2023

TeloPortWrapper: A New Tool for Understanding the Dynamic World of Fungal Telomere Ends

Trey Stansfield
Western Kentucky University, treydenhaze104@gmail.com

TELOPORTWRAPPER: A NEW TOOL FOR UNDERSTANDING THE DYNAMIC WORLD OF FUNGAL TELOMERE ENDS

A Capstone Experience/Thesis Project Presented in Partial Fulfillment

of the Requirements for the Degree Bachelor of Biology

with Mahurin Honors College Graduate Distinction at

Western Kentucky University

By

Treyden H. Stansfield

May 2023

CE/T Committee:

Dr. Simran Banga, Chair

Dr. Claire Rinehart

Dr. Joseph Marquardt

ABSTRACT

Telomeres are repetitive DNA sequence motifs found at eukaryote chromosome ends. Telomeres help protect chromosome ends from DNA damage and promote chromosome stability. Chromosomes play important roles in aging, mutation, and cancer. Eukaryotic pathogens also use telomeres to mutate and manage virulence genes. In response to chromosome end breakage newly formed telomeres, called *de novo* telomeres, are formed to recreate the lost telomere and sub-telomeric regions. *Magnaporthe oryzae* is a fungal pathogen which causes wheat blast, a deadly plant disease in wheat. *Magnaporthe oryzae* is also known for its highly variable sub-regions which show high amounts of induced variability due to *de novo* telomere formation. This variance is associated with mutation and adaptation. Little is known about *de novo* telomere formation as telomeres are often underrepresented in standard sequencing assemblies. TeloPortWrapper is a new tool to collect and sort telomeric reads from the raw reads, identify and analyze *de novo* telomeres, and create visual result summaries. Using TeloPortWrapper on 940,225,828 reads across 14 different *Magnaporthe oryzae* strain genomes and aligning them to their assembled genome, it was found that a vast majority of new telomeric regions are being pulled from sections in the middle of chromosomes rather than close to the breakage point itself, as was previously assumed. Manually checking

all the aligned reads confirmed that TeloPortWrapper is an accurate tool for collecting and analyzing *de novo* telomeres.

I dedicate this project to my wonderful parents, Tristen and J.R Stansfield, who have always supported me, no matter what, and helped me grow into the person that I am today.

ACKNOWLEDGEMENTS

I would like to show support for all those who have helped me on this project. First, I would like to thank Seth Baunach who created the core TeloPort telomere finding and sorting program. Without him the rest of this project wouldn't have been possible. Secondly, I would like to thank Dr. Mark Farman at the University of Kentucky for overseeing this project and allowing me to continue his *Magnaporthe oryzae* research and Mostafa Rahnama for providing help on the project while I worked at the University of Kentucky. I would also like to thank Andrew Tapia and Jerzy Jaromczyk for reviewing the TeloPortWrapper code and providing input.

Thirdly, I would like to thank my three readers: Dr. Simran Banga, Dr. Claire Rinehart, and Dr. Joseph Marquardt. Thank you, Dr. Banga, for advising me during the project. Thank you, Dr. King and Dr. Rinehart for introducing me to bioinformatics and setting up my internship with Dr. Farman's team. Finally, thank you Dr. Marquardt for taking time out of your day to be my third reader. I would also like to thank Ms. Susann Davis for guiding me through the process of setting up my CE/T. I would also like to give a special mention to the Carol Martin Gatton Academy and the innumerable opportunities it has offered me.

Finally, I would like to thank my family who supported me through my college years. I'd like to end this section with one more thank you to anyone and everyone who helped me through this project during my final two college years.

VITA

EDUCATION

Western Kentucky University, Bowling Green, KY

B.S. In Biology (No Minor)-- Mahurin Honors College

Honors CE/T: *TeloPortWrapper: A New Tool for Understanding the Dynamic World of Fungal Telomere Ends* May 2023

The Carol Martin Gatton Academy of Mathematics and Science April 2021

PROFESSIONAL EXPERIENCE

Department of Plant Pathology, University of Kentucky June 2020-July 2020

Student Research Intern July 2022

AWARDS AND HONORS

J. R., Beulah, Theresa, and Carolyn Whitmer Scholarship Fund, WKU Fall 2022-Spring 2023

Gatton/Craft presidential Finalist, WKU Fall 2022-Spring 2023

Gatton Academy Graduates Research and Experiential Learning Award, WKU Fall 2022-Spring 2023

Magna Cum Laude, WKU, May 2023

RESEARCH PRESENTATIONS

Kentucky Academy of Science Annual Meeting

TeloReport: Extraction and Classification of Telomeric Raw Reads and Identification of Potential De novo Telomeres November 2020

TeloPort: a tool to investigate into the weird world of telomere ends November 2022

OTHER SCIENTIFIC APPEARENCES

Bench Talk, Louisville FORward Radio 106.5 FM. July 12th, 2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Telomeres are repetitive sequences of DNA found at chromosome ends. Unlike prokaryotes who have a circular chromosome, eukaryotes have multiple linear chromosomes with distinct repetitive end regions (Srinivas *et al.*, 2020). DNA replication begins when the double-strands of the DNA are unwound by Helicase into two single strands. One of these strands, the leading strand, is able to be replicated continuously in the 5' to 3' direction, while the other strand of DNA, called the lagging strand is replicated in discontinuous chunks called Okazaki fragments. Each one of these fragments requires an RNA primer to start DNA synthesis (Langston & O'Donnell, 2006). Afterwards, these RNA primers are removed, but at chromosome ends there is a slight single-stranded DNA overhang where the RNA primer is removed. These single-stranded DNA regions are trimmed off from the chromosome ends and therefore a small portion of DNA is lost during every round of replication. This process is known as the end-replication problem (Wynford-Thomas & Kipling, 1997). In 2001, researchers from the Tokyo Institute of Technology were able to demonstrate the end replication problem *in vitro* for the first time, proving that the lagging strand really did get smaller with each replication (Ohki *et al.*, 2001). Telomeres have emerged as a ubiquitous adaption in eukaryotes to solve this problem by acting as a source of renewable protection. Instead of important genes being gradually

lost as the lagging strand shrinks, only repetitive telomere sequence is lost. Telomeres can then be extended via telomerase (Srinivas *et al.*, 2020). Despite being ubiquitous in eukaryotic cells, telomeres vary in both sequence and function across groups of organisms and even individual species (Fulnecková *et al.*, 2013). The exact telomere repeat is usually preserved among groups of species such as TTAGGG/CCCTAA in vertebrates and fungi or TTTAGGG/CCCTAAA in plants, but certain groups and species have high diversity of telomere repeats and sequences. (Fulnecková *et al.*, 2013).

Telomeres can be divided into two parts the actual short repeat, such as TTAGGG, and the sub-telomeric region which is the sequence immediately adjacent to the repeat. The sub-telomeric region is filled with a variety of repetitive sequences separate from the conserved telomere repeat: like coding genes, pseudogenes, duplications, transpositions, and recombination targets. Sub-telomeric regions are high in diversity and activity and greatly vary in length between species. Due to the type of repetitive elements, sub-telomeric regions are highly unstable with frequent recombination and duplication events which make them ideal for quickly generating genetic diversity. This rapid adaptation may explain the transcriptional activity found in sub-telomeric regions (Kwapisz & Morillon, 2020). These unique features of the sub-telomeric region make ideal incubators for cells which need to undergo rapid adaptation like human immune cells and pathogens. Barry *et al.* (2003) investigated the disproportionate amount of parasite contingency genes in sub-telomeric regions, particularly in African

trypanosome, and theorized that the instability of the region could contribute to diversification and duplication of contingency genes, or that the region could be used for gene silencing. No direct evidence could be found for any of these theories beyond pathogens having disproportionately expanded sequences adjacent to telomeres. Diotti *et al.* (2021) preformed a review of various opportunistic fungal pathogens- *Aspergillus fumigatus*, *Candida albicans*, *Candida glabrata*, and *Pneumocystis jirovecii*- and the role the telomeric structure played in virulence of the species. They identified highly variable gene clusters, likely associated with secondary metabolites which are believed to play a role in virulence, within the telomeric regions of *A. fumigatus*. Genes associated with virulence, pathogenicity, adherence, and host specificity were also found in the listed species to be highly concentrated and variable in telomeric regions. The researchers also believed that telomeric regions could also be associated with gene silencing where pathogenic genes could be “turned off” by moving them to sub-telomeric regions, as a means to avoid triggering host immune responses. Another important contributor to sub-telomeric diversity are transposons. Transposons are genes which are mobile throughout the genome. There a variety of methods for moving genes into new spots such as excising and inserting themselves into new locations- as seen in DNA transposons, or using reverse transcriptase to reverse transcribe a RNA intermediary of a gene back into the genome. Transposons play a role in silencing genes via their insertion into, or adjacent to, genes or their control regions. This can cause

adaptation by altering genes or their expression, or destabilizing the genome enough to cause a double strand break and further modifications. Transposons also contribute to sub-telomeric region diversity via inserting themselves into the region (Pray, 2008). Telomeres are believed to have evolved from retrotransposons and it is believed transposons may play a vital role in the regulation of telomeres (Kordyukova *et al.*, 2018). In species which lack telomerases, or have it turned off experimentally, retrotransposons have been observed to serve as an elongation method. Both telomerase and retrotransposons function via reverse transcriptase but little is known about how these retrotransposons regulate themselves compared to telomerase (Biessmann & Mason, 2003).

Magnaporthe oryzae and its sister species *Magnaporthe grisea* are both plant pathogens. *M. oryzae* is responsible for rice blast, the deadliest disease of rice which can lead to total crop failure if unmanaged. *M. oryzae* and *M. grisea* were initially believed to be the same species as they are morphologically indistinguishable, but mating experiments and genetic differences between the two fungi revealed they were two separate species with *M. grisea* infecting crabgrass and *M. oryzae* infecting rice and other grasses. *M. oryzae* has a fully sequenced and accessible genome making it an ideal model organism for studying plant pathogens (Wilson & Talbot, 2009). In 1995 *M. oryzae* suddenly emerged with a new disease known as gray leaf spot disease which targets perennial rye grass. It was assumed that these new strains of *M. oryzae* were

recent and would therefore have low genome diversity, which they did. However, it was found that *M. oryzae* exhibited high levels of telomere variation, which signaled a high level of adaptability, since *M. oryzae* telomeres host genes that are key for pathogenicity. This discovery drew into question the conclusion that these strains were a recent evolution, but the elevated level of recombination events at the telomere also suggested that such genetic diversity may be able to arise in such a brief period due to telomere hypervariability. A key dichotomy is drawn between rice-infecting and perennial ryegrass-infecting *M. oryzae* strains. The rice-infecting strains were found to have limited telomere variation in the wild across varying locations. This suggests that many telomeric variations and recombinations are fatal and not viable. In contrast, perennial ryegrass-infecting strains of *M. oryzae* exhibited extreme levels of telomere variability across different wild samples even with few polymorphisms detected at internal loci (Farman & Kim, 2005). It was this research which first suggested that *M. oryzae*'s telomeres were unique enough to study in further research.

Rehmeyer C. *et al.* (2006) took a further look at the organization of all 14 chromosome ends in *M. oryzae*. Using fosmid inserts Rehmeyer C. *et al.* (2006) were able to identify and sequence all 14 chromosome ends and then used a local Basic Local Alignment Search Tool (BLAST) to map these to the genome assembly. A custom Perl script named TruMatch was developed to identify unique alignments from the BLAST (Li *et al.*, 2005a). CrossMatch(www.phrap.org) was used to screen out known transposable

elements and a Fgenesh algorithm(Softberry, www.softberry.com) was used to for gene prediction. The researchers found that only one of *M. oryzae*'s telomeres was directly connected to ribosomal RNA-encoding genes, only two were attached to unique chromosomal sequences, and the rest were connected to a sub-telomeric region with a telomere-linked RecQ-helicase gene. There were also only 4 predicted genes found disproportionately represented at chromosome ends, meaning, it is unlikely *M. oryzae* uses switching mechanisms to evade host defenses. Instead, it was believed *M. oryzae* telomeres have undergone frequent truncation events in order to remove terminally positioned virulence which trigger host defenses. This paper was important for not only identifying the chromosomes of *M. oryzae* but also being the most complete analysis of an organism's chromosomes at that point. Farman (2007) offered a minireview of these past two papers, and many others, which further hypothesized about the lack of telomere ends enriched with host specificity genes. He hypothesized that the high number of virulence genes at chromosome ends was simply a function of trying to maximize virulence gene diversity, since the rice host that, *M. oryzae* infects, does not have as dynamic of an immune system as humans, therefore, requiring less switching off of virulence genes to avoid detection by the host immune system.

Rahnama *et al.* (2020) looked at further methods of *M. oryzae* telomere destabilization. He found a key contributor to *M. oryzae* telomere instability are *Magnaporthe oryzae* Telomeric Retrotransposons, or MoTeRs. MoTeRs are

retrotransposons which are found commonly in *M. oryzae* and are believed to play a role in telomere destabilization. MoTeR1 codes for reverse transcriptase, while MoTeR2 seems degraded to the point of having no reverse transcriptase function and relies on MoTeR1 for movement. Shotgun cloning and MinION sequencing were used to generate and assemble the genome sequence for analysis. Ultimately this research affirmed that MoTeRs are frequently involved with telomere destabilization but not necessarily required for it. A total of 3 chromosome ends frequently recombined while lacking MoTeR sequence, it is believed that this is caused by the stress created from frequent rRNA copies being replicated in late S phase causing breakages repaired via *de novo* telomere insertion. Other unknown short tandem repeats also seemed to play similar roles in causing stress and breakage. It was also found that the presence of MoTeRs did not lead to considerable amounts of recombination in 4 cases. Ultimately MoTeRs and these other newly discovered mechanisms seemed to be ways of driving genetic diversity and adaption via forcing duplication or translocation events through chromosome breakage.

Rahnama *et al.* (2021) looked at methods for telomere maintenance and repair of failed telomeres. *M. oryzae* DNA was extracted from cultures and sequenced using Illumina sequencing. Then a knockout test was done on telomerase to see how *M. oryzae* reacted to telomere failure. A key discovery was that many *de novo* telomeres had sub-telomere regions which mapped directly to internal chromosomal sequences; showing that in the case of telomere

failure internal sequences were captured and duplicated to serve as *de novo* telomere sequence. Telomere repeat motifs and MoTeR sequences were also found internally within the genome suggesting that telomeres had become internalized following improper maintenance or failed *de novo* telomere repair events. It is further believed that such spontaneous telomere failure may be another adaptive strategy to further increase genetic diversity and adaptivity in and around telomeres.

One problem with studying fungal telomeres is that currently existing sequencing technics are often biased in terms of fungal telomere representation. Schwartz & Farman (2010) found that the commonly used shotgun sequencing technique is poor at representing telomeric and sub telomeric regions of DNA. DNA ends are naturally resistant to hydrodynamic shearing used in shotgun sequencing, which is the primary cause of the underrepresentation of telomeres in assemblies. This bias primarily affects organisms with short telomere tracts so insects, vertebrates, plants, and other organisms with telomeres greater than 10 kb in length are not affected. The total size of telomeric gaps resulting from shearing bias may be small, but fungal sub-telomeric regions are often highly enriched and biologically interesting. Meaning such a gap leads to potentially relevant biological data being lost. Attempts to remedy this problem have been proposed in programs such as TERMINUS. TERMINUS is a set of three Perl scripts who mine raw read data from telomere-like sequence, screens the telomere-like sequence for low quality via PHRED scores, and then queries them

against the assembled genome using a local BLAST. TERMINUS was a useful tool to study the link between telomeric reads and genome assemblies, but due to its age it became deprecated over time and unable to run on modern operating systems (Li *et al.*, 2005b).

This research will focus on creating a modern tool which can extract, sort, and analyze fungal *de novo* telomeres in order to further understand the formation of *de novo* telomeres in *M. oryzae*, and discover where internal sequences are being pulled from to create *de novo* telomeres.

MATERIALS & METHODS

TeloPortWrapper Version 6

There are currently six major versions of TeloPortWrapper. Version 6 was the one used for testing on this project. For information on the development of TeloPortWrapper see Appendix A: TeloPortWrapper Version History.

Operating System Compatibility

TeloPortWrapper was run on a virtual machine of WKU's telofun supercomputer partition running Oracle Linux. TeloPortWrapper is currently only compatible with Linux based operating systems with BASH support.

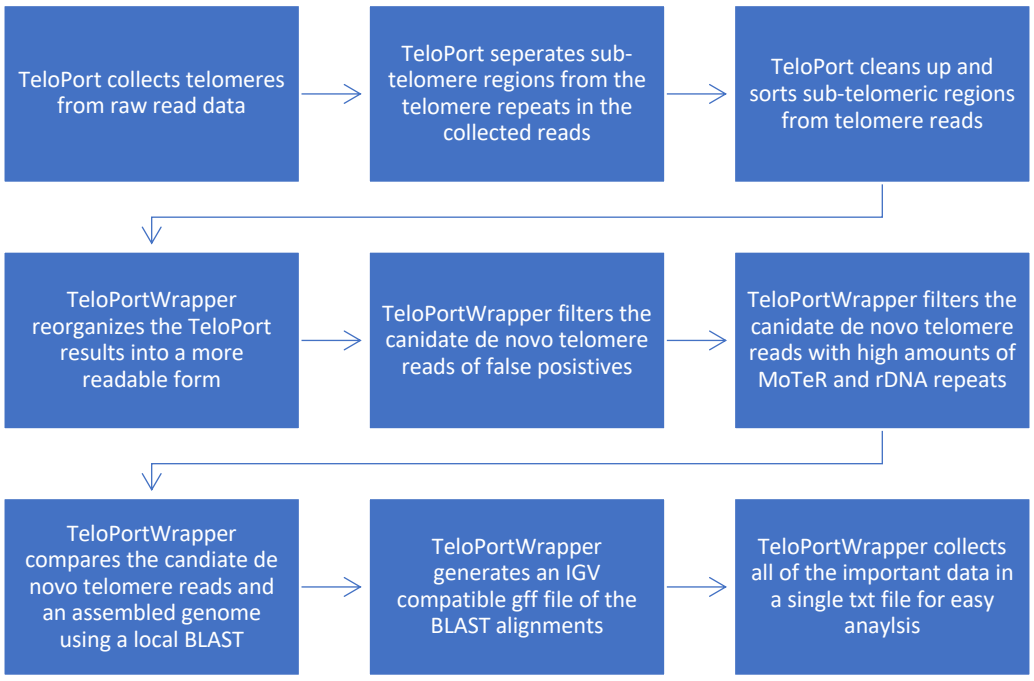
The TeloPortWrapper Workflow Summarized

TeloPortWrapper is a program divided into two parts: a C++ telomere finding program, called TeloPort, that was created by Seth Baunach while at University of Kentucky and a python-based wrapper script for TeloPort data analysis that wascreated by Trey Stansfield. TeloPort is divided into four separate programs which find all telomere reads in the raw reads, identify the telomere/sub-telomere junction, filter out the low-quality reads, and sorts the sub-telomere reads into clusters based on sequence similarity. Any reads which do not match with a cluster or exist in a cluster of less than 5 reads are considered candidates for being a *de novo* telomere. TeloPortWrapper first reorganizes and renames the TeloPort output for clarity purposes

before checking all candidate *de novo* telomere reads for false positives. The candidate *de novo* telomere reads are further filtered to remove sub-telomeric reads which contain rDNA, MoTeRs, or MoRepeats, so only *de novo* telomere reads taken from internal sequence are examined. The remaining candidates are compared to the assembled genome via a local BLAST and a GFF file is created from the alignments to visualize them using Integrative Genomics Viewer (IGV). TeloPortWrapper ends by creating a summary results file containing the BLAST alignments and other summary data (Fig. 1).

Figure 1

TeloPortWrapper Workflow Flow Chart



Note: IGV stands for Integrative Genomics Viewer, BLAST stands for Basic Local Alignment Search Tool.

TeloPort Pipeline

TeloPort consists of 4 separate C++ programs: telomereFinder, junctionFinder, sequenceQuality, and wcdInterrogate. TeloPort also uses a pre-existing program, WCD Express 6.3- created by Hazlehurst & Lipták, 2011. This program is used to sort the telomere reads. TelomereFinder finds and collects the full telomere reads from the raw read data containing the telomere repeats and joined sub-telomere regions. TelomereFinder works with separated fastq files or interleaved fastq files. JunctionFinder identifies the junction between the sub-telomere region and the telomeric repeats. SequenceQuality removes sequences which are too short and can cut off the ends of sub-telomere sequences where sequence quality drops off. WCD Express sorts the filtered and cut sub-telomere reads into clusters of similarity. WcdInterrogate collects all the reads and puts them in a file structure based on the clustering. Each cluster is turned into an individual file with all of its reads being copied into it. All four of these programs are separate, but TeloPortWrapper places them all in a pipeline where the output for one is automatically directed into the next program.

Unlike past versions of TeloPort, which used a strict telomere repeat match via GREP, this current version uses a “fuzzy matching” scoring algorithm which better captures reverse direction reads and can identify telomere repeats that have single nucleotide changes found in lower quality read data. This algorithm is faster and more accurate than the previous GREP version.

TeloPortWrapper Pipeline

TeloPortWrapper functions as a python-based wrapper which automatically feeds input data into the TeloPort pipeline and further analyzes it before summarizing the important results. The full code of TeloPortWrapper can be found in Appendix B. TeloPortWrapper starts by running telomereFinder and tagging every single read found by telomereFinder with a unique ID number in the fastq header, this ID number is maintained throughout the entire pipeline allowing each read to be manually traced. The rest of TeloPort proceeds to run as described above. TeloPortWrapper then checks every cluster folder of less than five reads and renames them as “single_reads” denoting them as candidate *de novo* telomeres. This five read cut off point can be changed by the user.

The next step is filtering out false positives from the candidate *de novo* telomere reads by checking if any candidate reads match any clusters via BLAST. MUSCLE 3.8(https://drive5.com/muscle/downloads_v3.htm) is used make a multi-fasta alignment from each cluster. The consensus feature of EMBOSS 6.6.0(https://emboss.sourceforge.net/download/) is then used to make a consensus fasta for each cluster. The consensus sequences are compared to the candidate *de novo* telomere reads via a local nucleotide BLAST, and any sequences which produce a hit are moved from the “single_reads” folder to the respective cluster they matched to. Using these updated cluster sizes, an ascii histogram is built to show the relative size of each cluster (fig. 2).

Figure 2

Sample Cluster Histogram Taken from Genome ERR2061619

cluster0 75

##

cluster1 20 ####################

cluster2 10 ##########

cluster3 12 ############

cluster4 8 ########

cluster5 7 #######

Notes: Each hash symbol corresponds to one read in the cluster.

The next round of filtering removes candidate *de novo* telomere reads which have MoTeRs, rDNA, and other moRepeats included in their sub-telomere region. These are still likely *de novo* telomeres, but for the purposes of this research question they are not important. This leaves only candidate *de novo* telomere reads whose sub-telomeric region is taken from internal sequence. The assembled genome is then analyzed to create a dictionary of every contig that has telomere ends. A similar dictionary is made for contigs that contain MoTeRs. The filtered *de novo* telomere reass are then compared to the assembled genome via a nucleotide BLAST and the alignment is output into in a custom version of out format 6 (table 1). The qseqid refers to the filtered *de novo* telomere's unique ID header. The sseqid refers to the chromosome that the *de novo* matched to. Pident refers to the percent match between the alignment. Length refers to the length of the alignment. Mismatch refers to the number of mismatches between the alignment. Gapopen refers to the number of gaps

between the alignment. Qstart and qend refer to the start and end points of the alignment on the *de novo* telomere read. Sstart and send refer to the start and end points of the alignment on the chromosome. Evalue refers to the expect value for the alignment. Slen refers to the total length of the chromosome.

Table 1

Sample BLAST output format from ERR2061611

qseqid	sseqid	pident	length	mismatch	gapopen	qstart	qend	sstart	send	evalue	slen
3692	Chr3	100	74	0	0	1	74	4325922	4325849	9.49e-33	8237918
3696	Chr3	100	43	0	0	12	54	4325849	4325891	8.50e-16	8237918
5448	Chr4	100	57	0	0	17	73	1973334	1973390	2.17e-23	5413369

Notes: This is actual data from the ERR2061611 results but is intended to show off how the data will look when running any genomes.

These BLAST reports are then automatically analyzed, and the results are appended to the end of each BLAST report line (table 2). The chromosome contig that the *de novo* telomere read was aligned to is checked against the dictionary of telomere-containing contigs to determine if the *de novo* telomere read aligned with a telomere-containing contig. If the *de novo* telomere read aligns with a telomere-containing contig, the program determines how far the alignment is from the telomere on the contig. If there are two telomeres on the contig it outputs the distance from the closest telomere. The final piece of information collected is whether the *de novo* telomere read aligns with the reverse strand of the chromosome.

Table 2

Sample BLAST interrogation format from ERR2061611

qseqid	Tels	Tele	Telomere distance	MoTer	Rev
3692	TelsY	TeleY	3911932	MoterY	revT
3696	TelsY	TeleY	3912005	MoterY	revF
5448	TelsY	TeleY	1973151	MoterY	revF

Notes: This is actual data from the ERR2061611 results but is intended to show off how the data will look when running any genomes. Qseqid refers to the query ID. Tels refers to if the contig has a telomere at the start, Y refers to yes and N refers to no. Tele refers to if the contig has a telomere at the end, Y refers to yes and N refers to no. MoTer refers to if the contig has MoTeRs, Y refers to yes and N refers to no. Rev refers to if the *de novo* is aligning to the reverse contig, T refers to yes and F refers to no.

An IGV compatible GFF file is then generated from the analyzed BLAST output to allow for a visual representation (fig. 3). At the end, numerous summary statistics are calculated: the number of reads processed, the number of telomeres found, the number of candidate *de novo* telomeres found, the number of clusters identified, the number of filtered *de novo* telomere reads, and the number of filtered *de novo* telomere reads which matched to the assembled genome. The cluster histogram and the interrogated BLAST file are also appended to the results summary for easy access.

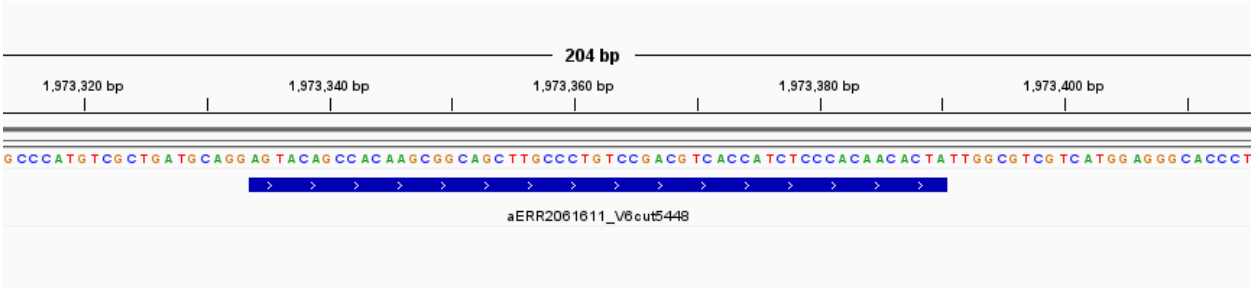
Datasets Used

Data analysis was performed on isolates of the B71 strain of *M. oryzae*. The raw read data from B71 isolates ERR2061611 to ERR2061624 were taken

from the European Nucleotide Archive FTP server. The B71 Illumina assembly file was provided by Dr. Mark Farman from the Plant Pathology department at University of Kentucky. Data for rDNA, MoTeRs, and MoRepeats was also provided by Dr. Mark Farman.

Figure 3

Sample IGV alignment output from ERR2061611



Notes: This is a zoomed in picture of how a single *de novo* telomere read will look aligned against the genome. The telomere repeat is cut off so only the sub-telomeric region is aligned to the genome.

Testing Methods

The entire TeloPortWrapper pipeline was run on B71 strains ERR2061611 to ERR2061624 and compared against the B71 Illumina assembly. The summary data was collected from all 13 data sets and compiled. Then every single filtered *de novo* telomere read that aligned with the genome was manually checked to ensure: that it was a telomere read, that the entire sub-telomere region aligned to the genome, that the telomere and sub-telomere regions were properly separated.

RESULTS & DISCUSSION

Across all 14 data sets a total of 540 filtered *de novo* telomere reads were found from a total of 33,644 telomeres extracted from 940,225,828 reads. Of these filtered *de novo* telomere reads, 88 matched to the assembled genome only a single time while they others matched to the assembled genome multiple times. Every single data set aligned with at least one MoTeR sequence with an average of 30.429 MoTeR alignments per dataset, and ERR2061613 having the largest number of MoTeR alignments at 116. The sub-telomeric sections of all of the telomere reads were sorted into clusters based on sequence similarity. The total cluster count varied between individual datasets from a minimum of 2 clusters to a maximum of 8. The most common cluster count was 2 across 6 of the 14 datasets. Eight and 6 were the second most common cluster count with there being a total of 2 datasets for each. Individual cluster size followed a pattern of the first cluster being over two times the size of all other clusters in the data set with each subsequent cluster getting progressively smaller at a less drastic rate (table 3).

An additional 18 filtered *de novo*s were found to match to the genome in multiple spots near the front of the chromosomes in a repetitive pattern. When visualized in IGV these hit patterns resembled those of rDNA (fig. 4). Meaning it is likely these are *de novo*s with rDNA sequence in the sub-telomere region that

failed to be filtered out via TeloPortWrapper. Additional testing on the individual reads found they could not make any alignment with the rDNA file. It is assumed that some of the ERR datasets have diverged from the B71 strain more than was initially predicted, meaning the strains have developed some unique rDNA, which is not caught by the filtering system based on the B71 strain. It should be noted that some datasets did not have any false positives get through the filters and all 14 genomes had some rDNA containing reads get filtered out via TeloPortWrapper implying this is a recent development.

Table 3

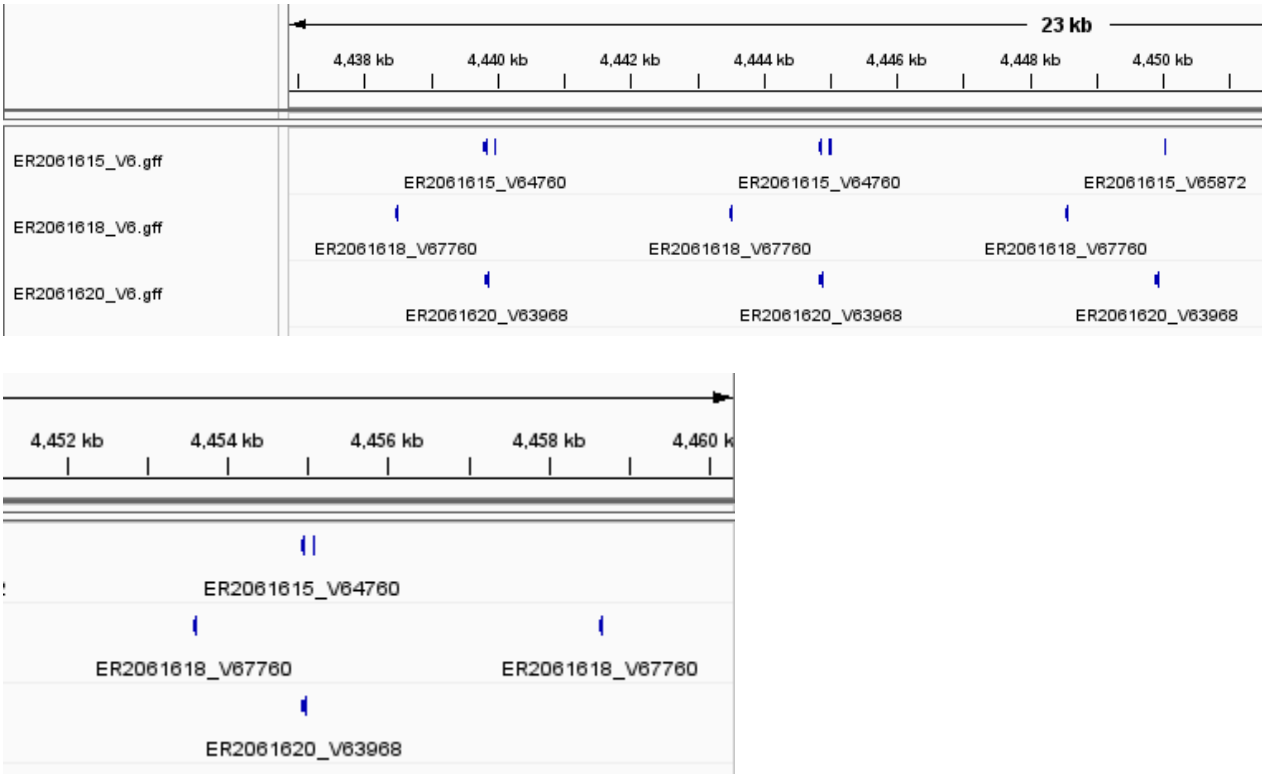
TeloPortWrapper Summary Table for Datasets ERR2061611 to ERR2061624

Dataset used	# of reads processed	# of telomere reads found	# of clusters	# of filtered *de novo* telomere reads	# of *de novo*s telomere reads that align with the genome	# of single *de novo* matches	# of *de novo* telomere read alignments with the MoTeRs
ERR2061611	65672682	2973	2	54	10	10	31
ERR2061612	47877132	1892	2	31	4	4	55
ERR2061613	75635046	2436	2	60	9	7	116
ERR2061614	81873758	2642	2	48	4	4	39
ERR2061615	49045367	2568	8	13	2	0	14
ERR2061616	78506911	2614	2	56	16	15	36
ERR2061617	43781229	1806	5	17	3	2	12
ERR2061618	59688642	2458	8	39	7	5	27
ERR2061619	59825422	2054	6	44	18	18	16
ERR2061620	90192512	3318	7	51	2	0	2
ERR2061621	80058680	2259	4	68	16	14	22
ERR2061622	52113073	2013	6	39	11	9	15
ERR2061623	43605884	1536	2	6	1	0	3
ERR2061624	112349490	3075	5	14	3	0	38

Notes: # of single *de novo* matches refers to the number of filtered *de novo* telomere reads which matched to the genome at only one location.

Figure 4

Sample "rDNA like" Alignment Pattern on Chromosome 5



Notes: In dataset ERR2061615 read ID 5872 is next to read ID 4760 but unlabeled by IGV.

Manually checking all 88 single *de novo* alignments confirmed that the TeloPort junctionFinder accurately found the telomere/sub-telomere junction and split it every single time (table 4). This means that every single alignment to the genome contained nearly the full sub-telomere region split off by junctionFinder with the telomere repeats being added directly to the beginning or end of the internal sequence. This further illuminates the process of forming *de novo* telomeres, as it shows that *de novo* telomere reads made from internal sequence are taken wholesale from internal sequence with no "buffer sequence" being added between the internal sequence and the telomere repeats or at the end of

the internal sequence. A full version of table 3 with all 106 of the *de novo* telomere reads which matched to the genome can be found in appendix C.

The average distance of the 88 internal sequences from the chromosome's telomere was 2,015,634.742 bp. The read that was closest to the telomere was 163,951 bp away from dataset ERR2061616 read ID 4388. The furthest telomere from the chromosome ends was 7,915,472 bp away from dataset ERR2061611 read ID 5500 (table 4). This shows that the internal sequences were gathered from a variety of spots across the chromosome with not even one being right next to the telomere end. This confirms earlier research which suggested that internal sequences were being captured for *de novo* telomeres, but suggests these sequences are coming from deeper in the genome than originally thought. This further begs the question of why these sequences are chosen and why the sequences closest to the telomere ends are not copied.

The high number of *de novo* telomere reads constructed from internal sequences, found within only 10 out of the 14 individual datasets, suggest that there is rapid chromosomal end breakage causing the formation of new telomeres made by wholesale copying further upstream gene sequences. Perhaps these internal sequences are an innately risky method of telomere repair only used in dire situations, which has the side effect of driving telomere diversity. The genetic data here was extracted from a culture of fungi with no ability to tell which DNA was coming from alive or dead fungi, so the question of the survivability of this repair mechanism is unanswerable with the current data.

Table 4

Candidate De novo BLAST Alignment Summary

Genome	De-novo telomere ID	Contig Hit	distance from telomere (contig end)	Telomere Junction
ERR2061611	3692	Chr3	3911932	**CAGCTCCTGGTTG** ATGGGTTCGGGTTAGGG
ERR2061611	3696	Chr3	3912005	CCCTAACCCGAACCCAT **CAACCAGGAGCTGTAGT**
ERR2061611	9796	Chr5	962965	**TACTAGTGGGTTCGG** GTTAGGGTTAGGG
ERR2061611	6616	Chr2	1393405	**GCTGATCGAAAAGGCACGG** TTAGGGTTAGGG
ERR2061611	6620	Chr2	1393369	CCCTAACCCTAACCCTAA **CCGTGCCTTT**
ERR2061611	5500	Chr2	7915472	**GACGCCAGACGC** ATTAGGGTTAGGG
ERR2061611	10948	Chr6	6133529	**CGCCGATGGTAG** GGTTAGGG
ERR2061611	5448	Chr4	1973151	CCCAAACCCGAAGGGT **AGTACAGCCA**
ERR2061611	4040	Chr3	1039395	**ATTCTTTTGG** TTAGGGTTAGGG
ERR2061611	6204	Chr4	1912175	CCCTAACCCTAT **GTTTGCTTAT**
ERR2061612	6524	Chr2	984942	**TCCGTTTA** AGGGTTAGGG
ERR2061612	6780	Chr7	935235	CCCTAACCCTAA **TCCCAATAAC**
ERR2061612	6784	Chr7	935142	**TTTTGTTATTGGGA** TTAGGGTTAGG
ERR2061612	7084	Chr 2	1067342	**CAGCGCGG** TTTGGGTTTGGG
ERR2061613	8552	Chr 7	1825871	**CAGGTGTTGTG** TTAGGGTTAGGG
ERR2061613	8548	Chr 7	1825797	CCCTAACCCTAACA **CAACACCTGCAA**
ERR2061613	3288	Chr6	1966404	**GCCGTCG** TAGGGTTAGGG
ERR2061613	8656	Chr4	1073968	**TAAAGTCGCG** TAGGGTTAGGGTTAGGG
ERR2061613	4720	Chr 8 mini	172434 and 78897	**TTCCAAATA** TTAGGGTTAGGGTTAGGG
ERR2061613	2356	Chr 9 mini	0 (no Telomere found on contig)	**TTAAAAAACA** GGGGGAACCCTTCGGGTTTGGG

ERR2061613	5584	Chr 8 mini	688504	**GGATTCTCA** GCCCTTCGGGTTTGGGTTTGGG
ERR2061614	6272	Chr 5	1845099	**TTTGTTTGA** TGGGTTTGGGTTTGGG
ERR2061614	9672	Chr 1	1439649	**CCCCGGCTACG** TAGGGTTAGGG
ERR2061614	252	Chr 1	2078176	**TTTCTTGG** TTAGGGTTAGGG
ERR2061614	9212	Chr 7	1048190	**ATGTTTTGA** TAGGGTTAGGG
ERR2061616	9324	Chr 6	3063180	**AGTGTCGT** AGTTAGGGTTAGGG
ERR2061616	9328	Chr 6	3063180	CCCTAACCCTAACT **ACGACACTCAT**
ERR2061616	9864	Chr4	2610572	CCCAAACCCGAAGG **CCCAGAATTAATAAA**
ERR2061616	9868	Chr 4	2610528	**TAATTCTGGG** CCTTCGGGTTTGGGTTTGGG
ERR2061616	3512	Chr 6	1620122	**ACTACAATTG** TGTTAGGGTTAGGG
ERR2061616	3508	Chr6	1620212	CCCTAACCCTAACA **CAATTGTAGTCGCCA**
ERR2061616	9552	Chr7	1966945	**ACCAAGGGCTTCCAG** GATGTTGAGGGTTAGGG
ERR2061616	4928	Chr 4	1062688	CCCTAACCCTAACCCTA **CGGACCCGCT**
ERR2061616	4932	Chr4	1062761	**GAGAGCGGGTCCG** TAGGGTTAGGGTTAGGG
ERR2061616	5504	Chr1	3157449	CCCTAACCCTAA **CCTTCCCAATCAG**
ERR2061616	5500	Chr1	3157502	**TGATTGGGAAGG** TTAGGGTTAGGG
ERR2061616	4432	Chr2	243697	**CCCAGGTCAGGTTGGG** CTTCGGGTTTGGGTTT
ERR2061616	2004	Chr 2	1282533	**TCGAATTGGTTT** GCCCTTCGGGTTTGGG
ERR2061616	5476	Chr2	276912	**TGGGTGGCGTGCCTG** GGTAGGGTTAGGGTTAGGG
ERR2061616	4388	Chr2	163951	**GAGTCACGGCTTTG** TTTGGGTTCGGGTTAGGG
ERR2061617	4032	Chr6	225967	**TTTGAGC** GCTACTTTCGGCGTTAGGGTTAGGG
ERR2061617	6068	Chr6	1369166	**CCCCATTCGAG** GATAGGGTTAGGGTTAGGG
ERR2061618	7204	Chr 2	3746746	**TTGCCTCTT** TTAGGGTTAGGG
ERR2061618	3276	Chr5	1706536	**TCATGAACG** AAGGCGTAGGGTTAGGG
ERR2061618	4476	Chr 4	2470349	**AATTCGACGTTCGG** GTTAGGGTTAGGG
ERR2061618	6820	Chr6	1797063	**CCTCATCAT** TAGGGTTAGGG

ERR2061618	5024	Chr2	3087198	**TGCTGTCCC** GGTAGGGTTAGGG
ERR2061619	6312	Chr1	2568138	**AATCCGAGAAG** TAGGGTTAGGGTTAGGG
ERR2061619	6308	Chr 1	2568053	TAACCCTAACCCTAC **TTCTCGGATTT**
ERR2061619	7376	Chr6	2092197	TAACCCTAACCCTAA **CTTATATGCCCGGCCCCA**
ERR2061619	7380	Chr6	2092245	**CATATAAG** TTAGGGTTAGGG
ERR2061619	2512	Chr 4	1347909	**TCGCCAACCTTG** TTTGGGTTCGGGTTAGGG
ERR2061619	2508	Chr 6	1619772	**AACCTTTTG** TTAGGGTTAGG
ERR2061619	3784	Chr3	3953235	**CGGTAAGGTA** TTAGGGTTAGGGTTAGGG
ERR2061619	3788	Chr3	3953285	TAACCCTAACCCTAA **TACCTTACCGGCC**
ERR2061619	7808	Chr2	1477118	**ATAGCCAG** CGGGTTAGGGTTAGGG
ERR2061619	2824	Chr5	1592553	**GAGCTGTTTG** TTAGGGTTAGGG
ERR2061619	5048	Chr5	612930	**CAGCCCCAG** CTTGCTGGTTAGGGTTAGGG
ERR2061619	1988	Chr2	208492	**TTCCGCCCACC** TTAGGGTTAGGG
ERR2061619	2928	Chr1	1765939	**TTTTAGGTTGG** CGGGTTAGGGTTAGGG
ERR2061619	4676	Chr4	175140	**TCGTGGGCGGCGGGG** TTAGGGTTAGGG
ERR2061619	5396	Chr5	1990297	**CCTGCCGGT** AGACGCGGCGGG
ERR2061619	6532	Chr6	2870291	**TTTATCGGGCG** GTTGGGTTTGGG
ERR2061619	2480	Chr3	1580710	TAACCCTTAGCCCAG **CCCCCCCTCAAACTTGTTTTT**
ERR2061619	5800	Chr6	633380	**TTCCAGAGGATGTCG** GGGTTAGGGTTAGGG
ERR2061621	4072	Chr4	2023082	**TGGTCCCATGGGTG** TTGGGTTTGGG
ERR2061621	4068	Chr4	2023001	CCCAAACCCAA **CACCCATGGGACCATTACTG**
ERR2061621	3532	Chr2	1417738	**GATTCATACCTCGGGG** CAGGGTTAGGGTTAGGG
ERR2061621	3536	Chr2	1417827	CCTAACCCTAAC **CCTGCCCCGAGGTAT**
ERR2061621	2640	Chr4	5413369	CCCTAACCCTAA **CCGAACAGCTAG**
ERR2061621	2644	Chr4	5413369	**CAGCTAGCTGTTCGG** TTAGGGTTAGGG
ERR2061621	5096	Chr2	3540522	**AAGATTCCTC** AATAGGGTTAGGGTTAGGG

ERR2061621	36	Chr1	1305518	CCCAAACCCAA **CCACCCACCCACACATCCTGAC**
ERR2061621	6552	Chr6	551086	**CCCTGGATCACCTACTC** AGGGTTAGGGTTAGGG
ERR2061621	6200	Chr3	3321915	**TTGAGTTGAGCGCG** TAGGGTTAGG
ERR2061621	7548	Chr6	1676791	**ACCCAACTTGG** TTGGGTTTGGG
ERR2061621	8388	Chr6	275244	**TGCGCTGGCAGG** TTAGGGTTAGGG
ERR2061621	8520	Chr 7	1762003	**GGGCATCAAAGC** TTCGGGTTTGGGTTTGGG
ERR2061621	5908	Chr4	266051	**AAGTTAGGCCAG** TTAGGGTTAGG
ERR2061622	5364	Chr 1	4450103	CCCTAACCCTAACCCT **CTCCGCACTTAACCGACC**
ERR2061622	5368	Chr 1	4450048	**TTAAGTGCGGAG** AGGGTTAGGGTTAGGG
ERR2061622	7176	Chr 7	1227339	CCCTAACCCTAACCCG **ACCGACAAGTAGCAG**
ERR2061622	4888	Chr5	1165853	**GTCTCTACTGGG** CCACCACGCTGGCAAGGG
ERR2061622	2380	Chr3	2178893	**CAGGGACGAGGTG** TTAGGGTTAGGG
ERR2061622	7560	Chr2	998998	**GCCACACTCACCGAGG** TAGGGTTAGGG
ERR2061622	2248	Chr3	3866700	**CCAAGAATGCTTTGCG** TCGGGTTAGGGTTAGGG
ERR2061622	7824	Chr9	0	**TAATAAAATTACAGTATTAA** TTAGGGTTAGGG
ERR2061622	2304	Chr3	2353073	**CCCTGCAGCACTTGGAGGAG** TAGGGTTAGGG

Notes: The bolded sequence in the 5th column is refers to the sub-telomere region and the non-bolded refers to the telomere repeats.

CONCLUSION

Studying fungal genomes is a challenging task due to the limitations of existing assembly tools. TeloPortWrapper is a promising first step in acquiring more of the telomere regions for analysis. During this research 14 *M. oryzae* strains, ERR2061611 to ERR2061624, derivative of *M. oryzae* strain B71 were analyzed by TeloPortWrapper to further understand the formation of *de novo* formed telomeres. The effectiveness of TeloPortWrapper was also tested via manually confirming each result by comparing it to both the assembled B71 genome and the dataset raw telomere reads. This research confirmed that TeloPortWrapper accurately collects, splits, sorts, and aligns telomere reads to the assembled genome. The only known flaw is the improper filtering of *de novo* telomere reads containing rDNA, which was an issue with some of the datasets used, which have more diversity in the rDNA than initially assumed. This issue can be solved by using more precise rDNA files for each genome. Additionally, TeloPortWrapper will need to undergo combability rewrites in order to be able to run on more computer types like Windows or non-BASH based Linux computers before it can be fully published. Ideally, TeloPort will have its finding algorithm expanded so it could search for sequences defined by the user. Using this a researcher could input the telomere for a different species and have TeloPort search for it, or even search for a different type of genetic repeat. The rest of the

pipeline is fully compatible for user defined files, so these sequences could be compared to a genome assembly of the user's choosing. Other features like allowing the user to choose what, if any, sequences will be filtered from the final BLAST would further help in diversifying the use case of TeloPort. This would greatly expand the scope of TeloPortWrapper and allow it to be used in a greater variety of research applications and problems. For more information on the future of TeloPortWrapper check Appendix D.

The dataset studied also added information about the process of *de novo* telomere formation in *M. oryzae.* Not only were there more *de novo* telomeres created from internal genome sequence than expected, but these internal sequences were located at least 160,000 bp away from the telomere of the chromosome. The deep average read length shows that these sequences are not being taken from areas adjacent to the telomere region, but from deeper areas of the chromosome. There was also zero overlap between any of the single match reads suggesting that there are not exact areas or sequences chosen to make *de novo* telomeres. This aligns with Rahnama *et al.* (2021) which found many internal sequences reaching into the millions of bp away from the chromosomal end. Rahnama *et al.* (2021) also found many internal sequences being found right at the chromosomal end, which this research was not able to find. Previous research on other species, like *Saccharomyces cerevisiae*, has suggested that internal sequences need a "telomere seed", or few base pair telomere section, in order to be used in de-novo telomere

formation (Pennaneach *et al.*, 2006). Rahnama *et al.* (2020) also found that of the 24 *M. oryzae de novo* telomere formation events studied, 19 of them were taken from internal sequence with telomere seed. The lack of any overlap between internal sequence taken from *de novo* telomere implies that either telomere seeds are widely distributed throughout the genome or are less required than initially thought for *de novo* telomere formation. Further analysis of the data will be needed to check for the exact amount of internal sequence with telomere seeds. The variety between replicate datasets in terms of the number of *de novo* telomere formation events was also striking. When adjusting for the filtering error some data sets had 0 de-novo telomere formation events, while others had as many as 18. The high number of *de novo* telomeres created from internal sequence in some datasets also brings into the question the effectiveness or purpose of capturing internal sequence to use as telomeres. It's possible that the *de novo* telomeres created from internal sequence continued to be unstable causing more breakage events, which lead to more de-novo telomere formation events; if this continued long enough it may have led to cell death. All the data sets studied were taken from lab grown cultures, so living fungal cells would be growing on top of dead fungal cells. There would be no guarantee that the DNA studied would be from a living cell compared to a dead one. It's possible then that some internal sequence may further destabilize the chromosomal ends leading to a death spiral where de-novo telomeres keep forming, breaking, and then being replaced until the cell dies. Cell autophagy is

also associated with replicative crises as an anti-tumor measure, so this pathway may also be involved in the cell death hypothesis (Nassour *et al.*, 2019). Is this form of *de novo* telomere formation then a strategy to further diversity and adaptation via telomere reconstruction, or is it an emergency telomere repair mechanism with a high failure rate? Further analysis will need to be done on more *M. oryzae* datasets to determine if these observations are consistent across multiple strains before these questions can be answered.

WORKS CITED

Barry, J. D., Ginger, M. L., Burton, P., & McCulloch, R. (2003). Why are parasite contingency genes often associated with telomeres?. *International journal for parasitology*, 33(1), 29–45. https://doi.org/10.1016/s0020-7519(02)00247-3

Biessmann, H., & Mason, J. M. (2003). Telomerase-independent mechanisms of telomere elongation. *Cellular and molecular life sciences: CMLS*, 60(11), 2325–2333. https://doi.org/10.1007/s00018-003-3247-9

Diotti, R., Esposito, M., & Shen, C. H. (2021). Telomeric and Sub-Telomeric Structure and Implications in Fungal Opportunistic Pathogens. *Microorganisms*, 9(7), 1405. https://doi.org/10.3390/microorganisms9071405

Farman, M. L., & Kim, Y. S. (2005). Telomere hypervariability in Magnaporthe oryzae. *Molecular plant pathology*, 6(3), 287–298. https://doi.org/10.1111/j.1364-3703.2005.00285.x

Farman M. L. (2007). Telomeres in the rice blast fungus Magnaporthe oryzae: the world of the end as we know it. *FEMS microbiology letters*, 273(2), 125–132. https://doi.org/10.1111/j.1574-6968.2007.00812.x

Fulnecková, J., Sevcíková, T., Fajkus, J., Lukesová, A., Lukes, M., Vlcek, C., Lang, B. F., Kim, E., Eliás, M., & Sykorová, E. (2013). A broad phylogenetic survey unveils the diversity and evolution of telomeres in eukaryotes. *Genome biology and evolution*, *5*(3), 468–483. https://doi.org/10.1093/gbe/evt019

Hazelhurst, S., & Lipták, Z. (2011). KABOOM! A new suffix array based algorithm for clustering expression data. *Bioinformatics (Oxford, England),* 27(24), 3348–3355. https://doi.org/10.1093/bioinformatics/btr560

Kordyukova, M., Olovnikov, I., & Kalmykova, A. (2018). Transposon control mechanisms in telomere biology*. Current opinion in genetics & development*, 49, 56–62. https://doi.org/10.1016/j.gde.2018.03.002

Kwapisz, M., & Morillon, A. (2020). Sub-telomeric Transcription and its Regulation. *Journal of molecular biology*, 432(15), 4199–4219. https://doi.org/10.1016/j.jmb.2020.01.026

Langston, L. D., & O'Donnell, M. (2006). DNA replication: keep moving and don't mind the gap. *Molecular cell*, *23*(2), 155–160. https://doi.org/10.1016/j.molcel.2006.05.034

Li, W., Rehmeyer, C. J., Staben, C., & Farman, M. L. (2005a). TruMatch--a BLAST post-processor that identifies bona fide sequence matches to genome assemblies. *Bioinformatics (Oxford, England)*, 21(9), 2097–2098. https://doi.org/10.1093/bioinformatics/bti257

Li, W., Rehmeyer, C. J., Staben, C., & Farman, M. L. (2005b). TERMINUS--Telomeric End-Read Mining IN Unassembled Sequences. *Bioinformatics (Oxford, England),* 21(8), 1695–1698. https://doi.org/10.1093/bioinformatics/bti181

Nassour, J., Radford, R., Correia, A., Fusté, J. M., Schoell, B., Jauch, A., Shaw, R. J., & Karlseder, J. (2019). Autophagic cell death restricts chromosomal instability during replicative crisis. *Nature*, 565(7741), 659–663. https://doi.org/10.1038/s41586-019-0885-0

Ohki, R., Tsurimoto, T., & Ishikawa, F. (2001). In vitro reconstitution of the end replication problem. *Molecular and cellular biology*, 21(17), 5753–5766. https://doi.org/10.1128/MCB.21.17.5753-5766.2001

Pennaneach, V., Putnam, C. D., & Kolodner, R. D. (2006). Chromosome healing by *de novo* telomere addition in *Saccharomyces cerevisiae*. *Molecular microbiology*, 59(5), 1357–1368. https://doi.org/10.1111/j.1365-2958.2006.05026.x

Pray, L. A. (2008). Transposons: The Jumping Genes. *Nature Education, 1*(1), 204. https://www.nature.com/scitable/topicpage/transposons-the-jumping-genes-518/

Rahnama, M., Novikova, O., Starnes, J. H., Zhang, S., Chen, L., & Farman, M. L. (2020). Transposon-mediated telomere destabilization: a driver of genome evolution in the blast fungus. *Nucleic acids research*, 48(13), 7197–7217. https://doi.org/10.1093/nar/gkaa287

Rahnama, M., Wang, B., Dostart, J., Novikova, O., Yackzan, D., Yackzan, A., Bruss, H., Baker, M., Jacob, H., Zhang, X., Lamb, A., Stewart, A., Heist, M., Hoover, J., Calie, P., Chen, L., Liu, J., & Farman, M. L. (2021). Telomere Roles in Fungal Genome Evolution and Adaptation. *Frontiers in genetics*, 12, 676751. https://doi.org/10.3389/fgene.2021.676751

Rehmeyer, C., Li, W., Kusaba, M., Kim, Y. S., Brown, D., Staben, C., Dean, R., & Farman, M. (2006). Organization of chromosome ends in the rice blast fungus, Magnaporthe oryzae. *Nucleic acids research*, 34(17), 4685–4701. https://doi.org/10.1093/nar/gkl588

Schwartz, S. L., & Farman, M. L. (2010). Systematic overrepresentation of DNA termini and underrepresentation of subterminal regions among sequencing templates prepared from hydrodynamically sheared linear DNA molecules. *BMC genomics*, 11, 87. https://doi.org/10.1186/1471-2164-11-87

Srinivas, N., Rachakonda, S., & Kumar, R. (2020). Telomeres and Telomere Length: A General Overview. *Cancers*, *12*(3), 558. https://doi.org/10.3390/cancers12030558

Wilson, R. A., & Talbot, N. J. (2009). Under pressure: investigating the biology of plant infection by Magnaporthe oryzae. *Nature reviews. Microbiology*, 7(3), 185–195. https://doi.org/10.1038/nrmicro2032

Wynford-Thomas, D. & Kipling, D. (1997). The end-replication problem. *Nature, 389*, 551. https://doi.org/10.1038/39210

APPENDIX A: TELOPORTWRAPPER VERSION HISTORY

TeloPortWrapper has undergone six major revisions during this research project. Version one used a convoluted output directory system where there were multiple results directories for each individual program in the pipeline and within those folders there were sub folders for each dataset. Version 2 reversed this structure by having the highest-level directory be the one named for each dataset with the individual output folders being contained within each dataset's own output folder. There was also support added for genome visualization via R compatible output files. This would be removed in later versions in favor of IGV compatible output files.

In version 3, paired end reads were used in place of the pure telomere reads due to the low quality of the reads. The automatic results builder was also added to this version. Version 4 added a dedicated outputs folder so the output files would not be on the same level as the TeloPortWrapper files, which allows for easier clearing of output folders. The initial output folders for TeloPort were also consolidated into a single folder called "pipeline_out." The dictionary of telomere containing contigs and the BLAST interrogation features were also added.

In version 5 the telomere read ID system was added and output files were also tagged with their dataset name. The candidate *de novo* telomere reads were

also concatenated into a single file to increase the efficiency of the program. Now BLAST would only need to be run once against the genome rather than every single candidate *de novo* needing to be run individually. MoTeR and rDNA filtering were also added. This is also the update which added IGV compatible file output. Version 6 rewrites version 5 to be more modular. Now every major component of the program is its own function which is called. It also adds command line input support via ArgParser.

APPENDIX B: THE COMPLETE TELOPORTWRAPPER V6 CODE

```
import os
import time
import argparse

#command line arguments

parser = argparse.ArgumentParser(prog='TeloPortWrapper',description='An all-in-one pipeline and analysis tool for de-novo Telomeres',epilog='Using no arguments will cause the program to run in simple mode. The -s and -i options are mutually exclusive. If simple mode is not used options -b is required.')

parser.add_argument('-s', action='store', nargs=2, help='Allows for separated fastq file names to be inputted on the command line. This argument expects two file names separated by a space.', default=['x','x'])

parser.add_argument('-i', action='store', help='Allows for the interleaved fastq name to be inputted on the command line. This argument expects one file name.', default="x")
```

```
parser.add_argument('-b', action='store', help='Allows for the assembled genome file name to be inputted on the command line.', default="x")

parser.add_argument('-d', action='store', help='The name of the directory where all output files will be placed. The directory name will default to the name of the assembled genome file. Directory names must be unique', default="x")

parser.add_argument('--cut', action='store', type=int, help='The number of reads in a cluster before it becomes labeled as a set of single reads. Default is 5.', default=5)

parser.add_argument('--simple', action='store_true', help='Activates Simple mode, a step by step input mode using default values', default=False)

args = parser.parse_args()

#determines if an interleaved of separated input is needed

def inputCollect():
        choiceMade = False
        global choice, genomeR1, genomeR2, genomeI, directory, blastD, cutoff
        cutoff = 5
```

```
        while not choiceMade:
                choice = input('Please enter if you are using separated (s) genome
files or an interleaved (i) file ')
                if choice == 'i' or choice == 's':
                        choiceMade = True
                else:
                        print('that is not a valid input. Please enter i for interleaved or
s for separated')
        if choice == 's':


                genomeR1 = input('Specify the name of the R1 file you want: ')
                filePathTest('Genomes/' + genomeR1)
                genomeR2= input('Specify the name of the R2 file you want: ')
                filePathTest('Genomes/' + genomeR2)
        elif choice == 'i':
                genomeI = input('Specify the name of the interleaved fastq file ')
                filePathTest('Genomes/' + genomeI)
        blastD = input('Specify genome file to be blasted against the single reads
')
        directory = input('Specify the output directory name. Press enter for
default: ')
        if directory == '':
```

```
            directory = blastD
        filePathTest('Genomes/db/' + blastD)

#test to make sure the file path is valid

def filePathTest(fileName):

        try:
            test= open(fileName, 'r')
        except:
            print('We cannot find one of the files you specified please reinput')
            inputCollect()
        else:
            test.close()
#write the input file to a list variable

def fileListBuilder(input):

        read = open(input, 'r')
        lineCnt = 0
        lines = []
        for line in read:
```

```
                lines.append(line)
                lineCnt = lineCnt + 1
        read.close()
        return lines, lineCnt
#TeloPort Pipeline
def teloPortPipeline():
        global telos
        telos = {}
        os.system('mkdir -p Programs/TeloPort/Outputs/' + directory +
'/{PipelineOut,muscle_out,interrogate_out/single_reads,blast_out/{blastGenomes,
blastMorepeats,blastMoters,blastRdna,blastTel},cons_out}')
        if choice == 's':
                os.system('Programs/TeloPort/build/apps/telomereFinder -s ' +
'Genomes/' + genomeR1 + ' ' + 'Genomes/' + genomeR2 + ' -o ' +
'Programs/TeloPort/Outputs/' + directory + '/PipelineOut/')
        elif choice == 'i':
                os.system('Programs/TeloPort/build/apps/telomereFinder -i ' +
genomeI + ' -o ' + 'Programs/TeloPort/Outputs/' + directory + '/PipelineOut/')
        os.system('mv Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/pairReads.fastq Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/' + directory + 'subtelReads.fastq')
```

```
        lines, lineCnt = fileListBuilder('Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/' + directory + 'subtelReads.fastq')
        write = open('Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' +
directory + 'subtelReads2.fastq', 'a')
        for x in range(lineCnt):
                if lines[x][0] == '@' and x % 4 == 0:
                        newLine = '@' + directory + str(x) + '\t' + lines[x].split('@')[1]
                        write.write(newLine)
                        try:
                                lines[x + 1].split("TTAGGGTTAGGG")[1]
                        except:
                                try:
                                        lines[x + 1].split("CCCTAACCCTAA")[1]
                                except:
                                        pass
                                else:
                                        telos[directory + str(x)] = (lines[x + 1],
"CCCTAA")
                        else:
                                telos[directory + str(x)] = (lines[x + 1], "TTAGGG")
                else:
                        write.write(lines[x])
```

```
        write.close()
        os.system('rm Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' +
directory + 'subtelReads.fastq')
        os.system('mv Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' +
directory + 'subtelReads2.fastq Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/' + directory + 'subtelReads.fastq')
        os.system('Programs/TeloPort/build/apps/junctionFinder -i
Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' + directory +
'subtelReads.fastq -o Programs/TeloPort/Outputs/' + directory + '/PipelineOut/ --
revc false --splitJunc true --splitDir false')
        os.system('mv Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/telAdjSeq.fastq Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/' + directory + 'mergedTelAdjSeqs.fastq')
        os.system('rm Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/telSeq.fastq')
        os.system('Programs/TeloPort/build/apps/sequenceQuality -i
Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' + directory +
'mergedTelAdjSeqs.fastq -o Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/' + directory + 'HiQualTelAdjacent.fasta --ofmt fasta -c 0 -l 30')
        os.system('wcd Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' +
directory + 'HiQualTelAdjacent.fasta -l 40 -T 5 -H 0 --show_clusters --histogram >
```

```
Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' + directory +
'clusters.wcd')
        os.system('Programs/TeloPort/build/apps/wcdInterrogate -w
Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' + directory +
'clusters.wcd -i Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' +
directory + 'HiQualTelAdjacent.fasta -s Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/' + directory + 'mergedTelAdjSeqs.fastq -f fastq -o
Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' + directory +
'Clusters.out -r Programs/TeloPort/Outputs/' + directory + '/interrogate_out/ --
indices --sort --size 1')
#This will turn all single clusters into single reads
def singleMaker():
        last = lastClusterCount()
        first = firstSingleCount()
        y = len(os.listdir('Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/')) - 1
        for cnt in range(y):
                read = open('Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/cluster' + str(cnt) + '.fasta', 'r')
                for line in read:
                        if line[0] == '>':
```

```
                                newLine = line.split('\t')[0] + '\tcluster' + str(cnt) + '\t' +
line.split('\t')[1]
                                write = open('Programs/TeloPort/Outputs/' + directory
+ '/interrogate_out/' + directory + 'cluster' + str(cnt) + '.fasta', 'a')
                                write.write(newLine)
                                write.close()
                        else:
                                write = open('Programs/TeloPort/Outputs/' + directory
+ '/interrogate_out/' + directory + 'cluster' + str(cnt) + '.fasta', 'a')
                                write.write(line)
                                write.close()
                read.close()
                os.system('rm Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/cluster' + str(cnt) + '.fasta')
        while first <= last:
                os.system('cat Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/' + directory + 'cluster' + str(first) + '.fasta >>
Programs/TeloPort/Outputs/' + directory + '/interrogate_out/single_reads/' +
directory + 'single_reads.fasta')
                os.system('rm Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/' + directory + 'cluster' + str(first) + '.fasta')
                first = first + 1
```

```
#this counts the last cluster in the directory
def lastClusterCount():
	intOut = open('Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' +
directory + 'Clusters.out', 'r')
	cnt = 0
	for line in intOut:
		if line[0] == 'c':
			cnt = cnt + 1
			last = cnt - 1
	intOut.close()
	return last
#this will count the first single read
def firstSingleCount():
	intOut = open('Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' +
directory + 'Clusters.out', 'r')

	global clusterDict
	clusterDict = {}
	for line in intOut:
		if line[0:2] == "cl":
			clusterDict[line.split(';')[0]] = line.split('=')[1]
```

```
	intOut = open('Programs/TeloPort/Outputs/' + directory + '/PipelineOut/' +
directory + 'Clusters.out', 'r')
	cnt = 0


	for line in intOut:
		if line[0:2] == "cl":
			if int(line.split('=')[1]) <= int(cutoff):
				first = cnt
				break
			cnt = cnt + 1
	return first
#Code to automatically run muscle
def autoMuscle():
	start = 0
	end = firstSingleCount() - 1
	while start <= end:
		os.system('Programs/bin/muscle -in Programs/TeloPort/Outputs/' +
directory + '/interrogate_out/' + directory + 'cluster' + str(start) + '.fasta -out
Programs/TeloPort/Outputs/' + directory + '/muscle_out/cons' + str(start) + '.msa')
		start = start + 1
	start = 0
	while start <= end:
```

```
            os.system('cons -sequence Programs/TeloPort/Outputs/' + directory
+ '/muscle_out/cons' + str(start) + '.msa -outseq Programs/TeloPort/Outputs/' +
directory + '/cons_out/clusterCons' + str(start) + '.fasta -name cluster' + str(start))
            start = start + 1
#This will concatonate all consensus sequences into a single file to blast
def catBlast():
    fileCnt = os.listdir('Programs/TeloPort/Outputs/' + directory + '/cons_out/')
    num = len(fileCnt)

    start = 0

    command = ''

    while start < num:
        command = command + ('Programs/TeloPort/Outputs/' + directory
+ '/cons_out/clusterCons' + str(start) + '.fasta ')
        start = start + 1
    os.system('cat ' + command + '> Programs/TeloPort/Outputs/' + directory
+ '/blast_out/' + directory + 'catCons.fasta')
    os.system('rm -r Programs/TeloPort/Outputs/' + directory + '/cons_out/')
#Filters false positives
def falsePosFilter():
```

```
        global falsePos
        singles, cnt = fileListBuilder('Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/single_reads/' + directory + 'single_reads.fasta')
        true_singles = 0
        for x in range(cnt):
                if singles[x][0] == '>':
                        try:
                                falsePos[singles[x].split('>')[1].split('\t')[0]]
                        except:
                                write = open('Programs/TeloPort/Outputs/' + directory
+ '/interrogate_out/single_reads/' + directory + 'trueSingles.fasta', 'a')
                                write.write(singles[x])
                                write.write(singles[x + 1])
                                true_singles = true_singles + 1
                                write.close()
                        else:
                                write = open('Programs/TeloPort/Outputs/' + directory
+ '/interrogate_out/' + directory + falsePos[singles[x].split('>')[1].split('\t')[0]][0] +
'.fasta', 'a')
                                write.write(singles[x])
                                write.write(singles[x + 1])
                                write.close()
```

```
#non telconig dictionary maker
def dictMaker(input, secondVal):
	read = open(input, 'r')
	dict = {}
	for line in read:
		try:
			dict[line.split('\t')[0]]
		except:
			dict[line.split('\t')[0]] = (line.split('\t')[1],
int(line.split('\t')[secondVal]))
		else:
			if secondVal == 3:
				if line.split('\t')[3] > falsePos[line.split('\t')[0]][1]:
					dict[line.split('\t')[0]] = (line.split('\t')[1],
line.split('\t')[3])
	read.close()
	return dict
#this will generate an updated collection of cluster sizes
def clusterHistogramBuilder():
	fileCnt = os.listdir('Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/')
	num = len(fileCnt) - 1
```

```
	start = 0
	append = open('Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/' + directory + 'clusterHistograph.txt','a')
	while start < num:
		clusterCnt = 0
		read = open('Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/' + directory + 'cluster' + str(start) + '.fasta', 'r')
		for line in read:
			if line[0] == '>':
				clusterCnt = clusterCnt + 1
		command = 'cluster' + str(start) + '\t' + str(clusterCnt) + '\t'
		for x in range(0, clusterCnt):
			command = command + '#'
		command = command + '\n'
		append.write(command)
		read.close()
		start = start + 1
	append.close()
#This will filter out reads which match to Rdna, Moters, and Morepeats and blast
the unmatched reads against the genomes
def moFilter():
	global moterSingles, RdnaSingles, MorepeatsSingles, ran, true_Singles
```

```
        true_Singles = 0
        lines, lineCnt = fileListBuilder('Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/single_reads/' + directory + 'trueSingles.fasta')
        for x in range(lineCnt):
                if lines[x][0]  == '>':
                        try:
                                moterSingles[lines[x].split('\t')[0].split('>')[1]]
                        except:

                                #filter out rdna
                                try:
                                        RdnaSingles[lines[x].split('\t')[0].split('>')[1]]
                                except:

                                        #filter out morepeats
                                        try:

        MorepeatsSingles[lines[x].split('\t')[0].split('>')[1]]
                                        except:
                                                write =
open('Programs/TeloPort/Outputs/' + directory + '/interrogate_out/single_reads/' +
directory + 'trueSinglesGenome.fasta', 'a')
```

```
                        write.write(lines[x])
                        write.write(lines[x + 1])
                        write.close()
                        true_Singles = true_Singles + 1
                else:
                        write =
open('Programs/TeloPort/Outputs/' + directory + '/interrogate_out/single_reads/' +
directory + 'trueSinglesMorepeats.fasta', 'a')
                        write.write(lines[x])
                        write.write(lines[x + 1])
                        write.close()

        else:
                write = open('Programs/TeloPort/Outputs/' +
directory + '/interrogate_out/single_reads/' + directory + 'trueSinglesRdna.fasta',
'a')
                write.write(lines[x])
                write.write(lines[x + 1])
                write.close()

    else:
```

```
                write = open('Programs/TeloPort/Outputs/' + directory
+ '/interrogate_out/single_reads/' + directory + 'trueSinglesMoter.fasta', 'a')
                write.write(lines[x])
                write.write(lines[x + 1])
                write.close()



#This will create a director of every telcontig
def telContigDictMaker():
    os.system('Programs/ncbi-blast-2.11.0+/bin/blastn -query Genomes/db/' +
blastD + ' -subject Genomes/db/telRepeats.fasta -dust no -outfmt "6 qseqid
sseqid pident length mismatch gapopen qstart qend sstart send evalue qlen" >>
Programs/TeloPort/Outputs/' + directory + '/blast_out/blastTel/' + directory +
'telContigsOut6.txt')
    read = open('Programs/TeloPort/Outputs/' + directory +
'/blast_out/blastTel/' + directory + 'telContigsOut6.txt', 'r')
    global telContigs
    telContigs = {}
    posistionStart = 0
    posistionEnd = 0
```

```
        for line in read:

                try:
                        telContigs[line.split('\t')[0] + '*s']
                except:
                        posistionStart = 0

                try:
                        telContigs[line.split('\t')[0] + "*e"]
                except:
                        posistionEnd = 0

                if int(line.split('\t')[7]) < (int(line.split('\t')[11]) / 3):
                        if posistionStart < int(line.split('\t')[7]):
                                telContigs[line.split('\t')[0] + "*s"] = ("start",
int(line.split('\t')[7]))
                                posistionStart = int(line.split('\t')[7])

                if int(line.split('\t')[7]) > (int(line.split('\t')[11]) / 3):
                        if posistionEnd < int(line.split('\t')[7]):
```

```
                                telContigs[line.split('\t')[0] + "*e"] = ("end",
int(line.split('\t')[6]))
                                posistionEnd = int(line.split('\t')[6])




        read.close()


        write = open('Programs/TeloPort/Outputs/' + directory +
'/blast_out/blastTel/' + directory + 'TelDictionary.txt', 'a')


        write.write(str(telContigs))


        write.close()
#blast interrogation
def blastInterrogation():
        readLines, lineNum = fileListBuilder('Programs/TeloPort/Outputs/' +
directory+ '/blast_out/blastGenomes/' + directory + 'blastGenomeOut6.txt')
        command = ""
        for cnt in range(lineNum):
                telContigCheck = False
```

```
command = ""


queryName = readLines[cnt].split('\t')[0]
print(queryName)


subjectName = readLines[cnt].split('\t')[1]
print(subjectName)
telDisS = 0


try:
        telContigs[subjectName + "*s"]
except:
        command = "\tTelNs"
else:
        command = "\tTelYs"
        telContigCheck = True



if telContigCheck:
        posistion = telContigs[readLines[cnt].split('\t')[1] + '*s'][1]
        telDisS =  abs(int(readLines[cnt].split('\t')[8]) - posistion)
telContigCheck = False
```

```
telDisE = 0


try:
        telContigs[subjectName + "*e"]
except:
        command = command + "\tTelNe"
else:
        command = command + "\tTelYe"
        telContigCheck = True



if telContigCheck:
        posistion = telContigs[readLines[cnt].split('\t')[1] + '*e'][1]
        telDisE =  abs(int(readLines[cnt].split('\t')[8]) - posistion)


telContigCheck = False
if abs(telDisS) < abs(telDisE) and telDisS != 0:
        command = command + "\t" + str(abs(telDisS)) + "\t"
elif abs(telDisS) > abs(telDisE) and telDisE == 0:
        command = command + "\t" + str(abs(telDisS)) + "\t"
elif abs(telDisE) < abs(telDisS) and telDisE != 0:
        command = command + "\t" + str(abs(telDisE)) + "\t"
```

```
elif abs(telDisE) > abs(telDisS) and telDisS == 0:
        command = command + "\t" + str(abs(telDisE)) + "\t"
else:
        command = command + "\t" + str(abs(telDisE)) + "\t"


try:
        moterContigs[subjectName]
except:
        command = command + "MoterN\t"
else:
        command = command + "MoterY\t"


revS = False


revQ = False
if int(readLines[cnt].split('\t')[6]) > int(readLines[cnt].split('\t')[7]):
        revS = True


if int(readLines[cnt].split('\t')[8]) > int(readLines[cnt].split('\t')[9]):
        revQ = True
rev = "revF"
```

```
                if not revS and revQ:
                        rev = "revT"
                command = command + rev +'\n'

                command = readLines[cnt].split('\n')[0] + command

                append = open('Programs/TeloPort/Outputs/' + directory+
'/blast_out/blastGenomes/' + directory + 'blastGenomeIntOut6.txt', 'a')

                append.write(command)

                append.close()

                cnt = cnt + 1
#Builds a Gff file
def gffBuilder():
        lines, linecnt = fileListBuilder('Programs/TeloPort/Outputs/' + directory+
'/blast_out/blastGenomes/' + directory + 'blastGenomeIntOut6.txt')
        write = open('Programs/TeloPort/Outputs/' + directory+ '/' + directory +
'.gff', 'a')
        command = '##gff-version 3.1.26\n'
        write.write(command)
```

```
	for x in range(linecnt):
		if int(lines[x].split('\t')[8]) < int(lines[x].split('\t')[9]):
			strand = '+'
			color = "#281e8d"
		elif int(lines[x].split('\t')[8]) > int(lines[x].split('\t')[9]):
			strand = '-'
			color = "#ff2211"
		length = abs(int(lines[x].split('\t')[9]) - int(lines[x].split('\t')[8]))
		command = lines[x].split('\t')[1] + '\t' + 'teloPortWrapper' + '\t' +
'telomere' + '\t' + lines[x].split('\t')[8] + '\t' + lines[x].split('\t')[9] + '\t' +
lines[x].split('\t')[10] + '\t' + strand + '\t' + '.' + '\t' + 'color=' + color + ';ID=Telomere'
+ str(x) + ';Name=' + lines[x].split('\t')[0] + ';Length=' + str(length) + ';Seq=' +
telos[lines[x].split('\t')[0]][0] + ';Distance from Telomere=' + lines[x].split('\t')[14] +
'\n'
		write.write(command)
#generates results txt file
def resultsBuilder():
	command = directory + '\n'

	rawReadCnt = 0

	if choice == 's':
```

```
                read = open('Genomes/' +genomeR1, 'r')
                for line in read:
                        if line[0] == '@':
                                rawReadCnt = rawReadCnt + 1
                read.close()
                read = read = open('Genomes/' + genomeR2, 'r')
                for line in read:
                        if line[0] == '@':
                                rawReadCnt = rawReadCnt + 1
                read.close()
        elif choice == 'i':
                read = open('Genomes/' + genomeI, 'r')
                for line in read:
                        if line[0] == "@":
                                rawReadCnt = rawReadCnt + 1
                read.close()
        command = command + ('Number of raw reads processed: ' +
str(rawReadCnt) + '\n')

        teloCnt = 0
```

```
        read = open('Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/telReads.fastq', 'r')

        for line in read:
                if line[0] == '@':
                        teloCnt = teloCnt + 1
        read.close()

        os.system('rm Programs/TeloPort/Outputs/' + directory +
'/PipelineOut/telReads.fastq')

        command = command + ('Number of Telomere reads processed: ' +
str(teloCnt) + '\n')

        command = command + ('Number of de novo telomeres: ' +
str(true_Singles) + '\n')

        clusterCnt = len(os.listdir('Programs/TeloPort/Outputs/' + directory +
'/interrogate_out')) - 2

        command = command + ('Number of clusters: ' + str(clusterCnt) + '\n')
```

```
        genomesCnt = 0

        read = open('Programs/TeloPort/Outputs/' + directory +
'/blast_out/blastGenomes/' + directory + 'blastGenomeIntOut6.txt', 'r')

        for line in read:
                genomesCnt = genomesCnt + 1
        read.close()

        command = command + ('Number of de novo telomeres that match up
with the genome: ' + str(genomesCnt) + '\n')

        moterCnt = 0

        read = open('Programs/TeloPort/Outputs/' + directory +
'/blast_out/blastMoters/' + directory + 'blastMoterOut6.txt', 'r')

        for line in read:
                moterCnt = moterCnt + 1
        read.close()
```

```
        command = command + ('Number of de novo telomeres that match up
with the MOTER sequences: ' + str(moterCnt) + '\n')

        command = command + 'Time spent running ' + (str(time.time() -
startTime)) + '\n'

        command = command + ('Distribution of clusters \n')

        append = open('Programs/TeloPort/Outputs/' + directory + '/' + directory +
'results.txt', 'a')

        append.write(command)

        append.close()

        os.system('cat Programs/TeloPort/Outputs/' + directory +
'/interrogate_out/' + directory + 'clusterHistograph.txt >>
Programs/TeloPort/Outputs/' + directory + '/' + directory + 'results.txt')

        os.system('rm Programs/TeloPort/Outputs/' + directory + '/interrogate_out/'
+ directory + 'clusterHistograph.txt')
```

```
        os.system('cat Programs/TeloPort/Outputs/' + directory+
'/blast_out/blastGenomes/' + directory + 'blastGenomeIntOut6.txt >>
Programs/TeloPort/Outputs/' + directory + '/' + directory + 'results.txt')




if args.simple:
        inputCollect()
elif args.b == "x":
        inputCollect()
elif args.s != 'x':
        genomeR1 = args.s[0]
        genomeR2 = args.s[1]
        if args.d == 'x':
                directory = args.b
        else:
                directory = args.d
        blastD = args.b
        cutoff = args.cut
        choice = 's'
elif args.i != 'x':
```

```
        genomeI = args.i
        if args.d == 'x':
                directory = args.b
        else:
                directory = args.d
        blastD = args.b
        cutoff = args.cut
        choice = 'i'
startTime = time.time()
teloPortPipeline()
singleMaker()
autoMuscle()
catBlast()
os.system('Programs/ncbi-blast-2.11.0+/bin/blastn -query
Programs/TeloPort/Outputs/' + directory + '/interrogate_out/single_reads/' +
directory + 'single_reads.fasta -subject Programs/TeloPort/Outputs/' + directory +
'/blast_out/' + directory + 'catCons.fasta -outfmt 6 > Programs/TeloPort/Outputs/'
+ directory + '/blast_out/' + directory + 'blastCons.txt')
falsePos = dictMaker('Programs/TeloPort/Outputs/' + directory + '/blast_out/' +
directory + 'blastCons.txt', 3)
falsePosFilter()
clusterHistogramBuilder()
```

```
os.system('Programs/ncbi-blast-2.11.0+/bin/blastn -query
Programs/TeloPort/Outputs/' + directory + '/interrogate_out/single_reads/' +
directory + 'trueSingles.fasta -subject Genomes/db/MoTeRs.gb -outfmt "6 qseqid
sseqid pident length mismatch gapopen qstart qend sstart send evalue slen" >>
Programs/TeloPort/Outputs/' + directory + '/blast_out/blastMoters/' + directory +
'blastMoterOut6.txt')
os.system('Programs/ncbi-blast-2.11.0+/bin/blastn -query
Programs/TeloPort/Outputs/' + directory + '/interrogate_out/single_reads/' +
directory + 'trueSingles.fasta -subject Genomes/db/Mo_rDNA.fasta -outfmt "6
qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue
slen" >> Programs/TeloPort/Outputs/' + directory + '/blast_out/blastRdna/' +
directory + 'blastRdnaOut6.txt')
os.system('Programs/ncbi-blast-2.11.0+/bin/blastn -query
Programs/TeloPort/Outputs/' + directory + '/interrogate_out/single_reads/' +
directory + 'trueSingles.fasta -subject Genomes/db/MoRepeats.fasta -outfmt "6
qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue
slen" >> Programs/TeloPort/Outputs/' + directory + '/blast_out/blastMorepeats/' +
directory + 'blastMorepeatsOut6.txt')
moterSingles = dictMaker('Programs/TeloPort/Outputs/' + directory +
'/blast_out/blastMoters/' + directory + 'blastMoterOut6.txt', 7)
RdnaSingles = dictMaker('Programs/TeloPort/Outputs/' + directory +
'/blast_out/blastRdna/' + directory + 'blastRdnaOut6.txt', 7)
```

```
MorepeatsSingles = dictMaker('Programs/TeloPort/Outputs/' + directory +
'/blast_out/blastMorepeats/' + directory + 'blastMorepeatsOut6.txt', 7)
ran = {}
moFilter()
os.system('Programs/ncbi-blast-2.11.0+/bin/blastn -query
Programs/TeloPort/Outputs/' + directory + '/interrogate_out/single_reads/' +
directory + 'trueSinglesGenome.fasta -subject Genomes/db/' + blastD + ' -outfmt
"6 qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue
slen" >> Programs/TeloPort/Outputs/' + directory + '/blast_out/blastGenomes/' +
directory + 'blastGenomeOut6.txt')
telContigDictMaker()
os.system('Programs/ncbi-blast-2.11.0+/bin/blastn -query Genomes/db/' + blastD
+ ' -subject  Genomes/db/MoTeRs.gb -outfmt "6 qseqid sseqid pident length
mismatch gapopen qstart qend sstart send evalue qlen" >>
Programs/TeloPort/Outputs/' + directory + '/blast_out/blastMoters/' + directory +
'moterContigsOut6.txt')
moterContigs = dictMaker('Programs/TeloPort/Outputs/' + directory +
'/blast_out/blastMoters/' + directory + 'moterContigsOut6.txt', 7)
write = open('Programs/TeloPort/Outputs/' + directory + '/blast_out/blastMoters/'
+ directory + 'MotersDictionary.txt', 'a')
write.write(str(moterContigs))
write.close()
```

```
blastInterrogation()

gffBuilder()

resultsBuilder()


#clean up


write = open('Programs/TeloPort/Outputs/' + directory + '/oldClusters.txt', 'a')


for x in range(len(clusterDict)):

        write.write('cluster' + str(x) + ': ' + clusterDict['cluster ' + str(x)])



print('final time : ' + str(time.time() - startTime))
```

APPENDIX C: FULL VERSION OF TABLE 4

Table 5 is a more complete version of table 4 which includes all 106 *de novo* telomere reads that aligned to the assembled genome rather than the 88 *de novo* telomere reads which only aligned with the assembled genome at a single location. The red colored text refers to the multi aligned *de novo* telomere reads which are assumed to contain rDNA or be taken from internal telomeres. Similar to table 4, the bolded sequence in the 9th column refers to the sub-telomere region and the non-bolded refers to the telomere repeats.

Table 5

Candidate De novo BLAST alignment and IGV visualization Summary

Genome	De-novo telomere ID	# of reads	genome match?	distance from telomere	telRepeats visible?	Contig Hit	Notes	Telomere junction
ERR2061611	3692	2	Yes (rev of 3696)	3911932	Yes	Chr3		**CAGCTCCTGGTTG** ATGGGTTCGGGTTAGGG
ERR2061611	3696	2	Yes	3912005	Yes	Chr3		CCCTAACCCGAACCCAT **CAACCAGGAGCTGTAGT**
ERR2061611	9796	1	Yes	962965	Yes	Chr5		**TACTAGTGGGTTCGG** GTTAGGGTTAGGG

ERR2061611	6616	1	Yes (rev of 6620)	1393405	Yes	Chr2		**GCTGATCGAAAAGGCACGG**TTAGGGTTAGGG
ERR2061611	6620	1	Yes	1393369	Yes	Chr2		CCCTAACCCTAACCCTAA**CCGTGCCTTT**
ERR2061611	5500	1	Yes	7915472	Yes	Chr2		**GACGCCAGACGC**ATTAGGGTTAGGG
ERR2061611	10948	1	Yes	6133529	Yes	Chr6		**CGCCGATGGTAG**GGTTAGGG
ERR2061611	5448	1	Yes	1973151	yes	Chr4		CCCAAACCCGAAGGGT**AGTACAGCCA**
ERR2061611	4040	1	yes	1039395	Yes	Chr3		**ATTCTTTTGG**TTAGGGTTAGGG
ERR2061611	6204	1	yes	1912175	Yes	Chr4		CCCTAACCCTAT**GTTTGCTTAT**
ERR2061612	6524	1	Yes	984942	Yes	Chr2	Small bit of telomere found in genome	**TCCGTTTA**AGGGTTAGGG
ERR2061612	6780	1	Yes	935235	Yes	Chr7		CCCTAACCCTAA**TCCCAATAAC**
ERR2061612	6784	1	yes (rev of 6780)	935142	Yes	Chr7	little bit of reverse telomere	**TTTTGTTATTGGGA**TTAGGGTTAGG
ERR2061612	7084	1	Yes	1067342	Yes	Chr 2	Missing a TTTGGGTTTGGGTTTG	**CAGCGCGG**TTTGGGTTTGGG

							GGTTCGGG from the genome	
ERR2061613	8428	1	Yes	6666023	Yes	Chr1	It is hitting multiple places near the start of Chr 1 with no telomeres or moters but an A rich compliment	**CGTCCCTACTTT** TGTTTTTGTTTTT
ERR2061613	8552	2	Yes	1825871	Yes	Chr 7		**CAGGTGTTGTG** TTAGGGTTAGGG
ERR2061613	8548	2	Yes (rev of 8552)	1825797	Yes	Chr 7		CCCTAACCCTAACA **CAACACCTGCAA**
ERR2061613	3288	1	Yes	1966404	Yes	Chr6		**GCCGTCG** TAGGGTTAGGG
ERR2061613	8656	1	Yes	1073968	yes	Chr4	Missing a TAGGG from the genome	**TAAAGTCGCG** TAGGGTTAGGGTTAGGG
ERR2061613	4720	1	Yes (twice)	172434 and 78897	Yes	Chr 8 mini	The forward and reverse version of this sequence hit	**TTCCAAATA** TTAGGGTTAGGGTTAGGG

							two different spots in the genome. No internal telomeres or moters	
ERR2061613	16	1	Yes	519885, 104016, 531690, 105253....	Yes	Chr 7, 3, 1 6, 8	Matches to internal telomeres, motoer repeats, and random parts of the genome.	Internal telomere
ERR2061613	2356	1	Yes	0 (no Telomere found on contig)	Yes	Chr 9 mini	Missing a GGGGAACCCTTCGGGTTTGGGTTGGGTTTGGGTTAGGG from the Genome (Moter 5' end)	**TTAAAAAACA** GGGGGAACCCTTCGGGTTTGGG
ERR2061613	5584	1	Yes	688504	Yes	Chr 8 mini	Missing a CCCTTCGGGTTT	**GGATTCTCA** GCCCTTCGGGTTTG

							GGGTTTGGGTTTGGGTTGGGTTCGGG from the genome	GGTTTGGG
ERR2061614	6272	1	Yes	1845099	Yes	Chr 5		**TTTGTTTGA** TGGGTTTGGGTTTGGG
ERR2061614	9672	1	Yes	1439649	Yes	Chr 1		**CCCCGGCTACG** TAGGGTTAGGG
ERR2061614	252	1	Yes	2078176	Yes	Chr 1		**TTTCTTGG** TTAGGGTTAGGG
ERR2061614	9212	1	Yes	1048190	Yes	Chr 7		**ATGTTTTGA** TAGGGTTAGGG
ERR2061615	5872	1	Yes	many hits	Yes	Chr 1-8	Internal telomere behavior like 16 but with a TTAGGG repeat at the end	**GGAAAAGG** TTAGGGTTAGGG
ERR2061615	4760	1	Yes	many hits	Yes	Chr 3-8	CCCTAA repeats internally, internal telomere behavior	CCCTAACCCTAACCCTAA **TAAAG**
ERR2061616	9324	2	Yes (rev of 9328)	3063180	Yes	Chr 6	Missing an AG/C	**AGTGTCGT**

							T from the end.	AGTTAGG GTTAGGG
ERR2061616	9328	2	Yes	3063180	Yes	Chr 6		CCCTAAC CCTAACT **ACGACAC TCAT**
ERR2061616	9864	2	Yes (rev of 9868)	2610572	Yes	Chr4	Missing a GG/CC from the end	CCCAAAC CCGAAGG **CCCAGAA TTAATAA A**
ERR2061616	9868	2	Yes	2610528	Yes	Chr 4	Missing a CC from the end	**TAATTCT GGG** CCTTCGG GTTTGGG TTTGGG
ERR2061616	3512	2	Yes	1620122	Yes	Chr 6	Missing about 2 bp	**ACTACAA TTG** TGTTAGG GTTAGGG
ERR2061616	3508	2	Yes (rev of 3512)	1620212	yes	Chr6	missing about 2 bp	CCCTAAC CCTAACA **CAATTGT AGTCGCC A**
ERR2061616	9552	1	Yes	1966945	yes	Chr7	missing an atg/cat	**ACCAAGG GCTTCCA G** GATGTTG AGGGTTA GGG
ERR2061616	4928	1	Yes(rev of 3532)	1062688	Yes	Chr 4		CCCTAAC CCTAACC CTA **CGGACCC GCT**
ERR2061616	4932	1	Yes	1062761	Yes	Chr4		**GAGAGC GGGTCCG** TAGGGTT AGGGTTA GGG
ERR2061616	5504	1	Yes (rev of 5500)	3157449	Yes	Chr1		CCCTAAC CCTAA **CCTTCCC AATCAG**
ERR2061616	5500	1	Yes	3157502	yes	Chr1		**TGATTGG GAAGG** TTAGGGT TAGGG
ERR2061616	4432	1	Yes	243697	Yes	Chr2		**CCCAGGT CAGGTTG**

								GG CTTCGGG TTTGGGT TT
ERR20616 16	2004	1	Yes	1282533	Yes	Chr 2	missin g a GCC/ CGG	**TCGAATT GGTTT** GCCCTTC GGGTTTG GG
ERR20616 16	1592	1	Yes	many hits	Yes	Chr 3,4,6,7	Hit Moter s, intern al Telom ere like behavi or	TAAAAAA TAATAT **ATTTATT AATATTA CGTGGAA CGTAAT**
ERR20616 16	5476	1	Yes	276912	Yes	Chr2		**TGGGTGG CGTGCCT G** GGTAGGG TTAGGGT TAGGG
ERR20616 16	4388	1	Yes	163951	Yes	Chr2		**GAGTCAC GGCTTTG** TTTGGGT TCGGGTT AGGG
ERR20616 17	5172	1	Yes	Many hits within 6000 bp	Yes	Chr 3	rDna like massi ve seque nce cover age increa se at each hit	rDNA
ERR20616 17	4032	1	Yes	225967	Yes	Chr6		**TTTGAGC** GCTACTT TCGGCGT TAGGGTT AGGG
ERR20616 17	6068	1	Yes	1369166	Yes	Chr6		**CCCCATT CGAG** GATAGGG TTAGGGT TAGGG
ERR20616 18	7760	1	Yes	many hits	Yes	Chr 1-8	Match ed with	n/a

							some moters despite not blasting to any	
ERR2061618	7204	1	Yes	3746746	Yes	Chr 2		**TTGCCTCTT** TTAGGGTTAGGG
ERR2061618	60	1	Yes	many hits	Not on ends	Chr 1,3 , 6	at the start of the sequence there is a AACATTTGGACCATA which is cut off leaving two internal telomere repeats in the denovo	n/a
ERR2061618	3276	1	Yes	1706536	yes	Chr5		**TCATGAACG** AAGGCGTAGGGTTAGGG
ERR2061618	4476	1	Yes	2470349	Yes	Chr 4	Mismatch	**AATTCGACGTTCGG** GTTAGGGTTAGGG
ERR2061618	6820	1	Yes	1797063	Yes	Chr6		**CCTCATCAT** TAGGGTTAGGG

ERR2061618	5024	1	yes	3087198	Yes	Chr2		**TGCTGTCCC** GGTAGGGTTAGGG
ERR2061619	6312	2	Yes	2568138	Yes	Chr1		**AATCCGAGAAG** TAGGGTTAGGGTTAGGG
ERR2061619	6308	2	Yes (rev of 6312)	2568053	Yes	Chr 1		TAACCCTAACCCTAC **TTCTCGGATTT**
ERR2061619	7376	2	Yes (rev of 7380)	2092197	Yes	Chr6		TAACCCTAACCCTAA **CTTATATGCCCGGCCCCA**
ERR2061619	7380	2	yes	2092245	Yes	Chr6	Had a single TAGGG in the genome	**CATATAAG** TTAGGGTTAGGG
ERR2061619	2512	1	Yes	1347909	Yes	Chr 4	Very degraded telomere at the end	**TCGCCAACCTTG** TTTGGGTTCGGGTTAGGG
ERR2061619	2508	1	Yes	1619772	Yes	Chr 6		**AACCTTTTG** TTAGGGTTAGG
ERR2061619	3784	1	Yes (rev of 3788)	3953235	Yes	Chr3		**CGGTAAGGTA** TTAGGGTTAGGGTTAGGG
ERR2061619	3788	1	Yes	3953285	Yes	Chr3		TAACCCTAACCCTAA **TACCTTACCGGCC**
ERR2061619	7808	1	Yes	1477118	Yes	Chr2	Telomere begins at a CGG	**ATAGCCAG** CGGGTTAGGGTTAGGG

ERR2061619	2824	1	Yes	1592553	Yes	Chr5		**GAGCTGTTTG** TTAGGGTTAGGG
ERR2061619	5048	1	Yes	612930	Yes	Chr5		**CAGCCCCAG** CTTGCTGGTTAGGGTTAGGG
ERR2061619	1988	1	Yes	208492	Yes	Chr2		**TTCCGCCCACC** TTAGGGTTAGGG
ERR2061619	2928	1	Yes	1765939	Yes	Chr1		**TTTTAGGTTGG** CGGGTTAGGGTTAGGG
ERR2061619	4676	1	Yes	175140	Yes	Chr4	Matched well despite an internal or degraded telomere	**TCGTGGGCGGCGGGG** TTAGGGTTAGGG
ERR2061619	5396	1	Yes	1990297	Yes	Chr5	Missing a GACGCGGCGGCGGGG sequence in the genome	**CCTGCCGGT** AGACGCGGCGGG
ERR2061619	6532	1	Yes	2870291	Yes	Chr6		**TTTATCGGGCG** GTTGGGTTTGGG
ERR2061619	2480	1	Yes	1580710	Yes	Chr3	An extra GCCCAG was cut from the read that is prese	TAACCCTTAGCCCAG **CCCCCCCTCAAACTTGTTTTT**

							nt in the genome	
ERR2061619	5800	1	Yes	633380	Yes	Chr6		**TTCCAGAGGATGTCG** GGGTTAGGGTTAGGG
ERR2061620	8	1	Yes	Many hits	Yes	Chr 1,3,7,8	Internal Telomere	n/a
ERR2061620	3968	1	Yes	Many Hits	Yes	Chr 2-8	A rich, rDna like,	n/a
ERR2061621	4072	2	Yes (rev of 4068)	2023082	Yes	Chr4		**TGGTCCCATGGGTG** TTGGGTTTGGG
ERR2061621	4068	2	Yes	2023001	Yes	Chr4		CCCAAACCCAA **CACCCATGGGACCATTACTG**
ERR2061621	3532	2	Yes (rev of 3536)	1417738	Yes	Chr2		**GATTCATACCTCGGGG** CAGGGTTAGGGTTAGGG
ERR2061621	3536	2	Yes	1417827	Yes	Chr2		CCTAACCCTAAC **CCTGCCCCGAGGTAT**
ERR2061621	2640	2	Yes	5413369	Yes	Chr4		CCCTAACCCTAA **CCGAACAGCTAG**
ERR2061621	2644	2	Yes (rev of 2640)	5413369	Yes	Chr4		**CAGCTAGCTGTTCGG** TTAGGGTTAGGG
ERR2061621	5096	1	Yes	3540522	Yes	Chr2		**AAGATTCCTC** AATAGGGTTAGGGTTAGGG
ERR2061621	36	1	Yes	1305518	Yes	Chr1		CCCAAACCCAA **CCACCCACCCACAC**

								ATCCTGAC
ERR2061621	1296	1	Yes	many hits	Yes	Chr 2,3,6	Matches with moters, A rich	n/a
ERR2061621	6552	1	Yes	551086	Yes	Chr6		**CCCTGGATCACCTACTC** AGGGTTAGGGTTAGGG
ERR2061621	6200	1	Yes	3321915	Yes	Chr3		**TTGAGTTGAGCGCG** TAGGGTTAGG
ERR2061621	7548	1	Yes	1676791	Yes	Chr6		**ACCCAACTTGG** TTGGGTTTGGG
ERR2061621	8388	1	Yes	275244	Yes	Chr6		**TGCGCTGGCAGG** TTAGGGTTAGGG
ERR2061621	8520	1	Yes	1762003	Yes	Chr 7		**GGGCATCAAAGC** TTCGGGTTTGGGTTTGGG
ERR2061621	1824	1	Yes	Many hits	Yes	Chr3	Internal Telomere, rDNA like	n/a
ERR2061621	5908	1	Yes	266051	Yes	Chr4		**AAGTTAGGCCAG** TTAGGGTTAGG
ERR2061622	2204	4	Yes	2260695	Yes	Ch2	Both 2204 and 2196 face the same direction and end in the same	n/a

							place. They have been both in cluster six which has a size of 4. The other two reads in the cluster didn't match	
ERR2061622	2196	4	yes	2260714	Yes	Chr2	They mapped in the opposite direction of a moter	n/a
ERR2061622	5364	2	Yes	4450103	Yes	Chr 1		CCCTAACCCTAACCCT **CTCCGCACTTAACCGACC**
ERR2061622	5368	2	Yes (rev of 5364)	4450048	Yes	Chr 1		**TTAAGTGCGGAG** AGGGTTAGGGTTAGGG
ERR2061622	7176	1	Yes	1227339	Yes	Chr 7		CCCTAACCCTAACCCG **ACCGACAAGTAGCAG**
ERR2061622	4888	1	Yes	1165853	Yes	Chr5	Genome is missing a CCACCACGC	**GTCTCTACTGGG** CCACCACGCTGGCAAGGG
ERR2061622	2380	1	Yes	2178893	Yes	Chr3		**CAGGGACGAGGTG**

								TTAGGGTTAGGG
ERR2061622	7560	1	Yes	998998	Yes	Chr2		**GCCACACTCACCGAGG** TAGGGTTAGGG
ERR2061622	2248	1	Yes	3866700	Yes	Chr3		**CCAAGAATGCTTTGCG** TCGGGTTAGGGTTAGGG
ERR2061622	7824	1	Yes	0	Yes	Chr9		**TAATAAAATTACAGTATTAA** TTAGGGTTAGGG
ERR2061622	2304	1	Yes	2353073	Yes	Chr3		**CCCTGCAGCACTTGGAGGAG** TAGGGTTAGGG
ERR2061623	0	1	Yes	Many hits	Yes	Chr 1. 3, 7, 6, 8	Internal Telomere	Internal telomere
ERR2061624	8620	2	Yes	Many hits	Yes	Chr3	A rich rdna like	**ATATTTAATA** TTTAGGG
ERR2061624	6532	2	Yes	Many hits	Yes	Chr3	A rich rdna like	**TTAATA** TTTAGGGTTAGGG
ERR2061624	8780	1	Yes	Many hits	Yes	Chr3	A rich rdna like	**TTTTTGGC** TTAGGGTTAGGG

APPENDIX D: THE FUTURE OF TELOPORTWRAPPER

The code repository for TeloPortWrapper can be found at https://github.com/TrStans606/TeloPortWrapper. Future iterations of TeloPortWrapper will be renamed to DeNovoTelomereMiner to better reflect the function of the program. DeNovoTelomereMiner will be a fully rewritten version of TeloPortWrapper with better compatibility and more stable and standardized code. DeNovoTelomereMiner will also have more features like an installer, a config file to easily change settings, and possibly a GUI settings interface for computers that can support it. Other expanded features will include the ability to filter out any given sequence rather than just the three provided, the ability to search for more genome repeat types, and more control on what reads end up being aligned with the assembled genome. This will broaden the use case of DeNovoTelomereMiner and hopefully solve the issue with improper filtering encountered within this project.