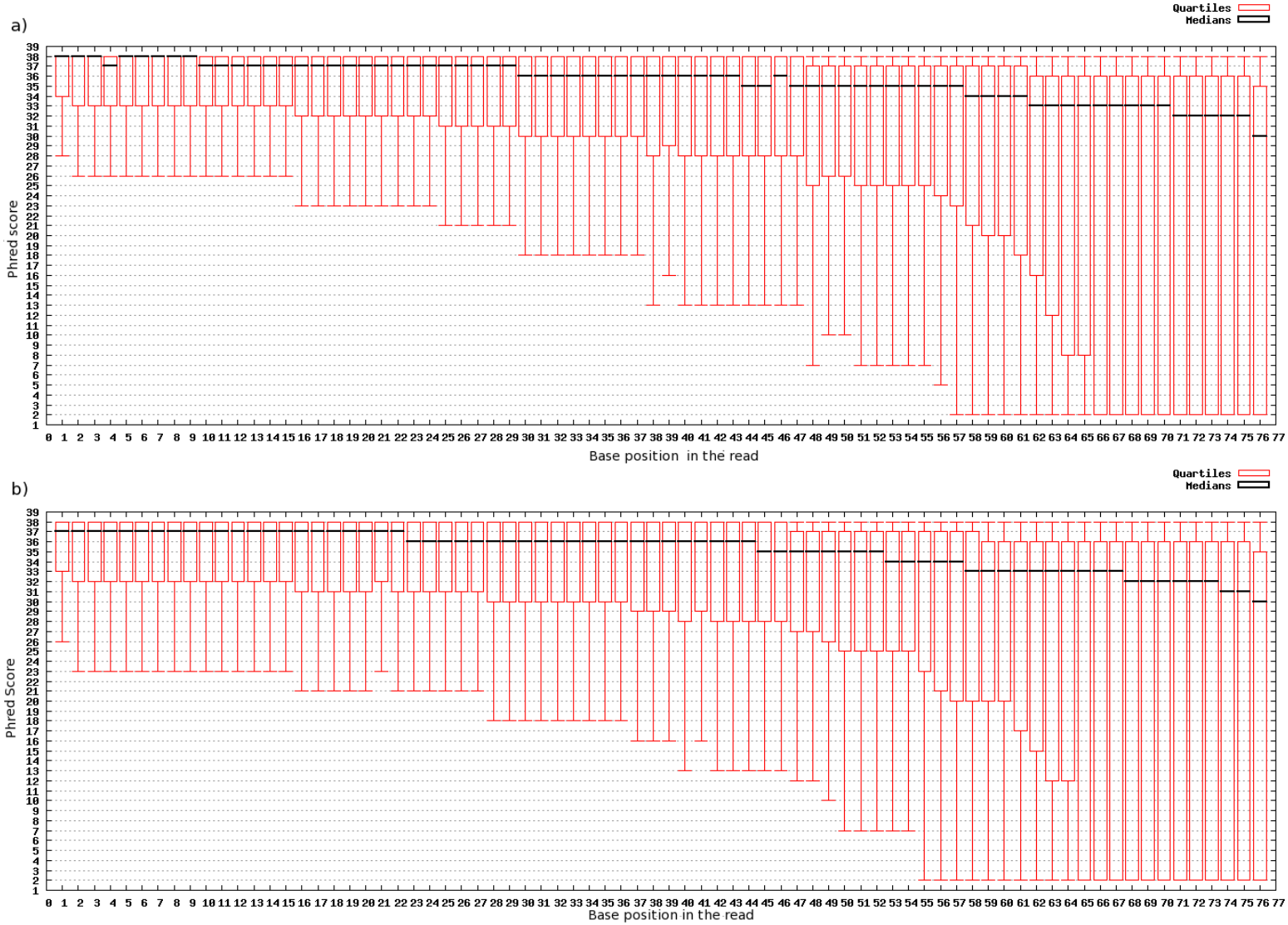


Figure 2.4: An example of `FASTQ` quality scores obtained from a 76 bp Illumina GAII paired-end run. The quality of each base is plotted on the y-axis, with the position of the base on the sequence on the x-axis. This lane contained around 38 million reads (2.8 billion bases) in the forward (a), and 38 million reads in the reverse (b) direction. The median (black line) and the standard error bars (red bars) for all the reads are shown in both directions. A quality drop is observable for bases closer to the 3' end (sharp increase in base-quality variation from base 56-58) and removing these bases with lower qualities might influence read mapping and assembling strategies.

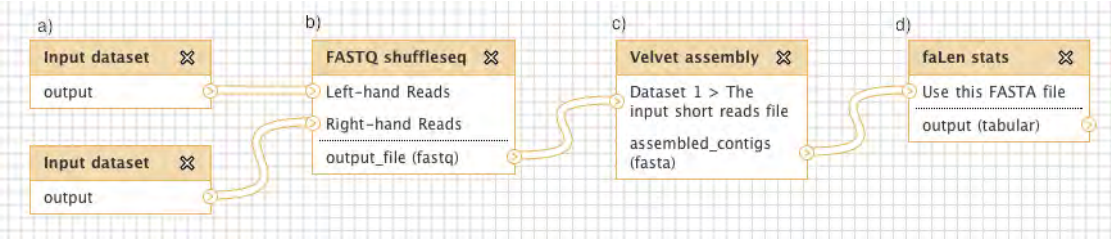


Figure 2.5: A `Galaxy` workflow which performs a *de novo* assembly with the Velvet assembler. The default input data (a) for this workflow is the forward and reverse FASTQ (fastqsanger) formatted mRNA-Seq reads. The reads are then reformatted with a "shuffleseq" script (b) to the correct input format for paired-end reads as required by Velvet, and the assembly is performed by Velvet (c). A script to calculate the N50, longest, mean and average sequence lengths is then run on the assembled fasta file.

evenly across the genome, with exceptions of the repeat regions, while the transcriptome data has varied coverage across a single transcript and between multiple transcripts. The variation in transcript coverage fluctuates due to the number of transcripts present in then sample mRNA pool, and the variation across a transcript has been postulated to be due to the folding patterns of the mRNA trancripts in the cell (Mortazavi *et al.*, 2008). There are several descriptive statistics available to assist in selecting the best possible assembly, namely the number of bases in the contigs (sum), the number of contigs (N), the contig length spread (minimum and maximum contig length, 1st and 3rd quartile length, mean and median length), and the N50 value. The N50 value is calculated as the contig length where 50% of the bases in the assembly are present in contigs of the reported length, or longer. A scoring function to empirically select the best assembly has been discussed on the Velvet users group mailing list[2], and defined as: $\frac{(N50_{all} * N_{long})}{Sum_{all} + log(Sum_{long})}$, where the long values are calculated for contigs longer than 1 000 bp. A higher score indicates a higher ratio between the bases located in the longer reads in the dataset and the bases assigned to short contigs. This scoring metric was also discussed on the community portal SeqAnswers [3], and later implemented in an optimisation script for `Velvet` as a third party script, and although this scoring function has been defined for genome assemblies, it provides a good guideline when applied to transcriptome assemblies. In the sections discussed below, the score of the assemblies were calculated with the scoring function to give an indication of the function's performance on multiple assembled datasets.

[2] http://listserver.ebi.ac.uk/mailman/listinfo/velvet-users
[3] http://seqanswers.com

Table 2.6: Velvet assembly statistics for a single lane of paired 76 bp sequences from *Eucalytpus* xylem tissue reads trimmed to different lengths (50 - 76 bp). The same assembly parameters (kmer 41) were used to illustrate the effect of sequence length on the assembly. Assemblies with the longest reads as input (65, 70 and 76 bp) generated the largest (N) assemblies, and the longest single contigs (max) were assembled with the 65 bp reads. The scoring function also indicates that the longer input reads generate better assemblies, except when the last 6 bp which were error prone are included. The 1 000 bp contig values (long contigs) used in the scoring function are presented in the Appendix I table A.1.

Read length (bp)	Number of contigs (N)	Sum of bases	Min (bp)	1st Quartile (bp)	Median (bp)	3rd Quartile (bp)	Max (bp)	Mean (bp)	N50 (bp)	Score
50	73 762	21 723 533	81	130	183	342	6 772	294.51	411	6.63
55	104 471	32 014 867	81	122	171	349	8 078	306.45	486	6.95
60	134 970	39 632 149	81	111	163	323	8 241	293.64	467	7.05
65	169 960	46 302 130	81	102	156	293	11 008	272.43	414	7.06
70	207 383	52 321 544	81	95	151	269	8 573	252.29	362	7.03
76	255 609	59 076 999	81	92	148	247	8 985	231.12	308	6.95

Low quality bases are generally present in the 3' end of Illumina reads (see Figure 2.4), and removing or trimming these reads tend to influence the subsequent assemblies. Assemblers using the *de Bruijn* graph approach, where kmers are used to find joins between reads and the high coverage paths between kmer nodes in the graph are used to assemble the contigs, have a higher tolerance towards low frequency erroneous bases in the input dataset (see Section 1.4). There also exists uncertainty about the optimal read length required to perform *de novo* transcriptome assemblies, and since longer reads require more reagents that influences the cost of sequencing this is an important consideration in project planning. Illumina mRNA-Seq paired-end reads from a deeply sequenced *Eucalyptus* xylem dataset were trimmed to a length ranging from 50 bp to 76 bp. The trimmed datasets were then assembled with the `Velvet` assembler (`Velvet assembly` workflow) with a defined kmer of 41 to determine the length of the input dataset reads that produced the best assembly. Table 2.6 indicates that longer reads produce longer individual contigs, but there is a decrease in overall assembly quality when the last 6 bp (low quality bases) of the 76 bp reads are not trimmed from the input dataset. The 55 bp assembly showed the largest N50 and the longest mean and median contigs, but if the additional ≈7 Mbp of sequence data gained

Table 2.7: Statistics for `Velvet` assembled contigs with a minimum length of 200 bp for a single lane of paired 76 bp sequences from *Eucalytpus* xylem tissue reads trimmed to different lengths. The values in parentheses indicate the same statistics obtained with the same dataset, but where the datasets were treated as single and not paired-end reads. The 1 000 bp contig values (long contig) used in the scoring function for the single end assemblies are presented in the Appendix A Table A.1.

Read length (bp)	Number of contigs (N)	Sum of bases	Min (bp)	1st Quartile (bp)	Median (bp)	3rd Quartile (bp)	Max (bp)	Mean (bp)	N50 (bp)	Score
50	33 475 (31 519)	16 411 541 (14 581 289)	200	268 (245)	365 (328)	570 (527)	6 772 (5 571)	490.26 (462.62)	562 (535)	6.68 (6.63)
55	42 934 (43 283)	23 989 757 (22 004 217)	200	278 (253)	403 (248)	672 (577)	8 078 (8 078)	558.76 (508.38)	693 (615)	7.03 (6.95)
60	49 152 (50 771)	28 489 587 (26 957 786)	200	275.5 (258)	407 (359)	689 (603)	8 241 (8 241)	579.62 (530.97)	733 (653)	7.19 (7.08)
65	55 059 (56 990)	31 759 222 (30 633 000)	200	272 (260)	398 (366)	676 (610)	11 049 (11 049)	576.82 (537.52)	730 (660)	7.23 (7.14)
70	60 039 (61 683)	34 307 077 (33 463 851)	200	270 (262)	394 (371)	662 (615)	11 008 (10 757)	571.41 (542.51)	718 (664)	7.25 (7.18)
76	64 713 (65 989)	36 602 687 (36 070 026)	200	268 (264)	389 (375)	652 (621)	9 925 (10 873)	565.62 (546.61)	705 (669)	7.26 (7.22)

by the 60 bp, or the additional ≈14 Mbp of data when the 65 bp input dataset is considered, those assemblies can certainly be considered when evaluating an assembly. The scoring function calculated on these datasets provide a ranking system for the assemblies, but ultimately the choice of read length depends on the discretion of the researcher. Assembled contigs of a length between 81 bp and 200 bp most likely consist of small fragments of larger contigs, or very rare low coverage transcripts, and an additional constraint can be applied to the assembled dataset that contigs need to have a least a length of 200 bp to be considered for downstream analysis and annotation (Table 2.7). Because the `Velvet` assembler was developed for the *de novo* assembly of genomes, not transcriptomes, alternative spliceforms will be lost during assembly since the assembler returns the longest graph of the most coverage in the final assembly.

The assembly of the various trimmed datasets were repeated with the two lanes of the paired datasets provided separately to the assembler, effectively re-fromatting the input data as two single-end datasets as oppose to a single paired-end dataset (Table 2.7, results in parentheses). Overall the single-end reads

did not perform worse than the paired-end assemblies, and even produced the same maximum length contigs in some cases. There is, however, a sampling bias in the data used for this single-end assembly, since the single-ends are not independently sampled fragments from the sequenced mRNA-Seq pool, but in fact represent sampled paired sequences. This simulated assembly of single end data thus does not represent the true effect of sequencing single-end *vs.* paired-end libraries, but rather reflects the difference in the assembler algorithm and the improvement achieved when enabling the paired-end flags. These values represent the practical best case scenario when single-end reads are used for assembly, and real independently sampled single-end assemblies will thus perform worse than reported here.

The graph traversing step of `Velvet` has multiple parameters that will ultimately affect the set of contigs assembled. One of the most notable parameters is the effect of kmer size (kmer of 41 - 63 bp) on the different assemblies, as presented in Table 2.8. The choice of kmer for assembly will vary with a change in length of the input reads, as well as the inherent sequence properties of the tissue or organism sampled. The scoring function defined above relates well to the a combination of the N50 value and the descriptive statistics of the assembly, and plotting the different assembly statistics as a fraction of the highest value of each parameter show that the scoring function can be successfully used as a guideline to select the best assembly for further analysis (Figure 2.6). The figure makes use of a normalised value for some descriptive statistics (N50, Sum and Score in Figure 2.6A) achieved during a specific kmer assembly according to the maximum value optained (y-axis) across all kmers (x-axis), and can be used to graphically select the set of kmers that produce an assembly with a high score. The kmer of 51 (k51) produced an assembly containing 69 485 contigs, ranging from 200 bp to 8 451 bp in lenght. The scoring algorithm assigned a score of 7.13 to the k51 assembly, but the k49, k53 and k55 assemblies also achieved a high score. The best choice of a kmer to use in further assemblies depends on whether full length transcripts were assembled during any of these kmer assemblies, but the scoring algorithm does provide some measure of comparison between the assemblies.

The effect of two additional parameters during the graph traversal step, the expected coverage and coverage cutoff value, on the results from multiple assemblies is presented in Figure 2.7. The expected

Table 2.8: `Velvet` assembly statistics for a single lane of paired 76 bp sequences from *Eucalytpus* xylem tissue. The same input parameters were used, except for the kmer-value to obtain these assemblies. Note a general trend that fewer contigs (N) and fewer total bases (Sum) are present in higher kmer assemblies, indicating that more contigs might be joined with longer kmers. The descriptive statistics in terms of median, mean and N50 values peak around the mid kmer (k49-k55) sizes. The assembly score was calculated to critically evaluate overall score of an assembly. All contigs longer than 200 bp were included in the analysis.

Kmer	Number of contigs (N)	Sum of bases	Min (bp)	1st Quartile (bp)	Median (bp)	3rd Quartile (bp)	Max (bp)	Mean (bp)	N50 (bp)	Score
k41	84 428	38 627 991	200	249	334	523	8 985	457.53	518	7.00
k43	81 527	38 434 796	200	250	339	538	8 862	471.44	543	7.05
k45	78 748	37 908 219	200	250	342	548	8 451	481.39	560	7.08
k47	75 732	37 110 906	200	250	345	557	8 451	490.03	576	7.10
k49	72 320	36 115 097	200	249	349	573	8 451	499.38	598	7.12
k51	69 485	35 124 810	200	250	351	581	8 451	505.50	613	7.13
k53	66 029	33 652 392	200	249	353	587	8 065	509.66	621	7.12
k55	62 391	31 953 361	200	248	351	593	8 582	512.15	632	7.12
k57	58 921	30 071 960	200	247	350	593	8 277	510.38	631	7.09
k59	54 966	27 877 831	200	246	349	591	9 622	507.18	626	7.04
k61	51 057	25 518 959	200	245	346	585	7 152	499.81	613	6.98
k63	46 684	22 658 706	200	244	338	563	6 360	485.36	585	6.89

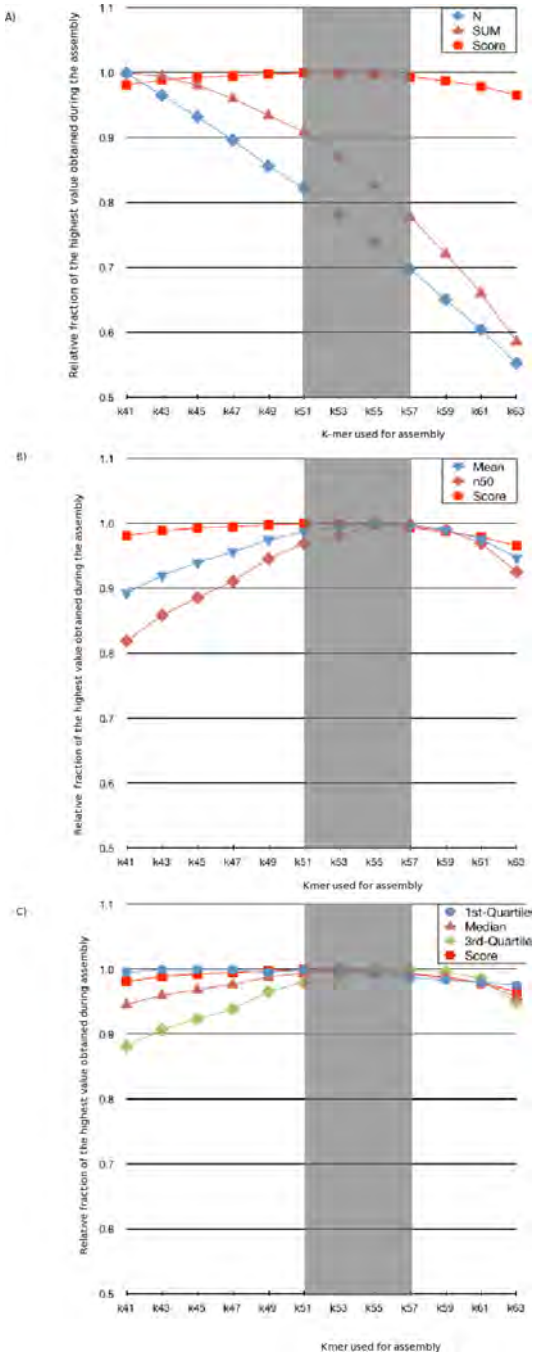


Figure 2.6: The assembly scoring function is a robust measure to select the kmer of the best `Velvet` assembly. The y-axis represents the value of a certain descriptive statistic obtained for a kmer as a fraction of the maximum value of that statistic (y-axis) across all kmers (x-axis). The scoring function is not sensitive to changes in total base count and number of contigs (a), and correlates well with the N50 and mean values (b) as well as the other descriptive statistics (c). The graphs were normalised so that the values correspond to a fraction of the maximum value achieved for each parameter across all kmer assemblies shown.

coverage parameter performs two key functions during the assembly. First, it is required to activate the paired-end read resolution function of `Velvet` (as stated in the `Velvet` manual), which programatically makes use of the insert size between pairs to join contigs; and secondly it assists in finding the optimal path through the nodes in the graph of kmers by searching for nodes in the graph that correspond to the expected coverage value. This assistance provides the assembler with a naive approach to filter the nodes in the graph based on the node coverage in order to determine optimal contigs (Zerbino and Birney, 2008). This approach is especially useful when a genome sequence is assembled, since the sequence coverage from a lane of genomic short-read data should have near uniform coverage, bar the repeat regions of the genome that should have higher coverage. The inherent properties of mRNA-Seq data, where coverage varies between transcripts based on the amount of transcript present in the sampled mRNA pool and across a single transcript based on the mRNA molecule's folding properties, the occurrence of alternative splicing, and the known 3' bias exhibited by mRNA-Seq technologies render this parameter less useful during transcriptome assemblies.

Figure 2.7A (left), indicates that for a transcriptome assembly, high expected coverage values produce the best possible assembly when evaluating the results based on the scoring function. The results were obtained by performing various assemblies with a constant set of parameters (insert length between paired reads = 150 bp, the coverage cutoff = 10X, and the kmer set to 51), but increasing the expected coverage value from 0 to 1 000 with each subsequent assembly. The graph shows that a higher expected coverage value can produce assemblies with longer mean length and N50 values (an expected coverage of 0 produced an assembly with an average N50 length of 1 018 bp, only 55% of the N50 value achieved by the assembly where the expected coverage was set to 1 000 (N50 = 1 854 bp)). These estimations of the expected coverage value are needed to assemble highly expressed transcripts to a complete length, and will remove lowly expressed transcripts from the assembly.

The coverage cutoff value effectively screens the contigs after graph generation, removing contigs that do not meet the minimum coverage cutoff value as specified. This parameter removes short, low coverage contigs from the assembly, and in general improves the assembly when set to a reasonable value

between 4 and 10 (Figure 2.7b). Setting the value too high will remove highly covered and good quality contigs, while a too low value will include short, low covered contigs which most likely originated from nucleotide errors in the sequence, or contain low covered introns that were captured when unprocessed mRNA molecules were selected before sequencing.

Varying the parameters used during an assembly has a measurable effect on the total number of contigs, the average contig length and the number of bases present in a transcript assembly. The quality of a transcriptome assembly is, however, not based on the global properties of the assembly, but on the presence of near complete or completely assembled cDNA transcripts in the assembly. By using known, well studied, full-length cDNA sets of genes the corresponding transcripts in the assembly can be evaluated. Figures 2.8, 2.9 and 2.10 presents six *Eucalyptus grandis* cellulose synthase (CesA) genes (Ranik and Myburg, 2006), and the results of performing a BLAST (e^{-100}) of the CesA genes against assemblies from kmer 41 (Figure 2.8), kmer 51 (Figure 2.9) and kmer 61 (Figure 2.10) presented in Table 2.8. The CesA sequences (DQ014510.1, DQ014509.1, DQ014508.1, DQ014507.1, DQ014506.1 and DQ014505.1) are connected with colored banners of high similarity to regions present in contigs in the assembly dataset. Each CesA sequence can have similarity regions on multiple contigs present in the assembly. A perfect assembly will have a one-to-one ratio of CesA sequence to assembled contig with both sequences showing similarity along the entire length of the transcript. A subset of these CesA genes have been shown to have high expression in either primary or secondary cell formation tissues (Ranik and Myburg, 2006), and since these assemblies were performed with a single lane of xylem mRNA-Seq data, it can be expected that the lower abundant transcripts would not fully assemble. In order to select the best assembly parameters, a similar analysis should be repeated with different gene families that have a range of expression across multiple tissues.

2.3.4. Annotating assembled transcript sequences

Several good EST annotation pipelines exists in the public domain. These pipelines consists mainly of a set of scripts that calls a subset of tools sequentially to annotate a set of protein or DNA sequences. Few

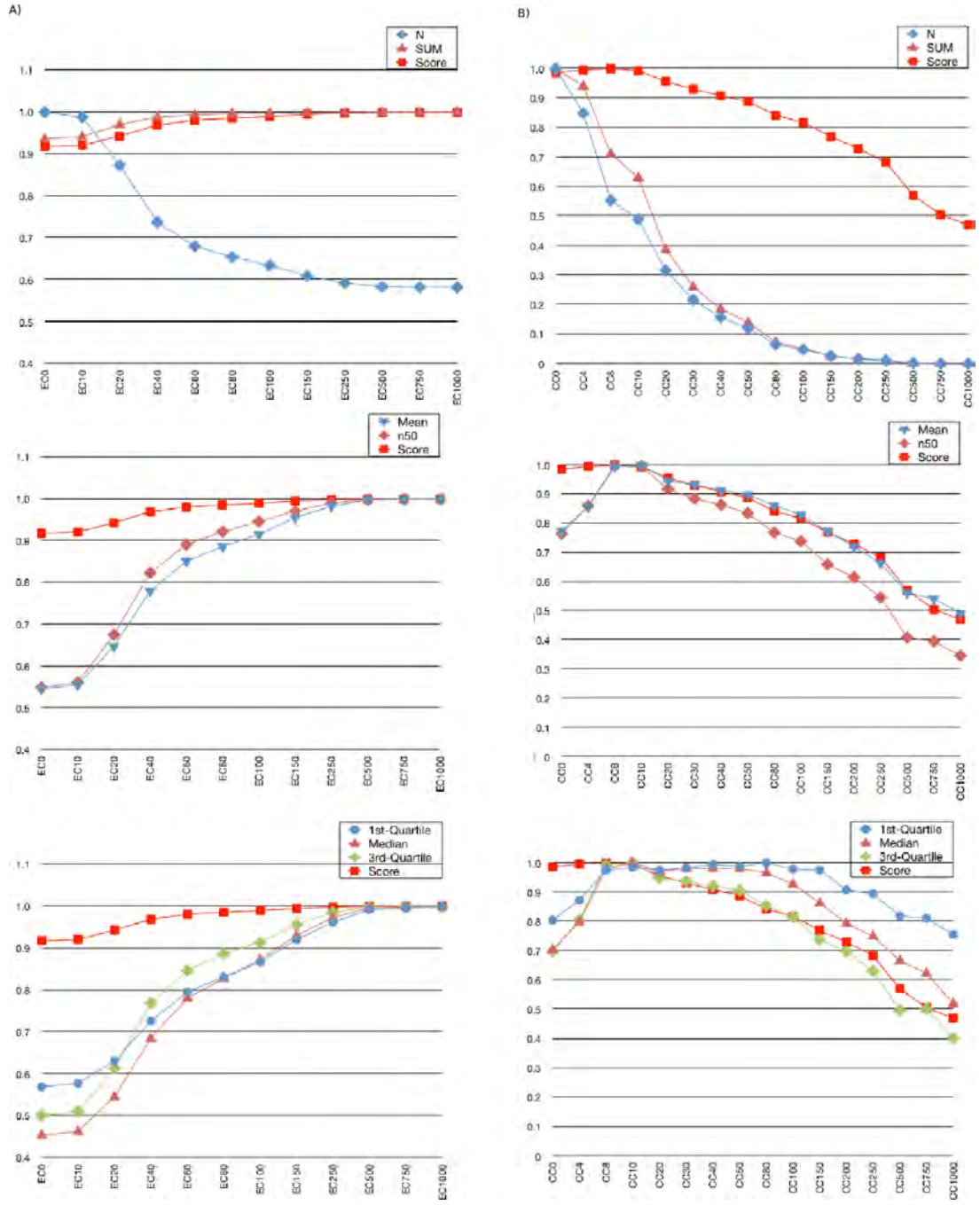


Figure 2.7: The effect of the expected coverage and the coverage cutoff parameters on a `Velvet` assembly. Due to the large dynamic range in transcript expression, high expected coverage values (A, left) produce the highest scoring assemblies. For the coverage cutoff parameter, it was found that the best `Velvet` assembly is achieved when the coverage cutoff parameter (B, right) ranges between 6 and 10. This will effectively remove low coverage contigs from the assembly while not removing the higher covered, longer contigs.

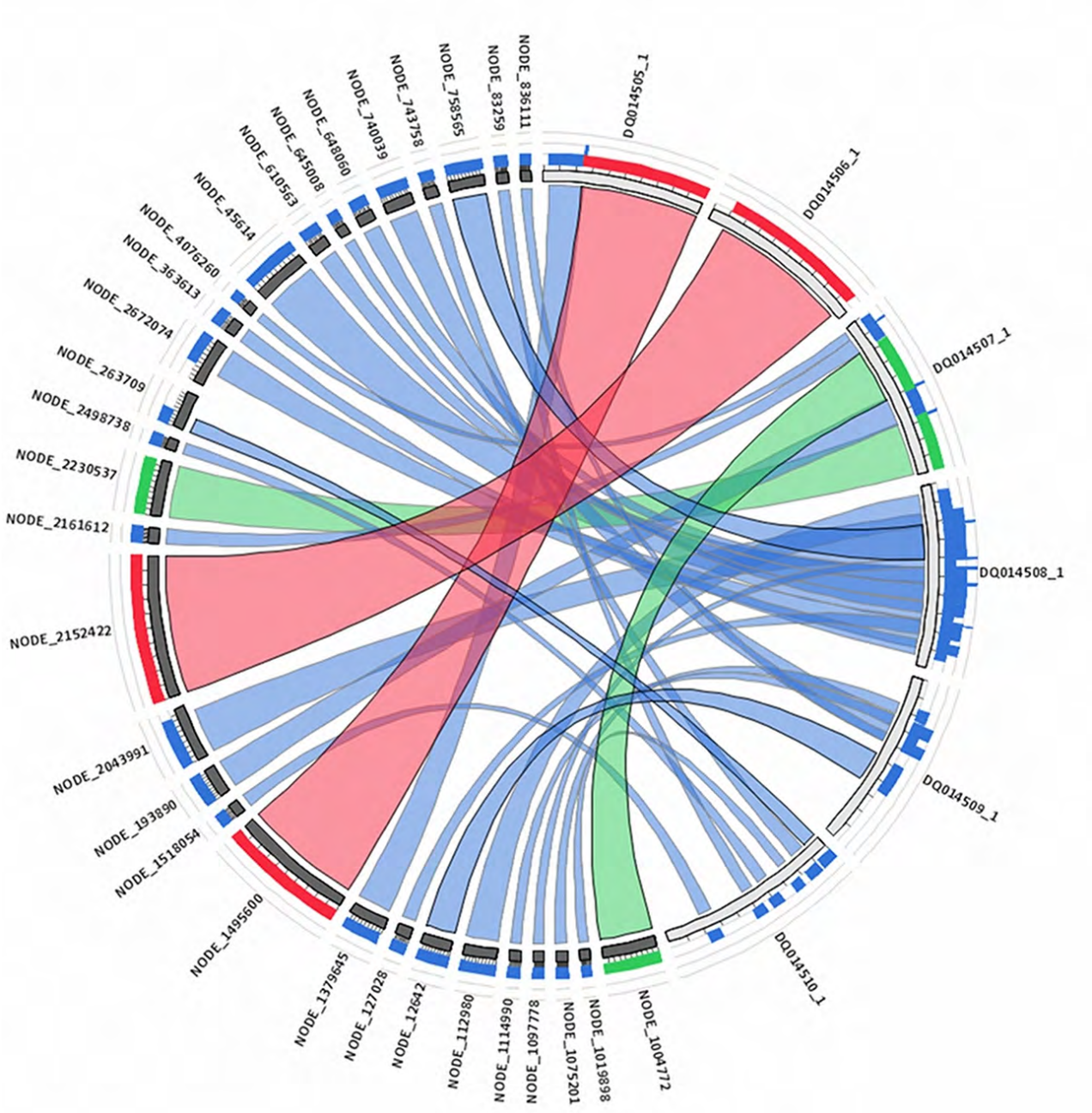


Figure 2.8: Alignment of the six full length CesA cDNA sequences against an assembly with a kmer size of 41 (k41). The CesA cDNA sequences (identifier starts with "DQ") show similarities to various contig sequences (identifiers "NODE") in the assembly. Blue ribbons indicate regions where the bit score of the alignment is $<$ 25% of the maximum bit score in the dataset. Warmer colors (25% $>$ green $<=$ 50%, 50% $>$ orange $<=$ 75% and red $>$ 75%) indicate higher bit scores. The two CesA cDNA sequences, DQ014506_a and DQ014505_1 are presented by near full length contigs NODE_2152422 and NODE_1495600. The cDNA sequence DQ014507_1 is represented by two large contigs (NODE_2230537 and NODE 1004772), while the remaining cDNA sequences are represented by various small contigs.

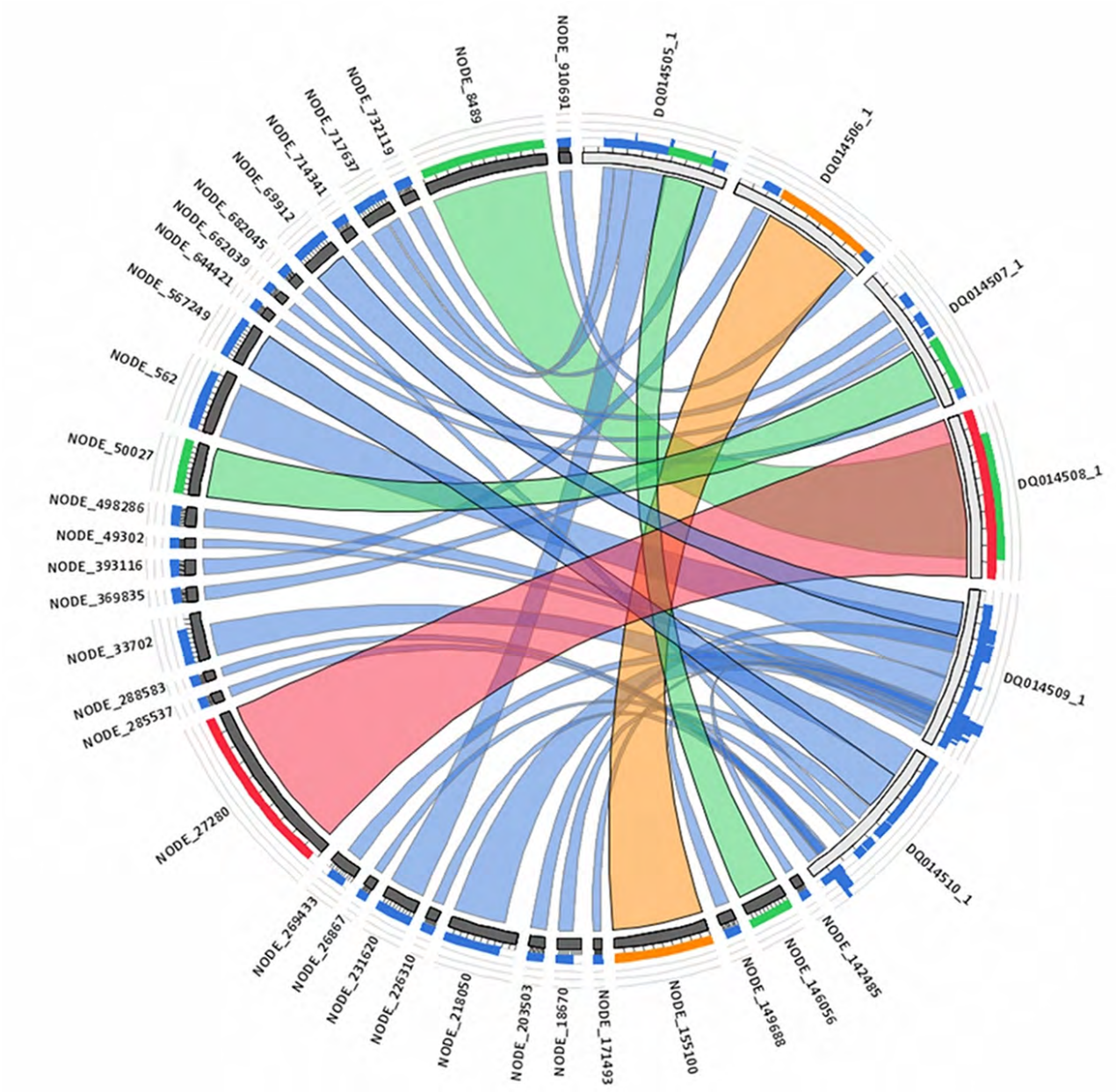


Figure 2.9: Alignment of the six full length CesA cDNA sequences against an assembly with a kmer size of 51 (k51). The CesA cDNA sequences (identifier starts with "DQ") show similarities (best BLAST hit) to various contig sequences (identifiers "NODE") in the assembly. Blue ribbons indicate regions where the bit score of the alignment is $< 25\%$ of the maximum bit score in the dataset. Warmer colors ($25\% >$ green $<= 50\%$, $50\% >$ orange $<= 75\%$ and red $> 75\%$) indicate higher bit scores. The alignment indicate two copies of the cDNA sequence DQ014501_1 in the assembly (NODE_27280 and NODE8489). A partially assembled contig (NODE_155100) that represent DQ014506_1 can also be identified. The remaining CesA's are represented by various shorter contigs in the dataset, indicating that there are still fragmented transcripts present in the assembly. The graph was generated with the Circoletto tool from the BLAST result file.

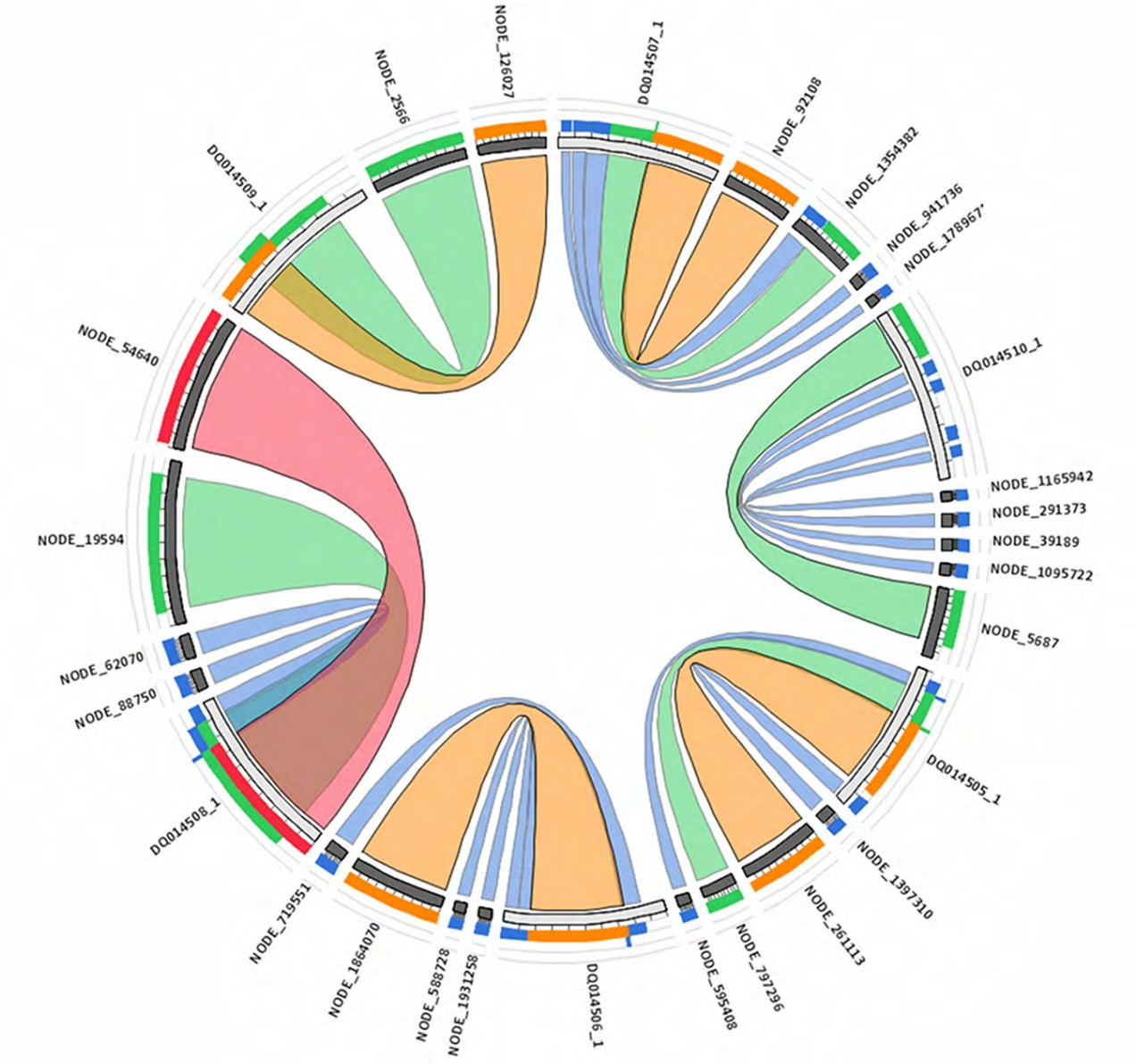


Figure 2.10: Alignment of the six full length CesA cDNA sequences against an assembly with a kmer size of 61 (k61). The CesA cDNA sequences (identifier starts with "DQ") show similarities (best BLAST hit) to various contig sequences (identifiers "NODE") in the assembly. Blue ribbons indicate regions where the bit score of the alignment is $<$ 25% of the maximum bit score in the dataset. Warmer colors (25% $>$ green $<=$ 50%, 50% $>$ orange $<=$ 75% and red $>$ 75%) indicate higher bit scores. The alignment represents the least fragmented assembly of the CesA cDNA sequences when compared to Figures 2.8 and 2.10. A duplicate assembled contig can be identified in the assembled dataset for sequence DQ014508_1. Most of the remaining CesA cDNA sequences are represented by at least one or two large contigs in the assembly, although not all of them aligning accross the whole length of the cDNA. The graph was generated with the Circoletto tool from the BLAST result file.

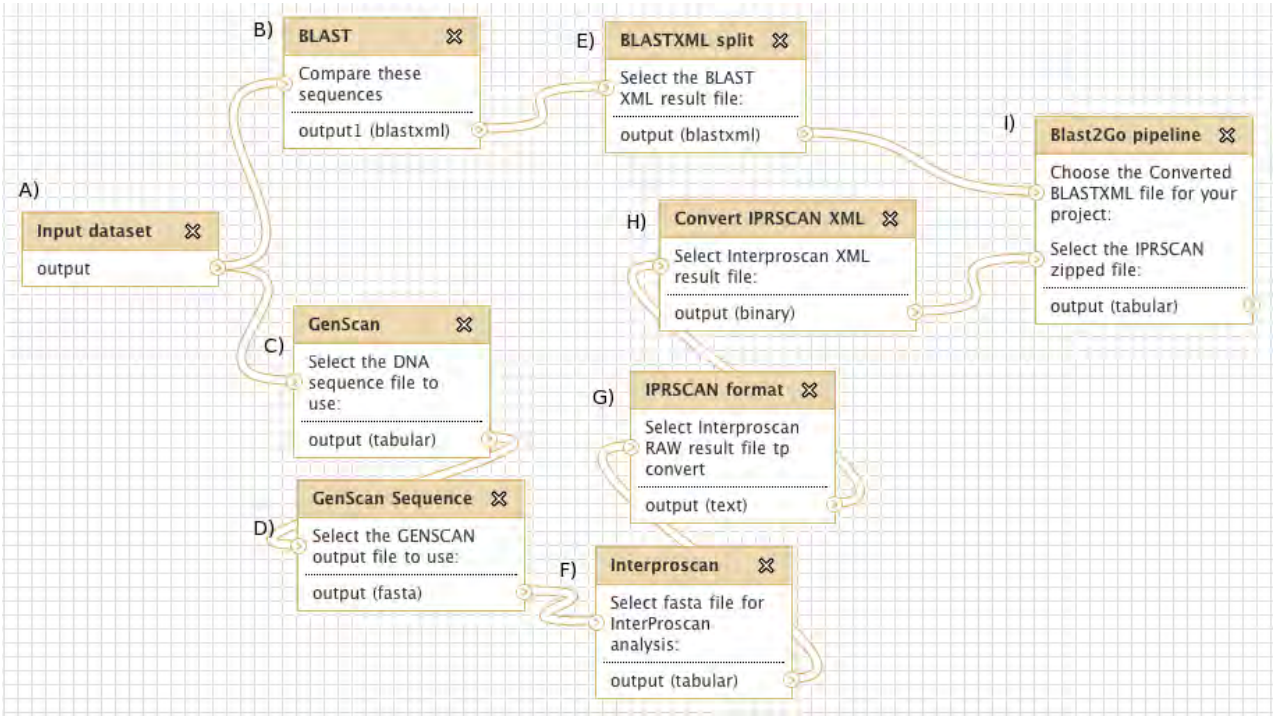


Figure 2.11: The automated annotation pipeline developed from tools available in `Galaxy`. The input for the pipeline (A) is a FASTA file containing cDNA sequence data. Protein sequence predictions are performed by `GenScan` (C and the results converted to `FASTA` format, D) and the resulting peptides submitted to the `IPRSCAN` pipeline. (F, G, and H) The input file is simultaneously submitted to BLAST (B and E) to perform homology searches (BLASTX), and the results of the `IPRSCAN` and BLAST searches used as input to the `BLAST2GO` pipeline (I) for further analysis.

pipelines allow the user to customise the different components used by the annotation pipeline specificaly for the organim that is to be annotated. Two widely used tools, the `InterProScan` set of scripts and databases (Zdobnov and Apweiler, 2001), and the `BLAST2GO` annotation pipeline Conesa *et al.* (2005) were incorporated in the BCBU `Galaxy` server. The InterProScan annotation scripts and associated databases are often used to unknown protein sequences with protein feature, protein family and detected motifs present on the protein sequence. The `BLAST2GO` pipeline assigns functional annotations to the submitted cDNA or protein dataset, which consists of Gene Ontology, `KEGG` and `InterPro` accessions. An automated workflow (Figure 2.11, available as the "`Annotation pipeline`" workflow in the BCBU `Galaxy` server) was developed to use both these annotation pipelines to annotate a set of cDNA sequences from the transcriptome assembly pipeline described above.

The automated assembly workflow takes cDNA sequences as input (ESTs or contigs assembled from

mRNA-Seq data), performs a translation of the coding sequence into a putative protein and CDS sequence, and uses the predicted protein sequence to find protein family and protein feature annotations with `IPRSCAN`, the interface to the EBI's `InterProScan` tool. Results from `IPRSCAN` analysis are then converted to a format acceptable for the `BLAST2GO` annotation tool. The protein sequences are also used for a homology-based search against an external database (for instance, the NCBI's database of non redundant protein sequences), and the results parsed for use in the `BLAST2GO` annotation pipeline. `BLAST2GO` analysis is performed with the homology search (`BLAST`) and the `IPRSCAN` results as input, and an annotation (`.annot`) file is constructed. This `.annot` file can then be used as direct input into a `BLAST2GO` instance for the perusal of the annotations or imported into an external database.

The input sequences to the pipeline can consist of portions of genomic cDNA, full-length CDS or partial CDS sequences. The gene finder application, `GenScan` (Burge and Karlin, 1997) was used to predict a protein and CDS sequence from the input sequence. This is a very crude approach to cDNA translation and peptide sequence prediction, since partially assembled sequences will not have all the sequence signals present on the sequence required by `GenScan` to perform a reliable prediction of the exact intron and exon structure of the input sequence. This particular tool can, however, be replaced by any other gene prediction or cDNA translation tool in the workflow, as long as a protein sequence is the output from the alternative tool. The pipeline was used to perform a basic annotation of the 18 894 full-length, or partially assembled sequences of a *Eucalyptus grandis* x *Eucalyptus urophylla* transcriptome generated from mRNA-Seq data (Chapter 3).

The `InterProScan` analysis tool scans a given protein sequence against a range of protein signatures stored in the InterPro member databases. These signatures, present in the `PROSITE`, `PRINTS`, `Pfam`, `ProDOM` and `SMART` databases can then be used to provide functional annotations of the input protein sequence based on motifs present in the sequence. The `InterProScan` tool is a scalable and extensible system for protein feature annotation, and searches databases installed on a local server of the mentioned sources in order to find signature sequences. Results from the `InterProScan` analysis tool can be converted into `XML`, `HTML` or a `TXT` based file, which can be used to create a summary of the features

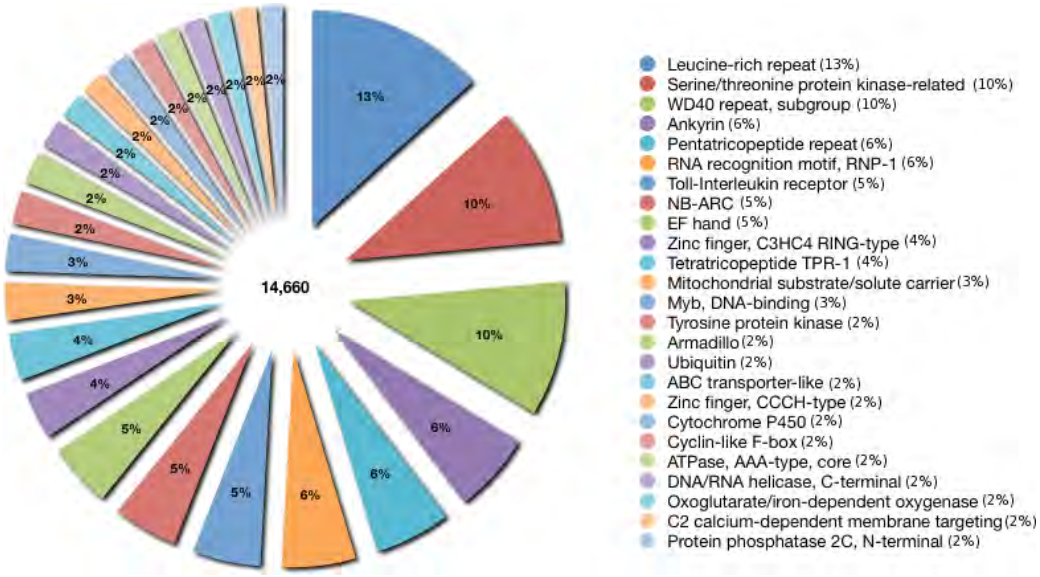


Figure 2.12: The 25 most prevalent protein family domains annotated in an assembled transcriptome dataset, expressed as a fraction of the total number of PFam annotations. The Leucine Rich Repeat (PF:PF00560) region was the annotation assigned in 13% of the annotations, and the dataset also represents annotations of kinases (PF00069 and PF07714) and the Myb transcription factor binding domains (PF00249). The figure was produced from the PFAM annotations assigned to the 18 894 assembled contigs by the `InterProScan` tool.

found in the dataset on a global scale (Figure 2.12), or to view the signatures and features annotated on a specific sequence (Figure 2.13).

Various functional annotation projects use the Gene Ontology system to group sequences into related functional groups. The `BLAST2GO` annotation tool offers a wide range of statistical validations in assigning a functional classification to a protein sequence. The results from the annotation workflow produce an annotation file, generated by the command line interface (`b2gPipe`) of the `BLAST2GO` annotation tool. The pipeline expects `BLAST` XML results formatted in a specific manner, and a directory containing `InterProScan` XML results in order to complete the annotation. The `BLASTXML2BLAST2GO` and `IPRSCANXML2BLAST2GO` `Galaxy` extensions perform the simple conversions between the formats, and also execute the `b2gPipe` pipeline. `BLAST2GO` relies on a local installation of public Gene Ontology and Gene Ontology Accession databases to assign the Gene Ontology annotations to the sequences in the `BLAST` XML file. The annotation file produced can then be imported in a stand-alone version of the

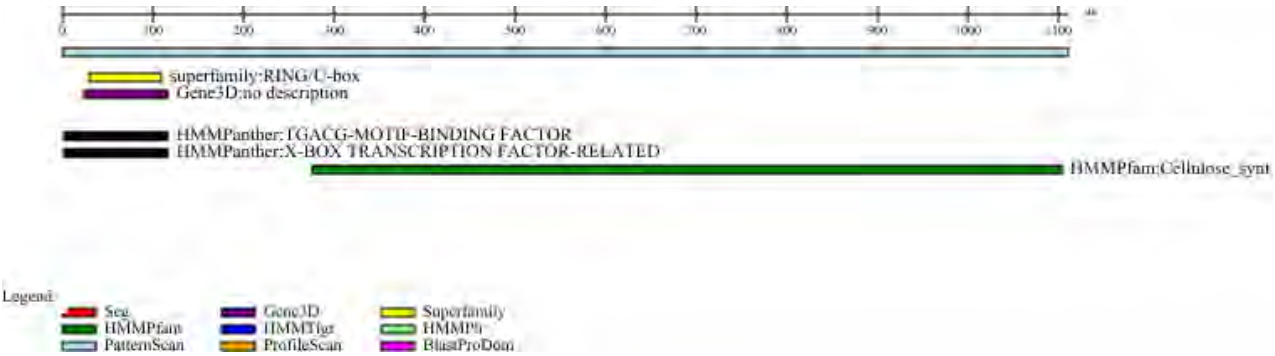


Figure 2.13: Protein features annotated by InterProScan present on the cellulose synthase 6 (CesA6) protein sequence assembled from reads derived from mRNA-Seq sequencing. The sequence represents the assembled contig with the highest homology to the CesA6 (DQ014510.1) mRNA sequence, and was annotated by the InterProScan annotation pipeline. The annotation indicates the presence of a transcription factor binding motif (TGACC-motif, black box), a X-Box transcription factor-related motif (black box) on the 5' end of the sequence identified by HMMPanther. The same 5' region has also been identified as having a Ring/U-box superfamily signature (yellow box). The long green box represents the presence of the cellulose synthase protein family signature identified by HMMPfam. The image was generated from the RAW results by the `InterProImageGenerator` tool in `Galaxy`.

`BLAST2GO` tool, and can be used to summarise the overall ontology structure of the dataset, as well as inspect the annotations made to a single protein sequence.

2.3.5. Using mRNA-Seq data to calculate transcript expressions values

Many research groups have calculated gene transcript abundance levels with the aid of mRNA-Seq data (see Section 1.3 for a review of RPKM and FPKM calculations and other high-throughput sequencing applications in genetics and genomics). Mortazavi *et al.* (2008) showed that the differences in transcript abundance can span five orders of magnitude, and that the mRNA-seq methodology used was shown to be sensitive enough to detect even single copies of a transcript in a cell . A recent methods paper used mRNA-Seq data to detect novel transcripts and alternative spliceforms of transcripts, and was made available as the `CUFFLINKS` package (Trapnell *et al.*, 2010). `CUFFLINKS` performs a *de novo* prediction of splice junctions, and generates a set of detected gene models with their corresponding expression values (FPKM). The following section describes the workflow developed to detect transcript expression values for an organism where no annotated gene information is available (Figure 2.14, available as the "`FPKM calculation`" workflow in the BCBU server). The workflow starts of by mapping an input

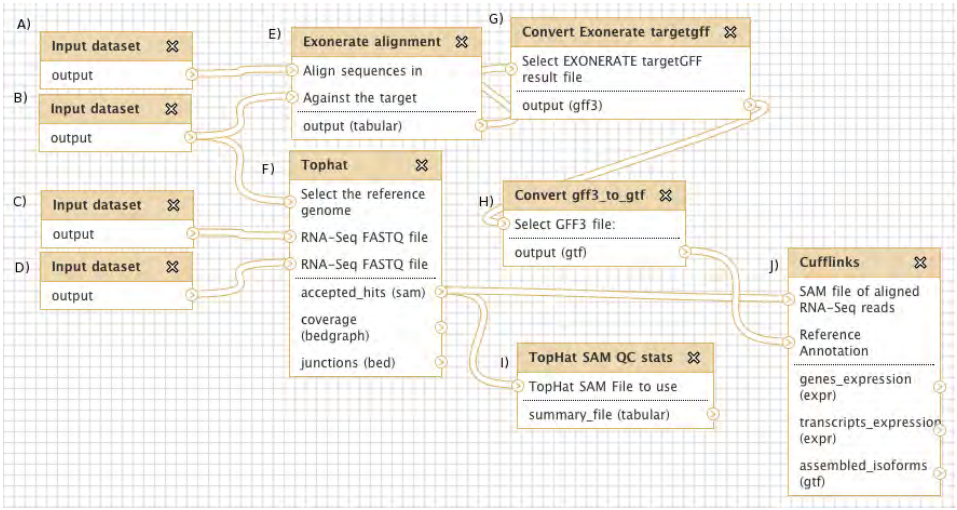


Figure 2.14: Calculating gene expression (FPKM) values for unigene aligned regions from a genome with no gene models available. The input dataset for the workflow is a reference genome (B), the forward and reverse reads of an mRNA-Seq lane (C and D), and a FASTA file containing a set of ESTs (A). TopHat aligns the mRNA-Seq reads to the genome (F) and also against the splice junction regions using the Bowtie aligner. The alignment file (SAM format) is then used in calculating some short read mapping statistics (I), and as input for `CUFFLINKS` (J). The unigenes input dataset is aligned against the genome with `EXONERATE` (E), and the GFF output of `EXONERATE` is converted to the required GTF format (H) for `CUFFLINKS`. The GTF and SAM files are used to calculate the FPKM values (J).

set of mRNA-Seq reads to a target genome with `TopHat` (Trapnell *et al.*, 2009), as well as aligning a set of cDNA sequences to the genome with the `EXONERATE` aligner (Slater and Birney, 2005). The workflow further generates a gene model file from the cDNA alignment, and calculates the FPKM values for each of the transcripts present in the alignment.

The normalised transcript expression values (FPKM) are calculated by mapping reads to a target genome, constructing splice sites where reads span intron junctions, and then calculating the number of fragments that map per unit transcript. The `TopHat` mapping program (Trapnell *et al.*, 2009) was designed to determine the splice junction alignment when mapping to genome sequences. A single lane of 76 bp Illumina mRNA-Seq data was trimmed to shorter lengths and mapped to the *Eucalyptus grandis* draft genome sequence. Since longer reads require more reagents during sequencing, a key question to address is how a difference in read length influences the read mapability. Figure 2.15 indicates that there is an increase in the number of paired reads that map uniquely to a genome when the read length is increased from 40 bp to 50 bp, but beyond 50 bp there is not a marked difference in the number of paired

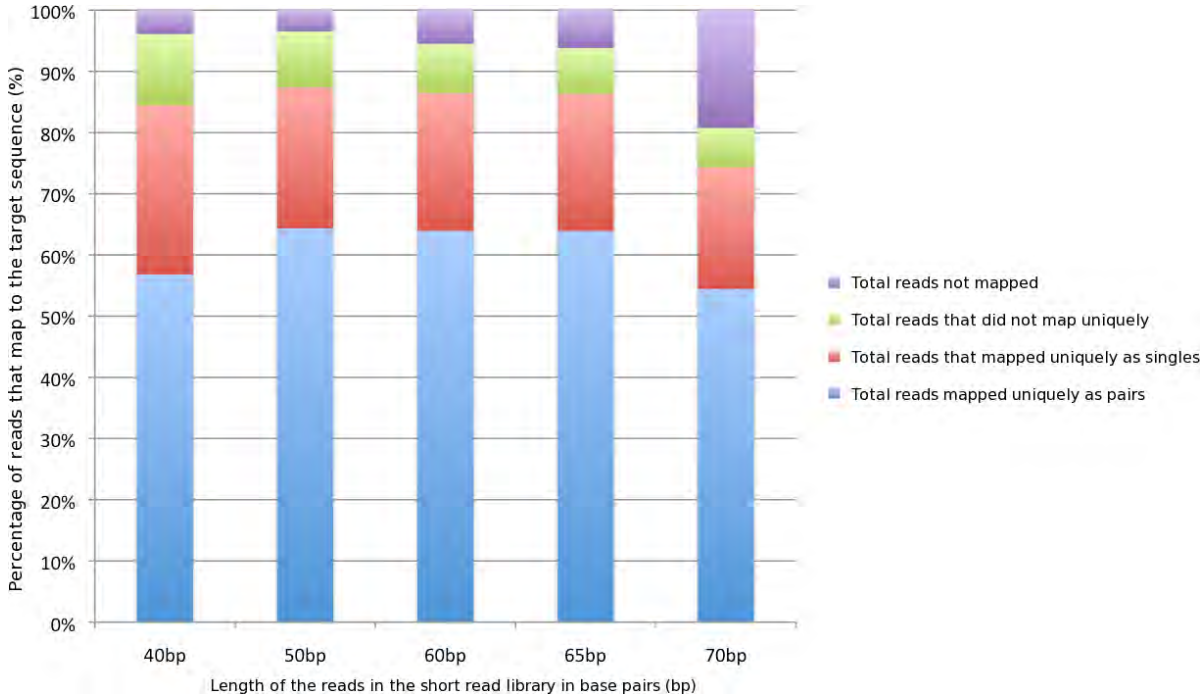


Figure 2.15: A breakdown of the number of reads which map uniquely, and non-uniquely as pairs or single reads to a target genome for different read lengths. No increase in read specificity can be detected when paired reads are longer than 50 bp in terms of unique paired mapping to the genome. Up to 97% (50-65 bp) of the reads were mappable to the genome, but this includes reads that map to regions outside gene models and within repeat regions. There is a significant increase in the number of reads that did not map to the genome when the read length was 70 bp.

reads that map to the genome. These results indicate that a paired read or fragment of 50 bp has a high enough specificity in the genome to map uniquely, and longer reads are not necessarily more specific. Reads longer than 70 bp shows a decrease in mappability, due to the stringency associated with the number of mismatches allowed when aligning a read to the target sequence. These mismatches have a higher probability to occur in longer reads, mostly due to the effect of sequencing errors in longer reads, but also due to SNPs present in a sequenced sample.

`CUFFLINKS` makes use of the genomic coordinates of genes or transcripts to calculate the FPKM expression value. The coordinates file needs to be supplied in the GTF (a condensed GFF3 file format) format to `CUFFLINKS`. The genome coordinates for a genome where no annotation, i.e. no GFF3 file exists, can be determined by performing a gapped alignment of cDNA sequences to the genome with `EXONERATE`. Output from `EXONERATE` needs to be reformatted to the GFF3 format and converted to the

GTF format before serving as input to `CUFFLINKS`. `CUFFLINKS` can calculate the FPKM values for the annotated genes present in the GTF file, or if no reference gene models are provided, it will identify new expressed transcripts.

Lists of genes and their expression values can serve as input to one of several statistical packages to determine groups of genes that are differentially expressed between experiments. The R package `DEGseq` (Wang *et al.*, 2010*a*) was used to determine a list of genes differentially expressed between immature xylem and young leaf tissue of a *Eucalyptus grandis* hybrid tree (Chapter 3). Figure 2.16 present the results from the DEGseq package used to determine differential expression. The figure presents the MA plot (where $M = log_2 tissue_1 - log_2 tissue_2$, $A = 1/2(log_2 tissue_1 + log_2 tissue_2)$) of differential expressed genes identified with a 2X fold change method to detect differential expression. The Venn diagrams below the MA plot shows the number of genes detected to be differentially expressed in immature xylem and in young leaf tissue, and the set of genes not being differentially expressed.

2.4. Conclusion

The management and data analysis of large DNA sequence datasets produced with high throughput biological experiments require sound data management principles, dedicated and sometimes specialised computational hardware, and a variety of software tools. The `Galaxy` framework was identified as one of many potential data management and automated data analysis workflow systems that can be used and adapted to analyse mRNA-Seq datasets. The framework can easily be extended to include new analysis tools, which can then be incorporated into complex workflows, which have the ability to make high throughput data analysis tools available to research groups. The framework effectively reduces the steep learning curve needed to master the command line interface of an analysis tool, by providing a web-based form to set the parameters used during the execution of the analysis program.

The quality evaluation of uHTS data is one of the first analysis steps when working with theses datasets. The current Illumina pipeline (version 3.6) produced quality scores associated with each base of sequence in an format that differs from the standard `Phred` based format, which needs to be converted

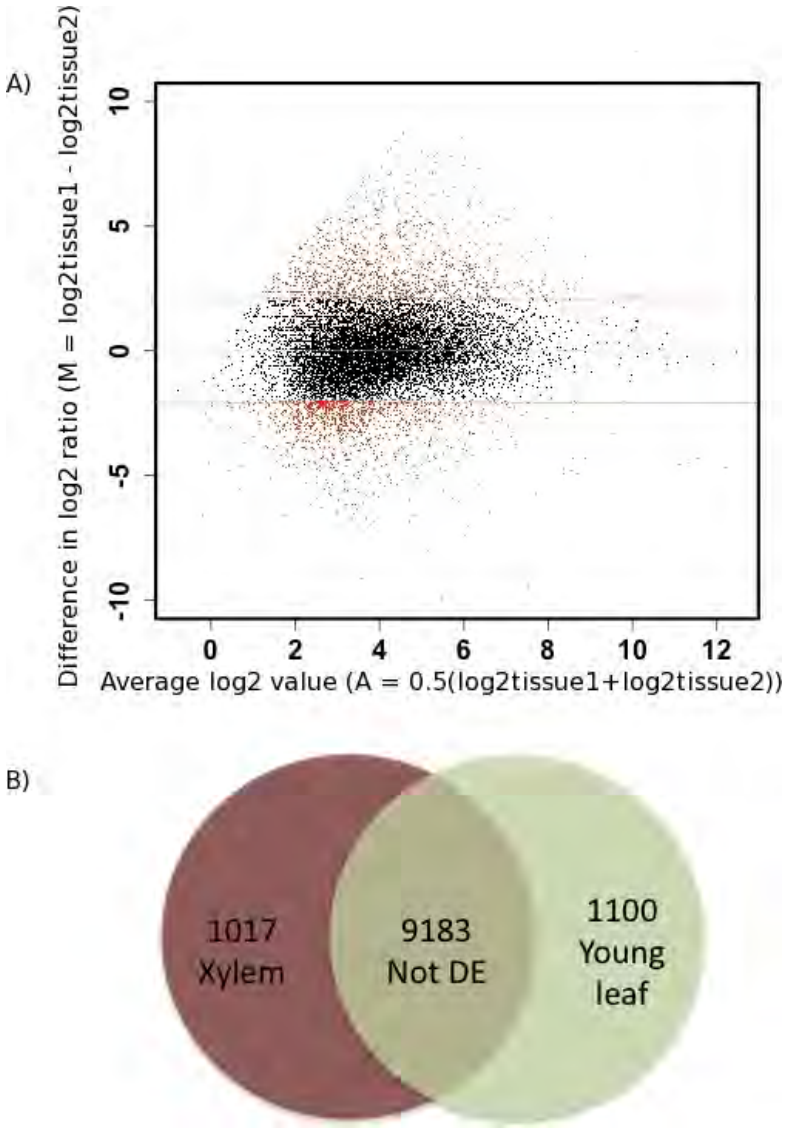


Figure 2.16: Genes identified as differentially expressed in immature xylem and young leaf tissues of a *Eucalyptus grandis* hybrid tree. The top figure (A) represent genes identified by the DEGseq tool as differentially expressed genes based on the MA (where where $M = log_2tissue_1 - log_2tissue_2$, $A = {}^1/_2(log_2tissue_1 + log_2tissue_2)$) using a 2X fold change method. The Venn diagrams representthe same set of genes identified as being differentially expressed in immature xylem (brown) and young leaf tissue (green), and the genes that are not detected as being deferentially expressed (Not DE, overlapping area).

to the standard `Phred` format. After conversion, a per base quality graph can be calculated for every base at every position of the read, and bases removed from the 3' ends of the reads. Depending on the amount of data available, it is recommended that a `Phred` quality value of 20 (base error rate of 1 in 100) is used as a guideline to trim the reads. Erroneous bases at the 3' ends of the reads have the ability to prohibit the alignment of a read to a target sequence as it increases the number of mismatches that will occur between a target sequence and the read, and also with graph-based assemblers it can create low coverage paths between the nodes of the graph. A default pipeline for the quality evaluation of short read Illumina data is available as the "Illumina QC" workflow in the BCBU `Galaxy` server installed at the University of Pretoria.

The assembly of a set of representative cDNA sequences from a pool of mRNA reads is still a challenging endeavor. A workflow which makes uses of the `Velvet` assembler to assemble contigs was developed to assist in performing multiple assemblies and keep track of the results. The workflow re-formats the input datasets to the format required for Velvet, performs the assembly of the input datasets, and produces a basic statistics file summarising the assembly. *De Bruijn* assemblers have a very high memory footprint, and hardware with the required RAM is required to succesfully complete the assembly. A dataset containing 35 million short reads of (35-50 bp in length) typically requires up to 120 GB of RAM, depending on the size of kmer used during assembly. A recent thread on the SeqAnswers forums[4] stated that the following formula can be used to calculate the amount of RAM needed for a genome assembly: $RAM = -109635 + 18977 * ReadLength + 86326 * GenomeSize + 233353 * NumberOfReads - 51092 * kmer$. No such formula exists to calculate the amount of RAM needed for a transcriptome assembly, mainly due to the uncertainties of transcriptome size, and number of alternative isoforms that can be present in a sample. For a typical Illumina dataset consisting of reads 76 bp long, a kmer value between 51 and 55 were found to produce the best assembly using a scoring function that takes into account the number of bases as well as the number and length of contigs present in an assembly. The choice of kmer, expected coverage and coverage cutoff depends greatly on the size and characteristics of the biological sample, as well as the amount and quality of sequence data used for

[4] http://seqanswers.com/forums/showthread.php?t=2101

the assembly, and therefore no conclusion can be reached in terms of the best parameters to use. One important aspect when evaluating the contiguity of the assembled transcripts is the comparison against known, full-length cDNA sequences in order to identify missassembled contigs and critically evaluate an assembly.

The availability of transcriptome specific assembly software, such as `trans-ABySS` (Robertson *et al.*, 2010), `OASES` (Zerbino et al., unpublished) and the recently released `Trinity` (Grabherr *et al.*, 2011) software packages will in future make *de novo* assemblies of full-length transcripts a standard bioinformatic operation. The `Velvet`-based assembler approach described here does not deal with the assembly of alternative splice forms, and may assemble some partial transcripts, but the analysis described did result in the assembly of near full-length, contiguous biological molecules, as described in Chapter 3.

Functional annotation of a set of assembled transcripts occurs mainly through homology-based searches to identify sequences similar to a newly sequenced organism. Both the `InterProScan` and `BLAST2GO` pipelines makes use of homology-based searches and functional protein domain signatures to assign functional annotation to a contig. These annotation pipelines have been used with great success to functionally annotate a vast range of EST and cDNA datasets (Vizoso *et al.*, 2009; Coetzer *et al.*, 2010; Arnaiz *et al.*, 2010; Blanca *et al.*, 2011; Mondego *et al.*, 2011), The `InterProScan` pipeline assigns `PROSITE`, `PRINTS`, `Pfam`, `ProDOM` and `SMART` annotations to each contig in the cDNA file, with the `BLAST2GO` pipeline makes use of these protein features to assign Gene Ontology, `KEGG` and `InterPro` categories to the contigs. The results from the pipeline is presented in a format that can be viewed by the `BLAST2GO` application, or parsed to a delimited text file that can be imported to a database system.

Gene expression calculated with mRNA-Seq data is reported to be more robust than microarray data (Li *et al.*, 2008*a*; Marioni *et al.*, 2008; Hiller *et al.*, 2009). Estimating gene expression values from known and novel genome models and transcripts aids in identifying pathways and functional gene classes that are over-expressed between different tissues or conditions. Functional expression analysis of different tissues and/or different stages of development can be viewed as the first steps to a complete functional characterisation of a species of interest. The first step in estimating gene expression is to re-align or

map the Illummina short-read data to the target genome and a set of splice juctions. Results show that for the *Eucalyptus grandis* genome, paired end reads longer than 50 bp do not increase the mapability of the fragments, when reads were aligned with the `TopHat` program (Trapnell *et al.*, 2009). This value will differ between different organisms, but can be used as a guideline to determine gene expression for organisms of similair genome complexity as eucalypts. Several statistical approaches have been developed to model the distribution of RNA-Seq data across a transcriptome (Langmead *et al.*, 2010; Srivastava and Chen, 2010; Trapnell *et al.*, 2010; Wang *et al.*, 2010*a*) and correct for transcript length (Oshlack and Wakefield, 2009), positional (Bohnert and Rätsch, 2010) and content bias of the technology (Hansen *et al.*, 2010). Improvements to the CUFFLINKS package to incorporate various normalisation methods for the detection of differential expression makes it a valueable benchmark to use for expression analysis (Trapnell *et al.*, 2010; Roberts *et al.*, 2011). The `DEGseq` package makes use of three different published methods (Marioni *et al.*, 2008; Bloom *et al.*, 2009; Tang *et al.*, 2009) and two novel methods to identify differential expression using mRNA-Seq data, and also serves as a good alternative starting point for different expression analysis. Both `CUFFLINKS` and `DEGseq` are available as tools in the BCBU `Galaxy` server. Investigations of transcriptome wide gene expression data assist in the selection of target genes of interest for genetic modification and the elucidation of complex traits when combined with population genetic data.

The workflows described here serve as a starting point to a whole range of uHTS DNA sequence analyses. The Galaxy environment facilitates easy incorporation of new tools, results storage and tracking, and a common interface to store and share analysis pipelines and results. Key parameters that can influence the output of the individual analysis tools that make up the workflows have been discussed and guidelines provided regarding the effect of these parameters on a dataset. The guidelines provided should, however, be used with caution, as they are only applicable to the datasets and organism evaluated. The workflows described here have been used to perform the *de novo* assembly of a gene catalog from mRNA-Seq, the subsequent annotation of the assembled gene catalog as well as the expression profiling of the assembled transcripts as described in Chapter 3.