5-2022

Comparative Analyses of De Novo Transcriptome Assembly Pipelines for Diploid Wheat

Natasha Pavlovikj
University of Nebraska-Lincoln, natasha.pavlovikj@huskers.unl.edu

COMPARATIVE ANALYSES OF DE NOVO TRANSCRIPTOME ASSEMBLY PIPELINES FOR DIPLOID WHEAT

by

Natasha Pavlovikj

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Jitender S. Deogun

Lincoln, Nebraska

May, 2022

COMPARATIVE ANALYSIS OF DE NOVO TRANSCRIPTOME ASSEMBLY PIPELINES FOR DIPLOID WHEAT

Natasha Pavlovikj, M.S.

University of Nebraska, 2022

Advisor: Jitender S. Deogun

Gene expression and transcriptome analysis are currently one of the main focuses of research for a great number of scientists. However, the assembly of raw sequence data to obtain a draft transcriptome of an organism is a complex multi-stage process usually composed of pre-processing, assembling, and post-processing. Each of these stages includes multiple steps such as data cleaning, error correction and assembly validation. Different combinations of steps, as well as different computational methods for the same step, generate transcriptome assemblies with different accuracy. Thus, using a combination that generates more accurate assemblies is crucial for any novel biological discoveries. Implementing accurate transcriptome assembly requires a great knowledge of different algorithms, bioinformatics tools and software that can be used in an analysis pipeline. Many pipelines can be represented as automated scalable scientific workflows that can be run simultaneously on powerful distributed and computational resources, such as Campus Clusters, Grids, and Clouds, and speed-up the analyses.

In this thesis, we 1) compared and optimized de novo transcriptome assembly pipelines for diploid wheat; 2) investigated the impact of a few key parameters for generating accurate transcriptome assemblies, such as digital normalization and error correction

methods, de novo assemblers and *k*-mer length strategies; 3) built distributed and scalable scientific workflow for *blast2cap3*, a step from the transcriptome assembly pipeline for protein-guided assembly, using the Pegasus Workflow Management System (WMS); and 4) deployed and examined the scientific workflow for *blast2cap3* on two different computational platforms.

Based on the analysis performed in this thesis, we conclude that the best transcriptome assembly is produced when the error correction method is used with Velvet Oases and the "*multi-k*" strategy. Moreover, the performed experiments show that the Pegasus WMS implementation of *blast2cap3* reduces the running time for more than 95% compared to its current serial implementation. The results presented in this thesis provide valuable insight for designing good de novo transcriptome assembly pipeline and show the importance of using scientific workflows for executing computationally demanding pipelines.

Table of Contents

List of Tables

List of Figures

PREFACE

The results presented in Chapter 2 have been published in:

- Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. Analysis of transcriptome assembly pipelines for wheat. In 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) 2016 Dec 15 (pp. 137-140). IEEE [34];
- Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. Comparing and optimizing transcriptome assembly pipeline for diploid wheat. In Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics 2014 Sep 20 (pp. 603-604) [35];
- Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. Evaluating assembly pipeline for transcriptomes. In 6th International Conference on Bioinformatics and Computational Biology, BICOB 2014 2014 (pp. 163-168). International Society for Computers and Their Applications [36].

The results presented in Chapter 3 have been published in:

- Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. Evaluating distributed platforms for protein-guided scientific workflow. In Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment 2014 Jul 13 (pp. 1-8) [37];

- Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. A comparison of a campus cluster and open science grid platforms for protein-guided assembly using pegasus workflow management system. In 2014 IEEE International Parallel & Distributed Processing Symposium Workshops 2014 May 19 (pp. 546-555). IEEE [38].

Chapter 1

Introduction

1.1. Motivation

Transcriptome assembly is the process of reconstructing the transcriptome of an organism using millions of short, sequenced RNA-Seq data. These short sequences, commonly produced by Illumina systems, are assembled into contigs (transcripts). RNA-Seq has established as a powerful technique to understand the molecular mechanisms of organisms, identify expressed genes, as well as address various biological questions [40]. While RNA-Seq provides meaningful information, it poses various bioinformatics challenges. The two main challenges are: 1) conducting accurate analysis to extract biologically relevant information; and 2) data handling (storage and processing).

The analysis of RNA-Seq data is composed of multiple stages, pre-processing, assembly and post-processing, and each stage is composed of multiple steps (e.g., trimming, quality check, assembly, annotation). All these steps are executed as part of an assembly analysis pipeline. For each of these steps, multiple efficient computational methods and algorithms exist. However, even though these methods tackle the same problem, different results may be obtained. Some reasons for this may be different algorithmic implementations, different parameter settings available, different species, different sequencing protocols, as well as different datasets used [66]. All of this poses the question of what tools and parameter settings can be used to produce good and

accurate assembly. While new tools and methods are frequently developed, currently there is no optimal assembly pipeline for analyzing all RNA-Seq data [66]. Merging the contigs from different assembly tools and different *k*-mer lengths seems to be the best way to obtain a comprehensive de novo transcriptome assembly [65][67]. The choice of tool for a specific step can significantly affect the downstream analysis and the biological discovery. Thus, having more comparative analyses of transcriptome assembly pipelines and evaluating the advantages and disadvantages of each tool is crucial for building accurate and good de novo transcriptome assemblies.

Due to the affordable sequencing technologies, RNA-Seq datasets are large and require significant data storage and computing time for its analysis. Not many biological labs that generate the data have large storage systems, computing resources, or computing skills for analyzing the data [68]. Some analyses can take anywhere from a few days to several months, and some analyses need to be conducted on the entire datasets, while some analyses can be modular and independent and performed only on a subset. Researchers mostly want to analyze the RNA-Seq data in a quick, easy, automated, and scalable manner on various computational platforms. Thus, using scientific automated workflows and parallelizing strategies across a single or multiple computational platforms is crucial for an efficient execution of these analyses. Some scientific workflows can be spilt into multiple sub-workflows that can be executed in parallel on powerful computational and distributed resources. Each workflow is composed of multiple computational tasks with different execution order. In the recent years, many Workflow Management Systems

(WMS) have been developed to support researchers build scientific workflows for efficient data processing and significant acceleration of their analyses [69].

1.2. Thesis contributions

In this thesis, we address the two main bioinformatics challenges when analyzing RNA-Seq data: 1) conducting accurate analysis to extract biologically relevant information; and 2) data handling (storage and processing). To tackle the challenge for conducting accurate analysis, we first designed and compared multiple de novo transcriptome assembly pipelines. Next, we investigated the impact of a few key methods for generating accurate transcriptome assemblies, such as digital normalization and error correction methods, de novo assemblers, as well as various *k*-mer length strategies. Based on our experiments, we propose a set of suggestions for choosing a good strategy for optimizing de novo transcriptome pipelines. Secondly, to approach the data handling and processing challenge, we converted the serial implementation of *blast2cap3*, a step from the transcriptome assembly pipeline for protein-guided assembly into a distributed and scalable scientific workflow using the Pegasus Workflow Management System (WMS). Next, we deployed and examined the scientific workflow for *blast2cap3* on two different computational platforms, a Campus Cluster and distributed Grid. The conducted experiments show that the Pegasus WMS implementation of *blast2cap3* significantly reduces the running time and show the importance of using scientific workflows for executing computationally demanding pipelines.

1.3. Thesis outline

In Chapter 2, Analysis of Transcriptome Assembly Pipelines for Wheat, we developed and compared 21 de novo transcriptome assembly pipelines for diploid wheat and investigated the impact of a few key parameters for generating accurate transcriptome assemblies, such as digital normalization and error correction methods, de novo assemblers and *k*-mer length strategies, on the overall assembly accuracy.

In Chapter 3, Evaluating Distributed Platforms for Protein-Guided Scientific Workflow, we built distributed and scalable scientific workflow for *blast2cap3*, a step from the transcriptome assembly pipeline for protein-guided assembly, using the Pegasus Workflow Management System (WMS). Next, we deployed and evaluated the scientific workflow for *blast2cap3* on two different computational platforms, a Campus Cluster and distributed Grid.

Finally, in Chapter 4 we present the conclusions of this thesis.

Chapter 2

Analysis of Transcriptome Assembly Pipelines for Diploid Wheat

2.1. Introduction

With the recent advance of sequencing technologies, transcriptome sequencing (RNA-Seq) has emerged as a powerful tool for obtaining large amount of functional genomic data in both model and non-model organisms [39]. The assembly of raw sequence data to obtain a draft transcriptome of an organism is a complex multi-stage process usually composed of pre-processing, assembly and post-processing. Each of these stages includes multiple steps such as data cleaning, contaminant removal, error correction, de novo or reference-based assembly, redundancy removal, and assembly validation. In order to implement all these steps, a great knowledge of different algorithms, various bioinformatics tools and software is required [40]. The assembly pipeline is used to simplify the entire assembly process by automating various steps of the pipeline for producing correct transcripts. A general transcriptome assembly pipeline with some common steps and the tools used for those steps is shown on Figure 1.

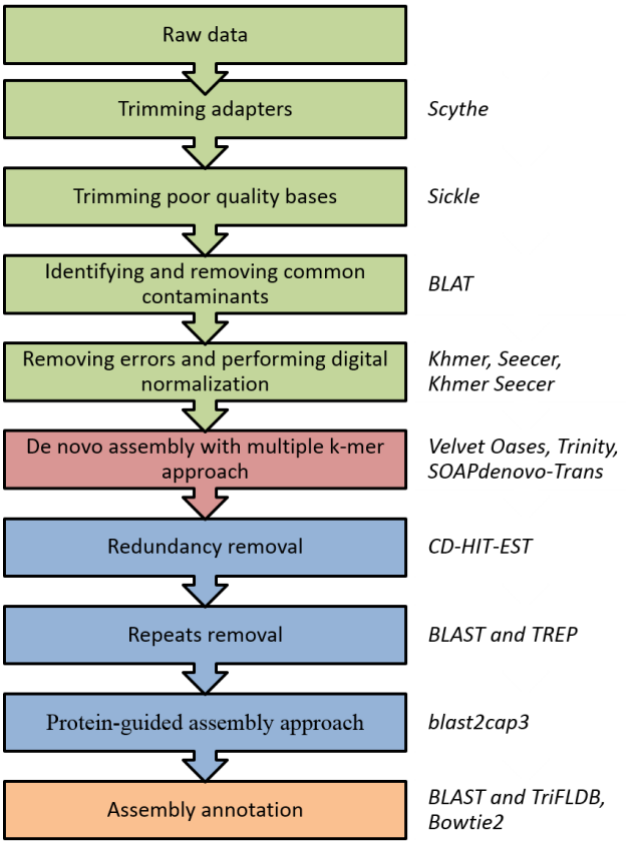

Figure 1. General transcriptome assembly pipeline with some common steps and the tools used for those steps. The rectangles colored in green represent steps that are part of the pre-processing stage. The rectangles colored in blue represent steps that are part of the assembly stage, while the rectangles colored in orange show steps part of the post-processing stage.

There are several available tools that examine the quality of the sequenced reads, their length, quality scores, duplication levels and overrepresented sequences. FastQC [41], FASTX-Toolkit [42], and the R package qrqc [43] are some of the widely used tools. Many software packages have been developed to remove the artificial elements from the sequenced reads. Tools such as Cutadapt [44], Scythe [45] and TagCleaner [46] trim off the adaptors from the raw reads, while Sickle [47] and Prinseq [48] remove the low-

quality bases. The digital normalization algorithm reduces the memory and the computational requirements for the transcriptome assembly by decreasing the differences in gene coverage in RNA-Seq, discarding redundant data and removing most errors [49], while the error correction method indicates significant improvements on the assembly accuracy [50]. After the data is cleaned and filtered in the pre-processing stage, the next step is to generate the transcriptome assembly from the filtered reads. There are two basic approaches in generating a transcriptome assembly: reference-based approach and de novo approach [51][52]. Most of the de novo assemblers are based on *k*-mer lengths and de Bruijn graphs. Velvet Oases [53], SOAPdenovo-Trans [54], Trinity [55], Trans-AbySS [56] are some of the widely used de novo transcriptome assembly tools.

In this Chapter, we develop and analyze 21 different de novo transcriptome assembly pipelines using three de novo assemblers with different range of *k*-mer lengths and different tools and packages for different assembly steps. We evaluate the performance of the pipelines when the digital normalization algorithm, the error correction method and the combination of both are used. Moreover, we investigate the range of *k*-mer lengths that need to be combined with the "*multi-k*" method to produce more accurate and full-length transcriptomes [15]. We additionally compare the generated transcriptome assemblies based on a few common metrics, such as N50 and the number of transcripts, as well as the utilized computational resources, such as memory and runtime. By comparing the performance of these tools and assemblies generated, we draw conclusions and provide guidelines for developing good transcriptome assemblies for a given application.

2.2. Materials and Methods

Here, we implement de novo transcriptome assembly pipelines which incorporate the pre-processing, assembly, and post-processing stages, as well as the assembly annotation. All the experiments in this Chapter are performed on Tusker[1], one of the High-Performance Computing Clusters at the University of Nebraska Holland Computing Center (HCC) [57].

2.2.1. Dataset

For the evaluation of the de novo transcriptome assembly pipelines, the diploid wheat Triticum Urartu (*T. urartu*) dataset is used. The sequencing of the dataset was performed on Illumina HiSeq2000 machine at the University of California Davis (UCD) Genome Center using 100 bp paired-end protocol. This produced a total of 82 GBs of sequence data with 248.5 million reads. The raw sequence *T. urartu* data is publicly available and it was downloaded from the NCBI Sequence Archive database under the NCBI BioProject PRJNA191053 [29].

2.2.2. Software tools

[1] After the experiments for this Chapter were completed, Tusker has been decommissioned and parts of it have been incorporated into Rhino, another High-Performance Computing Cluster at University of Nebraska Holland Computing Center. Since most Clusters are built the same way, the analyses performed here can easily be executed on other Campus and High-Performance Computing Clusters.

Adapter removal. The sequencing technologies attach adapters to one or both ends of the reads. This usually occurs when the read length of the sequenced molecule is shorter than the read length of the sequencer. Because these adapters are not part of the original sequences, the adapters need to be removed prior to the assembly process. There are multiple tools available that allow adapter removal.

Scythe is an adapter removal tool that uses Naive Bayesian approach [45]. It only checks for adapters at the 3' end. The poor-quality base trimming of reads can remove the bases that help in identifying the adapters. Therefore, it is recommended to run adapter removal tools before trimming the poor-quality bases in any assembly pipeline.

Trimming poor-quality bases. The sequences with poor- and low-quality base pairs can cause problems in the RNA-Seq analysis, and it can lead to misassembled, complicated, and even impossible assembly process. Therefore, those sequences need to be removed before the assembly. The quality score of 10 denotes a 1 in 10 chance of an incorrect base, and a quality score of 20 denotes a 1 in 100 chance of an incorrect base. For this work, we trim the poor-quality bases with quality score less than 20.

Sickle is a sliding window quality trimmer which is used after Scythe [47].

Detecting and removing common contaminants. Some organisms are under the influence of different environmental or living contaminators. Homo sapiens DNA, Escherichia coli DNA, wheat mitochondrial and chloroplast sequences and wheat rRNA are considered as commonly known contaminants for wheat and can influence the

generated transcriptome assembly [18]. Therefore, they need to be removed prior to the assembly.

BLAT (BLAST-like alignment tool) is an alignment tool like BLAST in many ways [58]. It is more accurate and about 500 times faster than the existing tools for mRNA/DNA alignments. After finding the common contaminants by aligning the raw reads with BLAT, the aligned reads are removed. The number of reads is reduced in this process, and the assembly quality is improved.

Digital normalization. The NGS produces millions of sequencing reads. The de novo sequence assembly of that large number of reads requires huge computational resources and time. Therefore, the pre-processing steps are important to reduce the size of raw data which might contain many redundant and low-quality reads. The digital normalization is a part of the pre-processing step that significantly reduces the size of the dataset which in turn reduces the memory and time requirements for de novo assembly process. It removes high coverage reads from the dataset and normalizes the coverage to a pre-specified value into nice Gaussian distributions.

Ksenia V. Krasileva et al. [18] tested the effect of digital normalization by comparing two assemblies: one with digital normalization, and second one without digital normalization. Although the number of reads was reduced in the first assembly, both datasets have identical distribution of the number of reference genes assembled at different levels of coverage. This result shows that digital normalization has no negative

effect on the quality of the assemblies, but reduces memory and runttime complexity, and does not require additional sequences and references.

Khmer is a software package that contains a set of bioinformatics programs, including functionality for digital normalization [49].

Error correction. Error correction algorithm can be applied on RNA-Seq raw sequencing data and can significantly impact the quality of the assembly. The error correction algorithm removes mismatch and indel errors from the reads.

Seecer is an error correction algorithm for RNA-Seq datasets based on hidden Markov models (HMM) [50]. Seecer does not depend on a reference genome and can work well with datasets with non-uniform coverage and alternative splicing.

De novo assembly. De novo assembly of plant genomes is a challenging task. In this work, we chose three assemblers, Velvet Oases [53], Trinity [55] and SOAPdenovo-Trans [54]. Velvet Oases and SOAPdenovo-Trans use multiple k-mers and Trinity uses single k-mer length to generate the transcripts.

Velvet is one of the most widely used de novo genome assemblers. It is based on de Bruijn graphs and k-mer approach. Oases is a software package used to generate transcripts from the assembly generated from Velvet. Therefore, the input to Oases is the contigs produced by the Velvet assembler. Trinity is another transcriptome assembler which is also based on the de Bruijn graph. It is a modular assembler consisted of three independent modules: Inchworm, Chrysalis, and Butterfly. These three modules are used

sequentially to produce transcripts. SOAPdenovo-Trans is a transcriptome assembler based on the de Bruijn graph and derived from SOAPdenovo2. The SOAPdenovo-Trans approach is composed of two steps - contig assembly and transcriptome assembly.

Both Velvet Oases and SOAPdenovo-Trans support different k-mer lengths, while Triniy has a fixed k-mer length of 25. Here, we run both Velvet-Oases and SOAPdenovo-Trans with a range of k-mer lengths (k=21, 25, 31, 35, 41, 45, 51, 55, 61, 63, 71, 81, 91).

Merging multiple assemblies and redundancy removal. Yann Surget-Groba and Juan I. Montoya-Burgos [15] proposed the multiple-k method in which various lengths for k are used for the de novo transcriptome assembly. Their experimental results show that the multiple-k method improves the transcript diversity of the assembly and increases its contiguity.

Although the multiple-k method improves the assemblies, there is no right way to determine the range of k lengths that produces the best assembly. For this purpose, for each de novo assembler we investigated the quality of the assembly when groups of 5, 8, and 10 different k lengths were merged (5 k lengths (k=45, 51, 55, 61, 63), 8 k lengths (k=21, 31, 41, 51, 61, 71, 81, 91), and 10 k lengths (k=21, 25, 31, 35, 41, 45, 51, 55, 61, 63)). The 5 k lengths are chosen because the individual assemblies for these k lengths have the highest N50 metric. The 8 k lengths contain the whole range from 21 to 91, while the 10 k lengths are based on the work of Ksenia V. Krasileva et al. [18]. After all the individual assemblies are merged, they are furthered clustered such that the redundancy is removed.

CD-HIT is a clustering tool that was initially designed for proteins, but it also can be used for DNA and RNA datasets [59]. CD-HIT groups a dataset into clusters if the sequences meet a predefined similarity threshold. The similarity threshold usually is a sequence identity that is calculated as number of identical amino acids or nucleotides in the alignment divided by the full length of the shorter sequence.

However, CD-HIT does not merge partially overlapping transcripts, thus blast2cap3 is used [17]. Blast2cap3 is a protein-guided assembly approach that first clusters transcripts based on a similarity to a common protein, and then passes each cluster to CAP3 [60]. CAP3 is used to remove redundancy by merging overlapping reads with minimum identity and similarity of 99% into a single transcript. Before running CAP3, more repetitive sequences were removed using the Triticeae Repeat Sequence Database (TREP) (BLASTN and BLASTX, E-value cutoff 1e-10) [61] to lower the risk of merging incorrect transcripts.

2.3. Results and Discussion

2.3.1. Pre-processing stage

The overall read quality of the 248.5 million 100 bp raw *T. urartu* Illumina paired-end reads is assessed using FastQC. After this initial quality check, the first step from the pre-processing stage is to remove the artificially added Illumina adaptors and trim off the reads with average quality score under 20 and length less than 20bp. The adapters were removed using Scythe, and the poor-quality bases were trimmed using Sickle.

Next, we investigate the importance of using digital normalization, and error correction on the reads before the assembly by using the programs Khmer, Seecer and combination of both (Khmer-Seecer). For better understanding of the results, we denote the datasets of the processed reads generated from Khmer, Seecer and Khmer-Seecer with Pipeline K, Pipeline S and Pipeline KS, respectively.

Afterwards, common contaminants from several environmental contaminants were detected and removed.

The total number of reads after each step of the pre-processing stage is shown on Figure 2. As it can be seen on the Figure, the removal of adapters changes the length of the reads, but not their number. On the other hand, during the trimming process if one of the paired-end reads is trimmed, the other read is saved as a single-end read in a separate output file. From the results generated when Khmer, Seecer, and Khmer-Seecer are used, we observe a huge difference in the total number of reads after Khmer and Seecer. When Khmer is used, the total number of reads is reduced to 57,359,540, while when Seecer is used, the total number of corrected reads is 246,593,875. The digital normalization algorithm removes redundant reads and errors and evens out the coverage. Because of its deep analyses, the digital normalization significantly reduces the size of the data set. The error correction method removes errors from the raw data just by reducing the read length, but not the total number of reads. When the combined approach of both Khmer and Seecer is used, we notice that the total number of reads is same as the number of reads when only Khmer is used. Therefore, we can say that the error correction method is just a part of the more general digital normalization approach. Next, with the removal of

common contaminants, such as *Homo Sapiens DNA*, *Escherichia coli DNA*, wheat mitochondrial and chloroplast sequences and *wheat rRNA*, with BLAT, the datasets sizes reduce slightly to 56,700,858 corrected reads for Pipeline K and Pipeline KS, and 246,593,875 reads for Pipeline S.

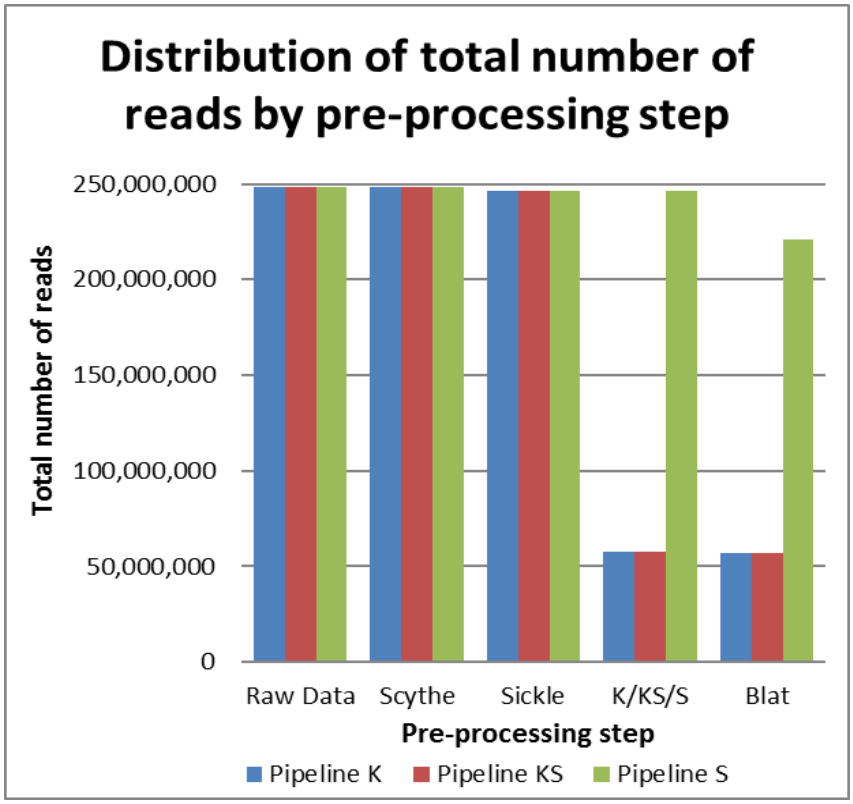

Figure 2. Distribution of total number of reads by different steps of the pre-processing stage for Pipeline K, Pipeline KS, and Pipeline S.

2.3.2. De novo transcriptome assembly

After the pre-processing stage, the paired- and single-end reads from Pipeline K, Pipeline KS and Pipeline S are used for the transcriptome assembly. In this work, three de novo assemblers, Velvet Oases, SOAPdenovo-Trans and Trinity are individually used and

evaluated. Assemblies are carried out using the three datasets Pipeline K, Pipeline KS and Pipeline S. For Velvet Oases and SOAPdenovo-Trans 13 individual assemblies are constructed when the *k*-mer length is 21, 25, 31, 35, 41, 45, 51, 55, 61, 63, 71, 81 and 91 respectively.

For each assembler and dataset, the importance of the *k*-mer length is further investigated by comparing the final assembly pipeline outputs when 5 *k*-mer lengths (*k*=45, 51, 55, 61, 63), 8 *k*-mer lengths (*k*=21, 31, 41, 51, 61, 71, 81, 91), and 10 *k*-mer lengths (*k*=21, 25, 31, 35, 41, 45, 51, 55, 61, 63) are grouped together, respectively. Therefore, we have 9 different assembly pipelines for Velvet Oases, 9 different assembly pipelines for SOAPdenovo-Trans and 3 different assembly pipelines for Trinity. For better understanding of the results, we denote the assemblies generated from Khmer, Seecer and Khmer-Seecer with ***K-Xk***, ***S-Xk*** and ***KS-Xk***, respectively, where ***X*** is the group of *k*-mer lengths they belong to (|*k*|=5, 8, 10).

One of the commonly used metrics to compare the generated assemblies is N50. N50 is a weighted median statistic which value represents the length of the shortest transcript (contig) in the group of longest sequences that together represent at least 50% of the total number of nucleotides in the set of sequences. Figure 3 and Figure 4 show the distribution of the N50 lengths for all 13 assemblies with different *k*-mer lengths for Velvet Oases and SOAPdenovo-Trans with Pipeline K, Pipeline KS and Pipeline S, respectively. Since Trinity uses only one value for *k* of 25, the N50 length for Pipeline K is 2,714 bp, for Pipeline KS is 2,791 bp and for Pipeline S is 2,471 bp. In general, the higher the N50 value is, the more complete the assembly is. As it can be seen on the

Figures, Pipeline S shows the lowest N50 values for the three assemblers. Moreover, when Velvet Oases is used, the higher N50 values are within the *k*-mer length range of 35 and 45. When SOAPdenovo-Trans is used, the higher N50 values are within the *k*-mer length range of 35 to 81. For the two assemblers, the *k*-mer lengths of 21 and 91 produce the lowest N50 values. This is because low *k*-mer lengths produce more repetitive sequences that cannot unambiguously map, and high *k*-mer lengths produce sequences that are hard to further extend and overlap. This can also be observed on Figure 5 and Figure 6, where the distribution of the number of transcripts generated from different *k*-mer lengths for Velvet Oases and SOAPdenovo-Trans is shown respectively.

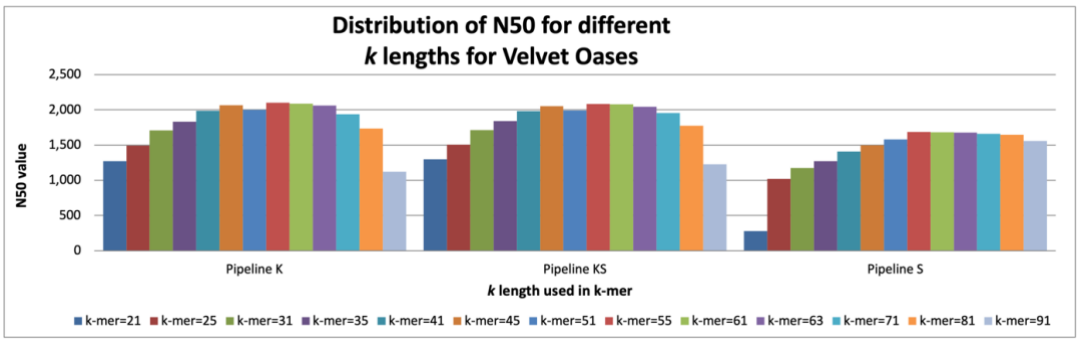

Figure 3. Distribution of N50 value for different *k*-mer lengths for Velvet Oases for Pipeline K, Pipeline KS and Pipeline S respectively.

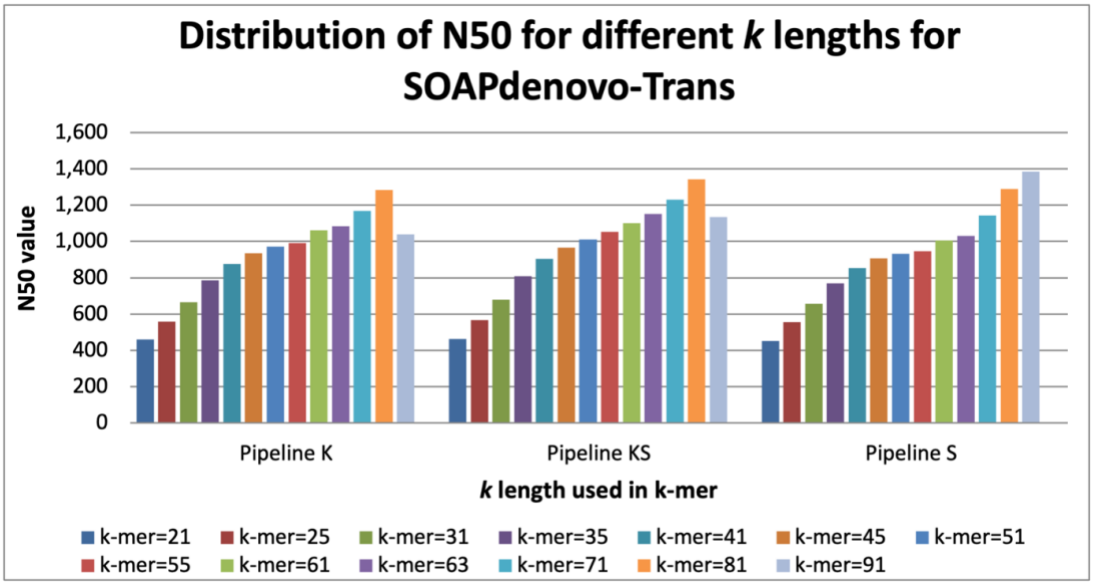

Figure 4. Distribution of N50 value for different *k*-mer lengths for SOAPdenovo-Trans for Pipeline K, Pipeline KS and Pipeline S respectively.

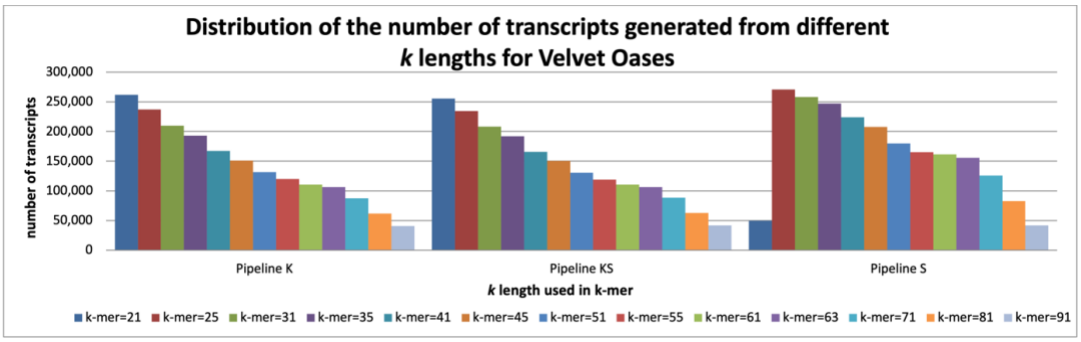

Figure 5. Distribution of the number of transcripts generated from different *k*-mer lengths for Velvet Oases for Pipeline K, Pipeline KS and Pipeline S respectively.

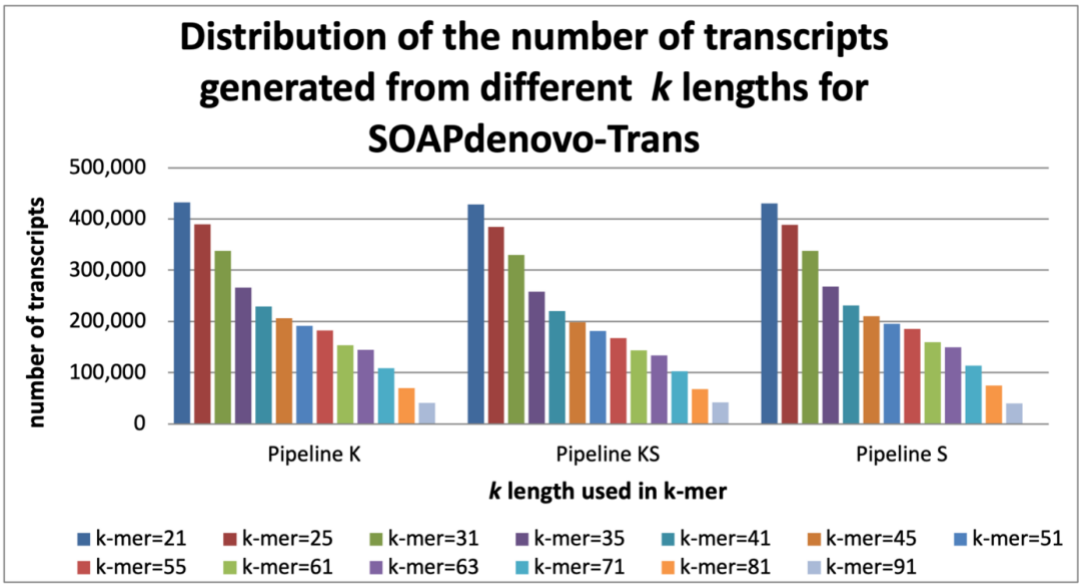

Figure 6. Distribution of the number of transcripts generated from different *k*-mer lengths for SOAPdenovo-Trans for Pipeline K, Pipeline KS and Pipeline S respectively.

2.3.3. Post-processing stage

After the assembly process, the next stage is the post-processing. Since the "*multi-k*" strategy introduces redundancy, it needs to be further removed using CD-HIT, *blast2cap3* and CAP3. Also, to lower the possibility of merging incorrect transcripts, repetitive sequences are identified and removed using the Triticeae Repeat Sequence Database (TREP) and BLAST.

Once the post-processing stage is complete, the 21 transcriptome assemblies are evaluated based on the previously used assembly quality metrics. The distribution of the N50 length is shown on Figure 7. The post-processing steps significantly increase the N50 lengths for Velvet Oases. On the other hand, for the three assemblies generated with Trinity, we can notice that these additional post-processing steps actually decrease the

N50 lengths. SOAPdenovo-Trans does not show big improvement with the post-processing steps either.

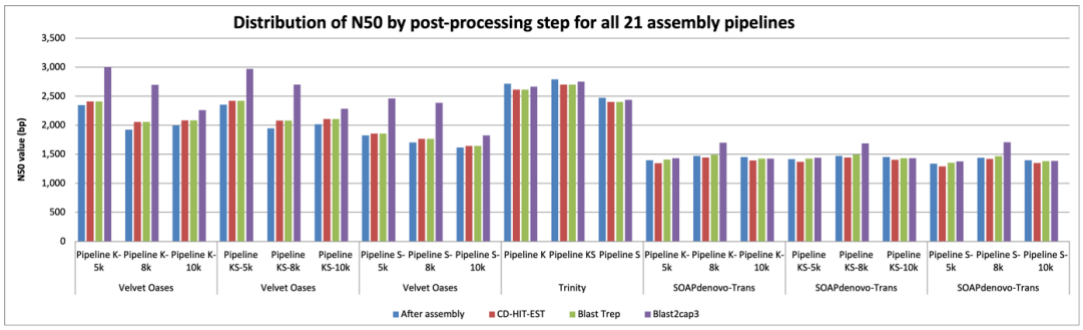

Figure 7. Distribution of N50 value for different k-mer lengths in the post-processing stage for Velvet Oases, SOAPdenovo-Trans and Trinity for Pipeline K, Pipeline KS and Pipeline S respectively.

2.3.4. Mapping raw reads to each assembly

To get assembly statistics for the number of raw reads mapped to the resulting transcripts we use Bowtie2 [62] and Samtools [63] to perform the mapping and examine the resulting *.bam* files. The percentages of paired- and single-end reads mapped more than once to the resulting transcripts, as well as the overall alignment rate is shown on Table 1 (part A). All three assemblers, datasets and groups of *k*-mer lengths show a high alignment rates which tells us that most of the raw reads are used in the transcriptome assembly. Parallel to the final transcriptome assemblies, we also map the raw reads to the transcripts generated before the post-processing stage. These results are also shown in Table 1 (part A). From these results, we can observe that in most cases, the overall alignment rates are either same or slightly higher for the assemblies obtained before the

post-processing stage. While the alignment rates are very close in both stages, the assemblies generated with Pipeline S show slightly higher alignment percentage.

Table 1. Assembly annotation summary statistics for all 21 assembly pipelines. Part A: Alignment rates when the raw reads are mapped to the transcripts. Part B: Alignment rates when the transcripts are mapped against the TriFLDB database.

assembler	dataset Pipeline *	A. Map raw reads against transcripts						B. Map transcripts against the TriFLDB database					
		paired-end reads aligned more than once		single-end reads aligned more than once		overall alignment rate		total number of transcripts		transcripts aligned more than once		overall alignment rate	
		after post-processing	before post-processing	after post-processing	before post-processing	after post-processing	before post-processing	after post-processing	before post-processing	after post-processing	before post-processing	after post-processing	before post-processing
Velvet Oases	K-5k	64.32	68.50	4.76	4.87	97.34	97.35	210,463	285,535	47.58	39.09	62.55	57.96
	K-8k	70.45	73.11	4.91	4.96	97.99	98.00	409,262	606,175	47.42	35.59	62.79	56.14
	K-10k	71.23	73.25	4.91	4.94	97.95	97.96	377,440	790,870	41.45	37.00	57.51	56.29
Velvet Oases	KS-5k	63.98	68.10	4.78	4.88	97.49	97.50	206,450	280,393	46.64	38.54	62.16	57.78
	KS-8k	70.11	72.67	4.92	4.97	98.11	98.12	399,845	587,113	47.42	35.64	62.77	56.09
	KS-10k	70.84	72.89	4.92	4.95	98.10	98.11	367,166	765,377	40.90	36.61	57.07	55.91
Velvet Oases	S-5k	79.51	80.58	1.47	1.49	99.30	99.31	356,696	428,340	56.89	46.28	67.85	62.42
	S-8k	81.90	82.88	1.49	1.51	99.46	99.46	466,964	593,277	54.62	41.83	66.22	59.31
	S-10k	81.15	81.91	1.50	1.51	99.50	99.51	441,723	867,719	47.83	42.14	61.21	58.97
Trinity	K	63.47	68.19	4.66	4.90	97.65	97.66	407,585	410,383	32.89	20.95	46.39	35.24
	KS	63.06	67.90	4.66	4.91	97.77	97.78	403,359	576,175	32.76	32.55	46.19	47.37
	S	52.89	78.11	1.07	1.48	98.95	98.96	291,401	568,944	20.75	32.47	33.66	47.20
SOAP denovo-Trans	K-5k	17.80	21.93	2.16	2.51	97.69	97.69	108,464	121,305	25.01	23.58	43.48	41.43
	K-8k	28.45	32.57	2.82	3.15	98.36	98.36	130,594	145,822	25.20	22.43	43.52	40.28
	K-10k	23.51	32.04	2.57	3.13	98.38	98.38	130,328	163,532	23.18	22.24	40.30	40.42
SOAP denovo-Trans	KS-5k	17.33	21.35	2.15	2.48	97.83	97.83	106,096	118,832	24.99	23.56	43.55	41.55
	KS-8k	27.50	31.68	2.78	3.10	98.46	98.46	128,455	144,689	24.86	22.33	43.28	40.37
	KS-10k	23.01	31.32	2.55	3.09	98.49	98.49	128,432	163,184	23.12	22.21	40.22	40.41
SOAP denovo-Trans	S-5k	25.91	30.65	0.75	0.82	99.32	99.32	111,523	123,813	25.83	24.16	44.39	42.11
	S-8k	43.76	47.95	0.98	1.05	99.58	99.58	135,806	149,507	26.27	23.17	44.84	41.10
	S-10k	33.39	43.61	0.86	1.01	99.55	99.55	134,221	166,237	23.92	22.79	41.05	40.82

2.3.5. Annotation of the final transcriptome assemblies

To test the overall quality of the assembly pipelines and techniques, we align the resulting transcripts to 19,200 sequences from full length common cDNA wheat dataset from TriFLDB with average read length of 1,652.9 bp using BLASTN [64]. The total

number of transcripts, the number of transcripts aligned more than once and the overall alignment rate before and after the post-processing stage is shown in Table 1 (part B).

Here, we observe that better alignment rate occurs when Velvet Oases is used as the de novo assembler with all datasets. Moreover, the best alignment rates are achieved when the Seecer dataset is used. For Velvet Oases, the best assembly is produced with Pipeline S when the 5 *k* assemblies with highest N50 are used in the merging process. For SOAPdenovo-Trans, slightly better results are observed when the group of 8 *k* values is used with Pipeline S. Using all the assemblies for various *k*-mer lengths did not improve the assembly quality. While the post-processing steps slightly improve the overall alignment rate for all pipelines, interestingly, when Trinity was used as transcriptome assembler, Pipeline S showed lower alignment rate after the post-processing step.

2.3.6. Comparison of digital normalization algorithm, correction method, and combination of both

One of the objectives addressed in this work, is to investigate the importance of using a digital normalization algorithm, an error correction method or combination of both on the reads before the assembly step. For this purpose, we individually apply Khmer, Seecer and Khmer-Seecer (combination of both) to the trimmed and filtered raw reads.

When Seecer is used with the trimmed and filtered reads, the total number of reads in Pipeline S is reduced by 1,938,921 reads. For all three assemblers and three groups of *k*-mer lengths, the assemblies generated with Pipeline S give the highest alignment rates. When Khmer is used with the trimmed and filtered reads, the total number of reads in

Pipeline K is reduced by 77% (57,359,540). The same number of reads occurs when the combined approach of both Khmer and Seecer is used as well. After the post-processing stage, slightly higher N50 lengths and alignment rates are generated for Pipeline KS compared to Pipeline K.

The digital normalization algorithm significantly reduces the computational and assembly costs. However, Seecer outperforms this algorithm in the number of aligned reads and full-length assemblies.

2.3.7. Comparison of the efficiency of different k-mer lengths

The *k*-mer length (*k* value) affects the accuracy of the overall assembly. Shorter *k*-mer lengths are better for less expressed transcripts, while larger *k*-mer lengths produce higher coverage. From Figure 7, we can observe that the group of 10 *k*-mer lengths gives the worst N50 value. On the other hand, better performance is observed when the group of 5 and the group of 8 *k*-mer lengths are used for Velvet Oases (Pipeline K-5k, Pipeline KS-5k, Pipeline S-5k) and SOAPdenovo-Trans (Pipeline K-8k, Pipeline KS-8k, Pipeline S-8k), respectively. Moreover, from Table 1, when the raw reads are mapped against the transcripts, lower alignment rate occurs when $|k|$ is 5, while the highest is when $|k|$ is 8. On the other hand, when the transcripts are aligned to the TriFLDB database, lower alignment rate is observed for $|k|$ of 10, while for both $|k|$ of 5 and 8, the alignment rates are significantly better. Even though we cannot claim the best range of *k*-mer lengths that produces the best assembly, we believe that using all the assemblies for various *k*-mer lengths does not improve the assembly quality. On the contrary, these values need to be

chosen based on the highest N50 value and/or additional metrics that need to be further investigated.

2.3.8. Comparison of three de novo transcriptome assemblers

To compare the performance of each assembler, we measure the number of assembled transcripts, and their N50 length. Moreover, for each generated assembly, we calculate the number of paired- and single-end reads aligned more than once when the raw reads are mapped against the transcripts. Also, the overall alignment rate is calculated when the transcripts are mapped against the TriFLDB database.

During the post-processing steps, SOAPdenovo-Trans is constantly reporting the lowest number of transcripts and the lowest N50 lengths. The performance of Trinity is second, while Velvet Oases has the highest N50 for all three datasets (Pipeline K, Pipeline KS, Pipeline S) and three groups of *k*-mer lengths ($|k|$=5, 8, 10). The percentage of paired- and single-end reads mapped more than once against the transcripts is shown on Table 1 (part A). The overall alignment rate when the transcripts are mapped against the TriFLDB database is shown on Table 1 (part B).

2.3.9. Comparison of computational resources used for each assembly

Transcriptome assembly pipelines are composed of multiple steps that require lots of computational resources. The memory and the runtime can be one of the main bottlenecks, especially when the de novo assembly is generated. Here, we compare the maximum memory and runtime utilized for each assembly with different *k*-mer lengths.

Figure 8 shows the distribution of maximum memory in GBs for each assembler and used *k*-mer length. The smaller the *k*-mer length is, the more memory is needed. While SOAPdenovo-Trans used the least memory (ranging from 21 GBs to 161 GBs), Velvet Oases used the most (ranging from 34 GBs to 478 GBs). On the other hand, the memory used by Trinity varied from 85 GBs to 193 GBs. Both Pipeline K and Pipeline KS used significantly less memory than Pipeline S. This just shows how important digital normalization is for being able to assemble transcripts with limited computational resources.

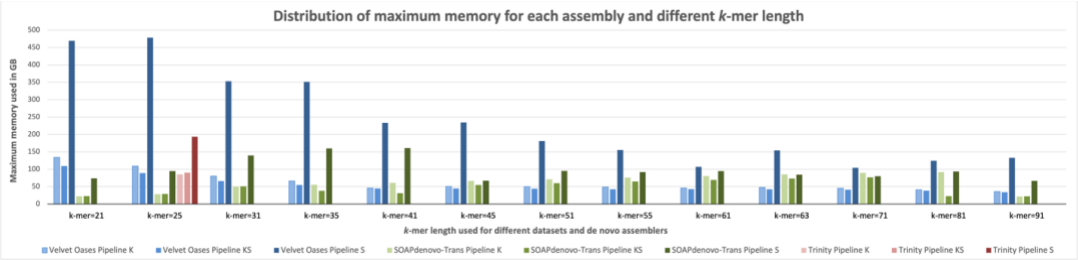

Figure 8. Maximum memory in GBs used by Velvet Oases, SOAPdenovo-Trans and Trinity for *k*-mers with different lengths for Pipeline K, Pipeline KS and Pipeline S.

Figure 9 shows the total running time in hours for each assembler and different *k*-mer length. In general, assemblies with smaller *k*-mer lengths and assemblies that only used the error correction method (Pipeline S) took longer to run. SOAPdenovo-Trans had the shortest running time across all datasets and *k*-mer lengths used ranging from 0.3 to 4.4 hours. The runtime of Trinity was 48.5, 46.7, 64.1 hours for Pipeline K, Pipeline KS and Pipeline S respectively. While the runtime for Velvet Oases for Pipeline K and Pipeline KS varied between 0.4 and 2.6 hours, the runtime was significantly higher for Pipeline S,

especially for the lower *k*-mer lengths. Namely, Velvet Oases ran for 140, 54, 25 and 16 hours for *k*-mer lengths of 21, 25, 31, and 35 for Pipeline S respectively. Due to the huge difference in runtime for different *k*-mer lengths, the selection of *k*-mer lengths is crucial.

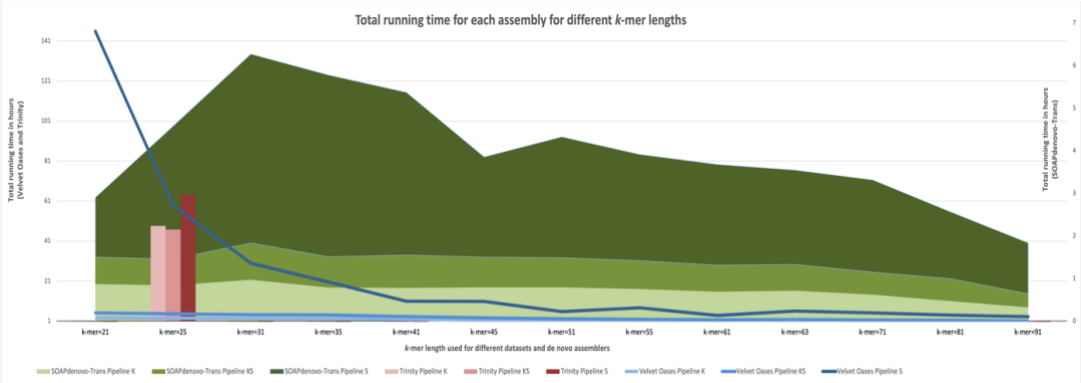

Figure 9. Runtime in hours used by Velvet Oases, SOAPdenovo-Trans and Trinity for *k-mers* with different lengths for Pipeline K, Pipeline KS and Pipeline S.

2.4. Conclusion

In this Chapter, we developed a bioinformatics assembly pipeline, and analyzed different tools used for the different steps of the pipeline. Analyzing 9 different assemblies generated by Velvet Oases, 9 different assemblies generated by SOAPdenovo-Trans, and 3 different assemblies generated by Trinity, we can observe that using the error correction method with Velvet Oases and merging the individual *k*-mer assemblies with highest N50 produce the most stable base for further transcriptome biological analysis.

Moreover, with the experiments provided here, we find the following useful insights for choosing the best strategy for optimizing de novo transcriptome assembly pipelines:

- Both the digital normalization algorithm and error correction method are useful pre-processing steps;
- Using the digital normalization algorithm significantly reduces the computational resources;
- The "*multi-k*" method gives better overall assembly results;
- From the results obtained here, we believe that the single *k*-mer assemblies with highest N50 lengths combined together with the "*multi-k*" strategy lead to better transcriptome;
- In our experiments Velvet Oases remains the best de novo transcriptome assembler;
- SOAPdenovo-Trans utilizes the least memory and runs the fastest;
- Shorter *k*-mer lengths require more powerful computational resources;
- The post-processing stage improves the overall transcriptome quality.

Utilizing error correction methods, as well as merging assemblies with *k*-mer lengths that have good metrics can improve the overall quality of the transcriptome assembly and provide stable base for further transcriptome biological analysis. Developing a multi-stage assembly pipeline is an important and crucial part for generating accurate and meaningful transcriptome assembly. Since choosing the optimal tools and parameters for building quality transcriptome assembly pipelines is a difficult task, the experiments performed as part of this thesis provide useful guidelines for choosing the best strategy for optimizing de novo transcriptome assembly pipelines.

Acknowledgements

This work was completed utilizing the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative.

Chapter 3

Evaluating Distributed Platforms for Protein-Guided Scientific Workflow

3.1. Introduction

The advances in life sciences and information technologies have led to proliferation of scientific data that needs to be stored and analyzed. The analysis of this so called "big data" is done by using a complex set of multitudes of software tools. A sequential series of these tools is known as an analysis pipeline [32]. The "big data" is too large to be processed by using only local computational resources. A possible approach to this problem is to make better use of multiple distributed resources including multi-core computers.

Scientists use various workflow systems to conduct their research modularly. This indicates that the whole scientific workflow can be decomposed into multiple sub-workflows that can be executed in parallel on distributed resources. Each workflow is composed of computational tasks, the order of execution of which is determined by the dependencies among the tasks [4]. The advantages of scientific workflows include automated complex analysis, real-time results and improved time performance that allow scientists to easily design, execute, debug, modify and re-run their experiments [20].

Over the past decade, several scientific workflows have been created and introduced. Pegasus Workflow Management System (Pegasus WMS) automatically maps high-level scientific workflows organized as directed acyclic graph (DAG) onto available distributed resources [5]. DAGMan (Directed Acyclic Graph Manager) is a meta-

scheduler that submits jobs to Condor [7] in an order defined in DAG, and processes the results afterwards [6]. Taverna [8] is an open-source workflow system that graphically connects bioinformatics web services together into a coherent flow. Kepler [9] also has a visual interface and separates the structure of the workflow model from its model of computation. The number of applications using scientific workflow systems has been steadily increasing [10].

The resources required by scientific workflows may exceed the capabilities of the local computational resources. Therefore, the scientific workflows are usually executed on distributed platforms, such as Campus Clusters or Grids. Grids such as Open Science Grid (OSG) [11] and XSEDE [12] allow distributed computing where the computational resources are spread on a geographically remote location. Beside the Cluster and Grid execution platforms, lately the scientists are analyzing the benefits of using Clouds for these scientific workflows. Cloud computing platforms like the commercial Amazon Elastic Compute Cloud [13] or the academic FutureGrid [14] provide rentable computational and storage resources over the Internet. Despite the advantages and disadvantages of Clusters, Grids and Clouds [33], the execution of scientific workflows deals with different challenges depending on the chosen computational platform.

In this Chapter we build a scientific workflow for *blast2cap3*, the protein-guided assembly, using Pegasus WMS. We chose two execution platforms for this workflow, Campus Cluster, and distributed Grid. Furthermore, we compare the running time and used resources for both platforms when the workflow is executed serially and parallel with alternating number of tasks.

3.2. Materials and Methods

3.2.1. BLAST2CAP3: Protein-Guided Assembly

With the recent and rapid development of next-generation sequencing technologies (NGS), RNA-Seq has become a powerful way of creating and analyzing transcripts and quantifying gene expression levels of different organisms [40]. Obtaining draft transcriptome of a raw sequence data is a complex multi-stage process usually composed of pre-processing, de novo assembly, and assembly validation. The most widely used de novo assemblers that produce transcripts are based on de Bruijn graphs and *k-mers* [51][52]. After generating assemblies with multiple *k-mer* length, the resulting transcripts should be grouped together. This grouping causes high redundancy of transcripts. Therefore, the transcripts need to be further merged in larger ones. Assemblers like CAP3 [16] are often used to merge transcripts based on overlapping regions and nucleotide similarity. However, because most of the produced transcripts code for a protein, a protein similarity should be also considered during the merging. Saturating the time and the memory limits and disregarding the protein similarity cause CAP3 to frequently lead to incorrect results.

Blast2cap3 [17] is a protein-guided assembly approach that first clusters the transcripts based on similarity to a common protein and then passes each cluster to CAP3. The recent use of *blast2cap3* on the wheat transcriptome assembly [18] shows that *blast2cap3* generates fewer artificially fused sequences compared to assembling the entire dataset with CAP3. Moreover, it also reduces the total number of transcripts by 8-9% [18].

Before running *blast2cap3*, the assembled transcripts are aligned with protein datasets closely related to the organism for which the transcripts are generated. BLASTX [19] is used for this alignment. Afterwards, transcripts sharing a common protein hit are merged using CAP3. Therefore, *blast2cap3* uses the assembled transcripts and the BLASTX alignments as input files.

3.2.2. Pegasus Workflow Management System

Pegasus Workflow Management System (Pegasus WMS) stands for Planning for Execution in Grids. Pegasus WMS is a framework that automatically maps high-level scientific workflows organized as directed acyclic graph (DAG) onto wide range of execution platforms, including Clusters, Grids, and Clouds [5]. Pegasus receives an abstract workflow and tries to simplify it before mapping it into a concrete workflow. The abstract workflow of Pegasus contains information and description of all executable files (transformations) and logical names of the input files used by the workflow. On the other hand, the concrete workflow specifies the location of the data and the execution platform [23]. The concrete workflow is then submitted to Condor's DAGMan meta-scheduler [6] for execution [21]. The high-level of abstraction of Pegasus allows scientists to ignore low-level configurations required by the middleware and the underlying execution platform [23].

DAG-based workflows use nodes to define the tasks and use the edges to denote the task dependencies. In DAG-based workflows, the structure can be characterized as *sequence* and *parallel* [20]. The sequence structure is defined as an ordered series of

tasks, where one task starts after the previous task is completed. The parallel structure allows concurrently execution of tasks. Pegasus also allows clustering of small tasks into larger clusters that are scheduled and executed to the same remote site. This setting allows improvement of the performance and reducing the remote execution overheads [22].

Pegasus uses DAX (directed acyclic graph in XML) files to specify an abstract workflow. The DAX file contains syntax for defining jobs, arguments, input and output files, and dependencies between the various tasks. This format is shared by many workflow tools. The DAX file can be created manually, or by using the Pegasus API. Pegasus uses Java, Perl, or Python libraries for writing DAX generators [22]. The abstract DAX is then mapped to one or more execution sites. This step is known as the *planning stage*.

Pegasus comes with a set of useful command-line tools that help users to submit and analyze the workflows, and generate useful statistics and plots about the workflow performance, running time, execution results, machines used, as well as for succeeded and failed tasks [22]. *Pegasus-plan* is used to plan the workflow, while *pegasus-run* is used to submit the workflow to DAGMan. After the workflow is submitted, it can be monitored using the *pegasus-status* command that shows information about the running jobs and the percentage of finished jobs. The whole workflow and the failed jobs can be debugged using the *pegasus-analyzer* tool. After the workflow execution ends, the resulting data can be summarized using *pegasus-statistics* and *pegasus-plots*.

Pegasus is used in a number of large scientific applications built for physics, astronomy, biology, earthquake sciences, ocean sciences, limnology and many other domains [23][24][25][26]. Pegasus can use both single systems and heterogeneous set of resources for executing the scientific workflows. The used resources can be distributed across laptops, Campus Clusters, Grids and Cloud platforms. Furthermore, Pegasus can support workflows ranging from few computational tasks to a few millions.

Scalability and handling large sets of data and computations, portability and ease of use are just part of the advantages that Pegasus has. In case of a job or data transfer failure, Pegasus can retry the job or the entire workflow given number of times. If the job fails again, then Pegasus generates a rescue workflow that contains information of the work that remains to be done such that it can be modified and resubmitted later. Therefore, Pegasus has capabilities for provenance tracking, execution monitoring and management, and error recovery.

3.2.3. Execution platforms

The resources that these scientific workflows require can exceed the capabilities of the local computational resources. Therefore, the scientific workflows are usually executed on distributed platforms, such as Campus Clusters, Grids or Clouds. These platforms are usually a set of heterogeneous hosts that are connected via a network. The experiments performed in this work were executed on two different computational platforms – a Campus Cluster and distributed Open Science Grid.

University of Nebraska Campus Cluster. Campus and other public clusters are shared by diverse communities of users and enforce fair-share scheduling and file and disk spaces quotas. These clusters are suitable for various types of jobs, such as serial, parallel, GPU, and high memory specific jobs, thus the high-performance.

Sandhills[2] is one of the High-Performance Computing (HPC) Clusters at the University of Nebraska Holland Computing Center (HCC) [27]. Sandhills was acquired by combining grants from various research groups at University of Nebraska. It is used by faculty and students in disciplines like bioinformatics, nanoscale chemistry, subatomic physics, meteorology, genomics, and artificial intelligence. Sandhills was constructed in 2011 and it has 1440 AMD cores housed in a total of 44 nodes. Each node has storage of approximately 1.5 TB. Sandhills is a heterogeneous cluster in terms of individual node resources.In order to use any of the HCC's Clusters, users obtain an HCC account associated with a University of Nebraska faculty or research group.

Open Science Grid (OSG). The Open Science Grid (OSG) is a distributed, high-throughput distributed computational platform for large-scale scientific research [28]. OSG is a national consortium of more than 100 academic institutions and laboratories that provide storage and tens of thousands of resources to OSG users. These sites share their idle resources via OSG for opportunistic usage. Because of its opportunistic approach, OSG as a platform is ideal for running massive numbers of independent jobs

[2] After the experiments for this Chapter were completed, Sandhills has been decommissioned and parts of it have been incorporated into Rhino, another High-Performance Computing Cluster at University of Nebraska Holland Computing Center. Since most Clusters are built the same way, the analyses performed here can easily be executed on other Campus and High-Performance Computing Clusters.

that require less than 10GB of RAM, less than 10GB of storage, and less than 24 hours running time. If these conditions are fulfilled, in general, OSG can provide unlimited resources with the possibility of having hundreds or even tens of thousands of jobs running at the same time. The OSG resources are Linux-based, and due to the different sites involved, the hardware specifications of the resources are different and vary. Access and use of OSG is free for academic purposes and the user's institution does not need to be part of OSG to use this platform.

3.2.4. Datasets

For the purpose of this experiment, we used diploid wheat *Triticum urartu* dataset to create the transcriptome assembly. The public NCBI BioProject PRJNA191053 [29] contains all sequence libraries submitted by the UCD group. The assembled transcripts were generated using Velvet Oases [53] as a de novo assembler. Afterwards, these transcripts were aligned with protein datasets of closely related wheat organisms, such as *Barley*, *Brachypodium*, *Rice*, *Maize*, *Sorghum* and *Arabidopsis* [18].

The input file "*transcripts.fasta*" is 404 MB big, and has 236,529 assembled transcripts. Moreover, the BLASTX tabular output, "*alignments.out*", is 155 MB big and contains 1,717,454 protein hits. These two files, "*transcripts.fasta*" and "*alignments.out*" are used as input to *blast2cap3*.

3.2.5. Current implementation of blast2cap3

Blast2cap3, the protein-guided assembly, is a Python script written by Vince Buffalo [17]. Beside Python modules [30], *blast2cap3* also uses Biopython [31], and CAP3 [16]. The current implementation of *blast2cap3* supports only serial execution. This means that first one cluster of similar transcripts is created and then is sent to CAP3. After the CAP3 program terminates, this process is repeated consecutively for all possible clusters of transcripts.

When the existing implementation of *blast2cap3* was run on Sandhills for the given input files "*transcripts.fasta*" and "*alignments.out*" with size of 404 MB and 155 MB respectively, the running time was 100 hours. Considering larger input files and datasets, the time requirements and complexity of running the protein-guided assembly grow. *Blast2cap3* is only one step of the many steps that are part of transcriptome assembly pipelines that can be more computationally and data-intensive.

Each cluster of transcripts that is generated from *blast2cap3* and uses CAP3 is an individual process. This means that as long as the final results from CAP3 for each cluster are concatenated at the end, the transcripts within the cluster can be generated and merged independently.

Therefore, an additional approach to *blast2cap3* execution should be considered that requires not just a single computer, but multiple computational nodes that can take advantage of the modularity of *blast2cap3* execution.

3.2.6. Pegasus Workflow Management System implementation of blast2cap3 for Campus Cluster

The modularity of *blast2cap3* allows us to decompose the existing approach on multiple tasks, some of which can be run in parallel. Therefore, this protein-guided assembly can be structured into a scientific workflow using the Pegasus Workflow Management System. The main reduction in the running time of the current implementation of *blast2cap3* is expected to be reached when the merging of transcripts belonging in a cluster is done in parallel for all clusters.

The Pegasus WMS implementation of *blast2cap3* for Campus Cluster is shown on Figure 10.

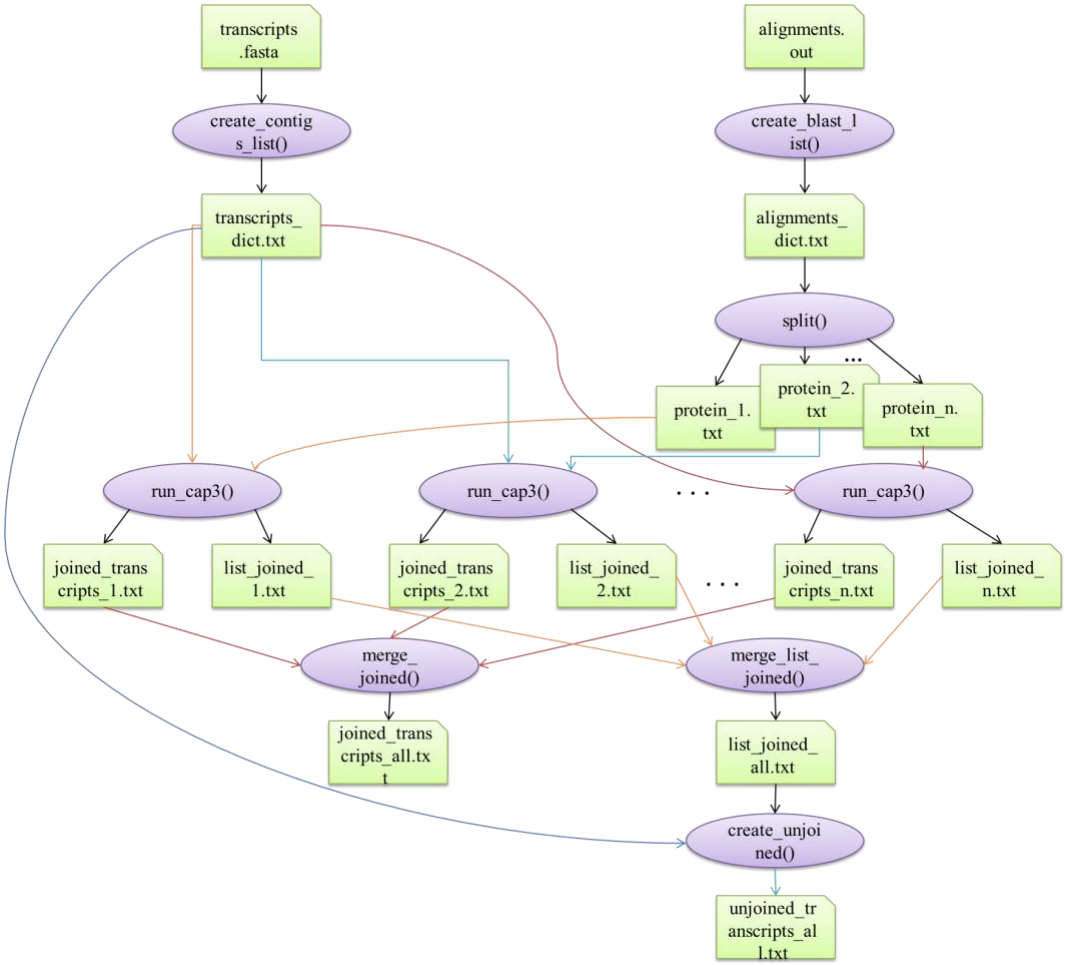

Figure 10. Pegasus WMS implementation of *blast2cap3* for Campus Cluster, where the squares represent the input and output files, the ovals represent the tasks, and the arrows represent the dependencies between the tasks.

For this workflow, we first create lists of both input files, "*transcripts.fasta*" and "*alignments.out*", respectively. These two tasks are independent of each other and can be run at the same time. Furthermore, in order to create multiple clusters of transcripts, the *split()* task is used to divide the big "*alignments.out*" file on "*n*" smaller files. For the purpose of these analyses, we use different values of "*n*", such as 10, 100, 300, and 500.

The number of tasks that merge the transcripts within a cluster depends on "*n*", the number of clusters. From the workflow shown on Figure 10, we can notice that this task, *run_cap3()*, uses two input files, "*transcripts_dict.txt*" and "*protein_n.txt*".

After "*n*" output files are generated from *run_cap3()*, the next step is to merge all these joined transcripts into one file. Knowing the transcripts that are joined helps us to combine all transcripts that are not joined into a new file.

The DAG structure of the workflow is helpful to define dependencies and execute a task if and only if its predecessor tasks have finished.

3.2.7. Pegasus Workflow Management System implementation of blast2cap3 for OSG

The Pegasus WMS implementation of *blast2cap3* for OSG is shown on Figure 11. The workflow and the logic behind both execution platforms differ only in the way how certain tasks are defined. The resources provided by Sandhills, the Campus Cluster, contain the most frequently used libraries, modules and software tools. This means that the Python and Biopython libraries and the CAP3 executable required by *blast2cap3* are already set and maintained on the Campus Cluster. On the other hand, the resources provided by OSG are more heterogeneous and most of the time belong to other academic institutions and laboratories that may provide different software and system configurations.

When the required libraries and executables like Python, Biopython and CAP3 are not installed on the remote node, the workflow execution fails. In order to avoid workflow failures, additional tasks that download and install the necessary software are executed

before the main tasks in the workflow. These modified tasks are represented with red rectangles on Figure 11.

Therefore, we can say that the Pegasus WMS implementation of *blast2cap3* for OSG is a slightly modified version of the implementation of *blast2cap3* for Campus Cluster.

Figure 11. Pegasus WMS implementation of *blast2cap3* for OSG, where the squares represent the input and output files, the ovals represent the tasks, the rectangles represent the tasks that has an additional step of downloading and installing the required libraries, and the arrows represent the dependencies between the tasks.

3.3. Results and Discussion

Here, our objective is to evaluate the performance of a scientific workflow for protein-guided assembly on a Campus Cluster and OSG. The experiments for this Chapter include creating and running a scientific workflow for *blast2cap3*, the protein-guided assembly. The workflow is run on two different execution platforms: Sandhills, the Campus Cluster, and the distributed Open Science Grid. Furthermore, the influence of the numbers of clusters of transcripts in *blast2cap3* over the execution time is also investigated and compared.

3.3.1. Performance evaluation

After the scientific workflow was created using Pegasus WMS, it was run on each platform multiple times with different values for "*n*". As mentioned previously, "*n*" determines the number of clusters of transcripts on which the input data, "*alignments.out*", is divided. For the purpose of this work, we used "*n*" with values of 10, 100, 300, and 500.

3.3.2. Comparing running time on Campus Cluster and OSG for different values of "n"

In order to compare the running time of the Pegasus WMS implementation of *blast2cap3*, we run the workflows when "*n*" is 10, 100, 300, and 500 respectively. After the workflow terminates, *pegasus-statistics* is used to generate general statistics for the workflow

execution. We use these statistics to compare the running time when *blast2cap3* is run serially and when is run as a scientific workflow with different values of "*n*".

The "*Workflow Wall Time*" statistic defines the total running time of the workflow from the start to its end. The comparison of this variable's value for the different workflows executed on the different platforms is shown on Figure 12.

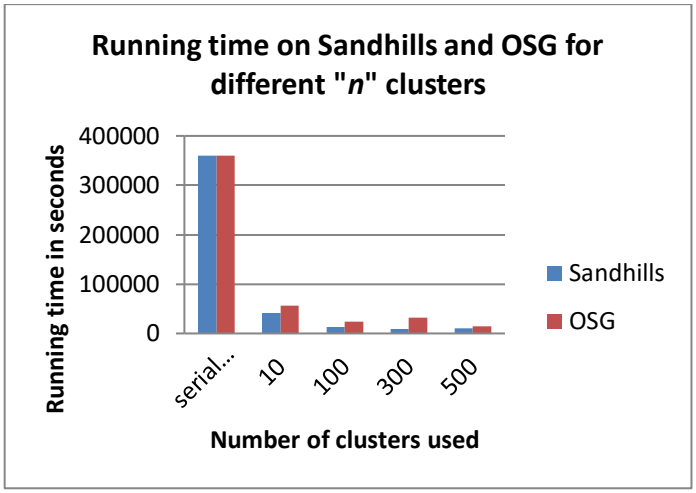

Figure 12. Comparing workflow running time on Sandhills and OSG when blast2cap3 is executed serially and as scientific workflow with "n" equals 10, 100, 300, and 500 respectively.

On Figure 12 we can notice that the Pegasus WMS implementation of *blast2cap3* significantly reduces the time execution for approximately more than 95%. If the current sequential implementation of *blast2cap3* for the given input files runs for 100 hours, the Pegasus WMS implementation runs for 3 hours in average.

Beside the difference between the serial and inherently parallel execution of *blast2cap3*, on Figure 12 we can also observe the difference in the running time on Campus Cluster and OSG platforms. Although OSG provides bigger variety of

computational resources than the Campus Cluster, for the experimental runs of our workflows, the Campus Cluster resulted in better running time. This difference is especially noticeable when "*n*", the number of clusters used, is 10, 100, and 300. Some possible reasons for this occurrence are the additional tasks required for setting the proper software configuration on the OSG resources, as well as the common failures and workflow retries that happen when OSG is used as a platform. OSG is an opportunistic platform, so some larger jobs can be held and resubmitted when the resources are available. On the other hand, we encountered no failures when the workflow was executed on Sandhills. The Campus Cluster may need a long waiting time to access nodes with more memory and time resources, but after these resources are allocated, they are utilized until the tasks terminate.

The running time on Sandhills when "*n*" is 10 is 41,593 seconds. On the other hand, when "*n*" has value of 100, 300, and 500, the running time on Sandhills is around 10,000 seconds. The usage of 100 or more clusters of transcripts improves the running time on Sandhills for approximately 80% compared to the running time of 10 clusters. Although the usage of more than 100 clusters doesn't decrease this running time significantly, the selection of 300 clusters gives the optimum performance with the resources allocated from Sandhills for this experiment. We must emphasize that the running time for both platforms and the optimal number of used clusters of transcripts will vary for every new run due to the availability of the current resources.

3.3.3. Comparing running time per task on Campus Cluster and OSG for different values of "n"

The running time of the submitted tasks and jobs varies among the two execution platforms and "*n*", the number of clusters of transcripts. Here, we analyze the running time of the individual tasks from the workflow, both for the Campus Cluster and OSG when "*n*" is 10, 100, 300, and 500. In order to achieve this, we use "*Kickstart Time*", "*Waiting Time*" and "*Download/Install Time*" statistics.

The "*Kickstart Time*" statistic defines the actual duration and running time of a job on the remote node. The "*Waiting Time*" statistic is a sum of the time spent waiting on the *submit host* and the time spent waiting on the remote host before the actual execution starts. The "*Download/Install Time*" statistic refers to the Pegasus WMS implementation of *blast2cap3* for OSG and indicates the time spent for downloading and installing the Python and Biopython libraries and CAP3 executable required for this experiment.

On Figure 13 the running times per tasks are shown for both Sandhills and OSG execution platforms when "*n*" is 10, 100, 300, and 500 respectively.

Figure 13. Comparing *blast2cap3* workflow running time per task for Sandhills and OSG when "*n*" is 10, 100, 300, and 500 respectively.

While the tasks for creating lists of the input files and for merging the final results have running time of few minutes, the higher consumption of time occurs when CAP3 is used for merging the transcripts within the clusters.

The "*Waiting Time*" value for the tasks ran on Sandhills is small and negligible. On the other hand, this value unevenly changes, increases and decreases, for the tasks ran on OSG. This observation once again shows that the resources available on OSG are opportunistic, and the OSG user cannot control the availability or the lack of resources over time. Unlike the Campus Cluster, failures and retries of the workflow were observed on OSG. This occurrence that is generally common and frequent on grids also increases the value of the "*Waiting Time*" statistic.

The "*Kickstart Time*" value per task on Sandhills slowly decreases when "*n*" increases. Higher values of "*n*" induce even more significantly greater reduction of the running time of the tasks ran on OSG.

However, the "*Download/Install Time*" value influences over the total running time of the tasks ran on OSG. Although some tasks on OSG have smaller running time than the tasks ran on Sandhills for the same value of "*n*", they still exceed the running time of the tasks on Sandhills. This happens because an additional time is required for the tasks on OSG to download and install the necessary libraries and executables on the OSG resources.

3.4. Conclusion

The expansion of scientific data leads to research that requires complex and data-intensive analyses and simulations. Therefore, many scientists use workflows over distributed resources to manage these large and complex computational tasks. Workflow applications can be used in different scientific fields, such as biology, physics, astronomy, and many others.

In this Chapter we built a scientific workflow for *blast2cap3*, the protein-guided assembly, using the Pegasus Workflow Management System (Pegasus WMS). Furthermore, we describe our experience deploying this workflow on two different computational platforms: the University of Nebraska Campus Cluster, and the distributed Open Science Grid (OSG). Our objective was to compare and evaluate the performance of the built scientific workflow for both used platforms. Furthermore, we wanted to show the importance of using scientific workflows for executing computationally demanding granular tasks and pipelines.

The performed experiments for this work show that the Pegasus WMS implementation of *blast2cap3* ran on both platforms significantly reduces the running time compared to the current serial implementation of *blast2cap3* for more than 95%. This high percentage shows the importance and the efficiency of using scientific workflows.

Beside the difference between the serial and parallel execution of *blast2cap3*, we also observed the difference in the running times on both Campus Cluster and OSG execution platforms. Although OSG provides bigger variety of computational resources than the Campus Cluster, for our experiments, the workflows that ran on the Campus Cluster

resulted in better running time. Moreover, the selection of 300 clusters of transcripts gives the optimum performance with the resources allocated from Sandhills for the completed experiment.

While the Campus Clusters support the most frequently used software tools, the OSG resources may have different software configuration. Therefore, the tasks on OSG used more running time than the tasks running on the Campus Cluster because of downloading and installing the required libraries and tools for *blast2cap3*. In addition, the availability of resources on OSG is highly variable and opportunistic, and therefore the performance and the running time of the tasks vary significantly. Workflows running on OSG may result with excellent or very poor throughput depending on whether there are plenty or a few available resources. In addition, workflow failures and retries were observed on OSG that also increase the running time. The OSG staff works hard to promptly and efficiently address these issues.

However, if comparing only the actual duration and running time of tasks on both platforms, ignoring the "*Waiting Time*" and the "*Download/Install Time*", OSG gives significantly better results. Hence, setting the proper software configuration on the OSG resources for less time will be considered as part of the future work.

Despite Campus Clusters and Grids, scientists are also investigating the use of Clouds for deploying scientific workflows. Using academic and commercial Clouds as an execution platform for the *blast2cap3* workflow built in this work will be challenging, but important and useful further step of this research.

Developing scientific workflows for applications from different scientific fields is a valuable and crucial step that connects complex and large granular tasks with thousands available powerful computational and distributed resources. The outcome of this process are automated complex analysis, real-time results and improved time performance that allow scientists to easily design, execute, modify, and re-run their experiments.

Acknowledgements

We would like to thank Dr. Adam Caprez, HPC Applications Specialist at the University of Nebraska Holland Computing Center, for his extensive help and useful suggestions during the preparation of the experiments for this work.

This work was completed utilizing the Holland Computing Center of the University of Nebraska, as well as using resources provided by the Open Science Grid, which is supported by the national Science Foundation and the U. S. Department of Energy's Office of Science. This research used the Pegasus Workflow Management Software funded by the National Science Foundation under grant #1664162.

Chapter 4

Conclusion

In this thesis, we focus on two main bioinformatics challenges when analyzing RNA-Seq data: 1) developing accurate transcriptome assemblies; and 2) efficient data handling (storage and processing). For each challenge, we performed comprehensive analyses and deliver conclusions and suggestions that can improve the quality of transcriptome assembly pipelines, as well as reduce some of their computationally-intensive requirements.

To provide insights in building accurate transcriptome assemblies, we generated 21 transcriptome assembly pipelines using different combinations of pre-processing and assembly methods, such as digital normalization, error correction, de novo assembly tools and *k*-mer length strategies. After a comprehensive evaluation of these assembly pipelines, we observed that using the error correction method with Velvet Oases and the "*multi-k*" strategy that combines the 5 *k*-mer assemblies with highest N50 value produces the best results when the transcripts are mapped against the raw sequences and the TriFLDB database.

To address the handling of data- and computationally-exhaustive transcriptome assembly pipelines, we selected *blast2cap3*, a serial and intensive step from the transcriptome assembly pipeline for protein-guided assembly. For this step, we built distributed and scalable scientific workflow and deployed it on two different computational platforms. This implementation for *blast2cap3* reduced the running time

by 95% and showed that scientific workflows that can be parallelized and executed on various computational platforms can significantly affect the runtime of the analyses.

Knowing the advantages and disadvantages of the tools used in each step of the assembly pipeline, performing comprehensive comparisons, as well as utilizing efficient scientific workflows and computational platforms, are essential steps in the direction of building accurate assemblies and answering important biological questions rapidly. With the analyses performed in this thesis, we provide useful guidelines for choosing good strategies for optimizing de novo transcriptome assembly pipelines.

Bibliography

[1] A. Gottlieb, S. G. Almasi, “Highly parallel computing,” Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1, 1989.

[2] D. Bader, R. Pennington, "Cluster Computing: Applications," Georgia Tech College of Computing. Retrieved 2007-07-13, June 1996.

[3] M. Bux, U. Leser, “Parallelization in Scientific Workflow Management Systems,” arXiv preprint arXiv:1303.7195 (2013).

[4] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers, “Examining the Challenges of Scientific Workflows,” Computer 40.12 (2007): 24-32.

[5] E. Deelman, G. Singha, M. Sua, J. Blythea, Y. Gila, C. Kesselmana, G. Mehtaa, K. Vahia, G. Berrimanb, J. Goodb, A. Laityb, J. Jacobc, D. Katzc, “Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems,” Scientific Programming Journal, Vol 13(3), pages 219-237, 2005.

[6] P. Couvares, T. Kosar, A. Roy, Jeff Weber and Kent Wenger, "Workflow in Condor", in In Workflows for e-Science, Editors: I.Taylor, E.Deelman, D.Gannon, M.Shields, Springer Press, January 2007 (ISBN: 1-84628-519-4).

[7] D. Thain, T. Tannenbaum, M. Livny, “Distributed Computing in Practice: The Condor Experience," Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[8] T. Oinn, M. Greenwood, M. Addis, M Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, C. Wroe, “Taverna: lessons in creating a workflow environment for the life sciences,” Concurrency Computat: Pract. Exper., 18: 1067–1100 (2006). doi: 10.1002/cpe.993.

[9] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, Y. Zhao, “Scientific workflow management and the Kepler system,” Concurrency Computat.: Pract. Exper., 18: 1039–1065, (2006). doi: 10.1002/cpe.994.

[10] S. Cohen-Boulakia, U. Leser, "Search, Adapt, and Reuse: The Future of Scientific Workflow," SIGMOD Record 40(2):6-16, 2011.

[11] Open Science Grid. [http://www.opensciencegrid.org/].

[12] Extreme Science and Engineering Discovery Environment (XSEDE). [http://www.xsede.org/].

[13] Amazon Elastic Compute Cloud. [http://aws.amazon.com/ec2/].

[14] FutureGrid. [http://futuregrid.org/].

[15] Y. Surget-Groba, J. Montoya-Burgos, "Optimization of de novo transcriptome assembly from next-generation sequencing data," Genome Res. 2010 Oct;20(10):1432-40. doi: 10.1101/gr.103846.109.

[16] H. Xiaoqiu, M. Anup, "CAP3: A DNA Sequence Assembly Program," Genome Res. 1999 September; 9(9): 868–877.

[17] Buffalo V: *Blast2cap3* software . [https://github.com/vsbuffalo/blast2cap3/].

[18] K. Krasileva, V. Buffalo, P. Bailey, S. Pearce, S. Ayling, F. Tabbita, M. Soria, S. Wang, IWGS Consortium, E. Akhunov, C. Uauy, J. Dubcovsky, "Separating homeologs by phasing in the tetraploid wheat transcriptome," Genome Biology 2013, 14:R66 doi:10.1186/gb-2013-14-6-r66.

[19] S. Altschul, W. Gish, W. Miller, E. Myers, D. Lipman, "Basic local alignment search tool, J Mol Biol 1990, 215:403-410.

[20] J. Yu, R. Buyya, "A taxonomy of scientific workflow systems for grid computing," SIGMOD Rec., vol. 34, pages 44–49, September 2005, 2, 13, 56.

[21] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," Journal of Grid Computing 2003, Volume 1, Issue 1, pp 25-39.

[22] Pegasus 4.3 User Guide. [https://pegasus.isi.edu/wms/docs/latest/pegasus-user-guide.pdf/].

[23] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, "Pegasus: Planning for Execution in Grids," GriPhyN technical report 20(17):12-22

[24] J. C. Good, J. C. Jacob, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, M. Su, and R. Williams, "Astronomical Image Mosaicking on a Grid: Initial Experiences," in Engineering the Grid: Status and Perspective, B. D.Martino, J. Dongarra, A. Hoisie, L. T. Yang, and H. Zima, Eds.: American Scientific Publishers, 2006.

[25] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman, "Workflow Management in GriPhyN," in Grid Resource Management: State of the Art and Future Trends, J. Nabrzyski, J. M. Schopf, and J. Weglarz, Eds.: Springer, 2003.

[26] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R.Graves, N. Gupta, V. Gupta, T. H. Jordan, C. Kesselman,P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi,and L. Zhao, "Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example," presented at Second IEEE International Conference on e-Science and Grid Computing, 2006.

[27] Sandhills UNL HPC Cluster. [http://hcc.unl.edu/sandhills/].

[28] M. Altunay, P. Avery, K. Blackburn, B. Bockelman, M. Ernst, D. Draser, R. Quick, R. Gardner, S. Goasguen, T. Levshina, M. Livny, J. McGee, D. Olson, R. Pordes, M. Potekhin, A. Rana, A. Roy, C. Sehgal, I. Sfiligoi, F. Wuerthwein, The Open Science Grid Executive Board, “A Science Driven Production Cyberinfrastructure - the Open Science Grid,” Journal of Grid Computing (Impact Factor: 1.6). 9(2):201-218. DOI:10.1007/s10723-010-9176-6 Source: dx.doi.org.

[29] NCBI BioProjects [http://www.ncbi.nlm.nih.gov/bioproject/?term=PRJNA191053/]/

[30] Python Programming Language. [http://www.python.org/].

[31] Biopython. [http://biopython.org/].

[32] R. Littauer, K. Ram, B. Ludascher, W. Michener, R. Koskela, “Trends in Use of Scientific Workflows: Insights from a Public Repository and Recommendations for Best Practice,” The International Journal of Digital Curation, Volume 7, Issue 2 | 2012.

[33] H. AlHakami, H. Aldabbas, T. Alwada'n, "Comparison Between Cloud and Grid Computing: Review Paper," International Journal on Cloud Computing: Services and Architecture (IJCCSA),Vol.2, No.4, August 2012.

[34] Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. Analysis of transcriptome assembly pipelines for wheat. In 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) 2016 Dec 15 (pp. 137-140). IEEE;

[35] Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. Comparing and optimizing transcriptome assembly pipeline for diploid wheat. In Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics 2014 Sep 20 (pp. 603-604);

[36] Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. Evaluating assembly pipeline for transcriptomes. In 6th International Conference on Bioinformatics and Computational Biology, BICOB 2014 2014 (pp. 163-168). International Society for Computers and Their Applications.

[37] Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. Evaluating distributed platforms for protein-guided scientific workflow. In Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment 2014 Jul 13 (pp. 1-8);

[38] Pavlovikj N, Begcy K, Behera S, Campbell M, Walia H, Deogun JS. A comparison of a campus cluster and open science grid platforms for protein-guided assembly using pegasus workflow management system. In 2014 IEEE International Parallel & Distributed Processing Symposium Workshops 2014 May 19 (pp. 546-555). IEEE.

[39] Ozsolak F, Milos PM. RNA sequencing: advances, challenges and opportunities. Nature reviews genetics. 2011 Feb;12(2):87-98.

[40] Zhao QY, Wang Y, Kong YM, Luo D, Li X, Hao P. Optimizing de novo transcriptome assembly from short-read RNA-Seq data: a comparative study. InBMC bioinformatics 2011 Dec (Vol. 12, No. 14, pp. 1-12). BioMed Central.

[41] Andrews S. FastQC: a quality control tool for high throughput sequence data.

[42] Assaf G, Hannon GJ. FASTX-toolkit. FASTX-Toolkit. 2010.

[43] Buffalo V. Using the qrqc package to gather information about sequence qualities.

[44] Martin M. Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet. journal. 2011 May 2;17(1):10-2.

[45] Buffalo V. Scythe-a Bayesian adapter trimmer. Website: http://github. com/vsbuffalo/scythe. 2011.

[46] Schmieder R, Lim YW, Rohwer F, Edwards R. TagCleaner: Identification and removal of tag sequences from genomic and metagenomic datasets. BMC bioinformatics. 2010 Dec;11(1):1-4.

[47] Joshi NA, Sickle FJ. a sliding-window, adaptive, quality-based trimming tool for FastQ files (Version 1.33)[Software]. 2011.

[48] Schmieder R, Edwards R. Quality control and pre-processing of metagenomic datasets. Bioinformatics. 2011 Mar 15;27(6):863-4.

[49] Crusoe M, Edvenson G, Fish J, Howe A, McDonald E, Nahum J, Nanlohy K, Ortiz-Zuazaga H, Pell J, Simpson J, Scott C. The khmer software package: enabling efficient sequence analysis. URL http://dx. doi. org/10.6084/m9. figshare. 2014;979190.

[50] Le HS, Schulz MH, McCauley BM, Hinman VF, Bar-Joseph Z. Probabilistic error correction for RNA sequencing. Nucleic acids research. 2013 May 1;41(10):e109-.

[51] Paszkiewicz K, Studholme DJ. De novo assembly of short sequence reads. Briefings in bioinformatics. 2010 Sep 1;11(5):457-72.

[52] Martin JA, Wang Z. Next-generation transcriptome assembly. Nature Reviews Genetics. 2011 Oct;12(10):671-82.

[53] Schulz MH, Zerbino DR, Vingron M, Birney E. Oases: robust de novo RNA-seq assembly across the dynamic range of expression levels. Bioinformatics. 2012 Apr 15;28(8):1086-92.

[54] Xie Y, Wu G, Tang J, Luo R, Patterson J, Liu S, Huang W, He G, Gu S, Li S, Zhou X. SOAPdenovo-Trans: de novo transcriptome assembly with short RNA-Seq reads. Bioinformatics. 2014 Jun 15;30(12):1660-6.

[55] Grabherr MG, Haas BJ, Yassour M, Levin JZ, Thompson DA, Amit I, Adiconis X, Fan L, Raychowdhury R, Zeng Q, Chen Z. Trinity: reconstructing a full-length transcriptome without a genome from RNA-Seq data. Nature biotechnology. 2011 Jul;29(7):644.

[56] Robertson G, Schein J, Chiu R, Corbett R, Field M, Jackman SD, Mungall K, Lee S, Okada HM, Qian JQ, Griffith M. De novo assembly and analysis of RNA-seq data. Nature methods. 2010 Nov;7(11):909-12.

[57] HCC. Holland Computing Center | Nebraska. Available from: https://hcc.unl.edu/.

[58] Kent WJ. BLAT—the BLAST-like alignment tool. Genome research. 2002 Apr 1;12(4):656-64.

[59] Fu L, Niu B, Zhu Z, Wu S, Li W. CD-HIT: accelerated for clustering the next-generation sequencing data. Bioinformatics. 2012 Dec 1;28(23):3150-2.

[60] Huang X, Madan A. CAP3: A DNA sequence assembly program. Genome research. 1999 Sep 1;9(9):868-77.

[61] Wicker T, Matthews DE, Keller B. TREP: a database for Triticeae repetitive elements.

[62] Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. Nature methods. 2012 Apr;9(4):357-9.

[63] Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R. The sequence alignment/map format and SAMtools. Bioinformatics. 2009 Aug 15;25(16):2078-9.

[64] Mochida K, Yoshida T, Sakurai T, Ogihara Y, Shinozaki K. TriFLDB: a database of clustered full-length coding sequences from Triticeae with applications to comparative grass genomics. Plant Physiology. 2009 Jul;150(3):1135-46.

[65] Behera S, Voshall A, Moriyama E. Plant transcriptome assembly: review and benchmarking. Exon Publications. 2021 Mar 20:109-30.

[66] Hölzer M, Marz M. De novo transcriptome assembly: A comprehensive cross-species comparison of short-read RNA-Seq assemblers. Gigascience. 2019 May;8(5):giz039.

[67] Lu B, Zeng Z, Shi T. Comparative study of de novo assembly and genome-guided assembly strategies for transcriptome reconstruction based on RNA-Seq. Science China Life Sciences. 2013 Feb;56(2):143-55.

[68] Liu Y, Khan SM, Wang J, Rynge M, Zhang Y, Zeng S, Chen S, Dos Santos JV, Valliyodan B, Calyam PP, Merchant N. PGen: large-scale genomic variations analysis workflow and browser in SoyKB. InBMC bioinformatics 2016 Oct (Vol. 17, No. 13, pp. 177-186). BioMed Central.

[69] Larsonneur E, Mercier J, Wiart N, Le Floch E, Delhomme O, Meyer V. Evaluating workflow management systems: A bioinformatics use case. In2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) 2018 Dec 3 (pp. 2773-2775). IEEE.