HTStream: A Toolkit for High Throughput Sequencing Analysis

A Thesis

Presented in Partial Fulfilment of the Requirements for the

Master of Science

with a

Major in Bioinformatics and Computational Biology

in the

College of Graduate Studies

University of Idaho

by

David Streett

Major Professor: Barrie Robison, Ph.D.

Committee Members: Samuel Hunter, Ph.D.; Terrence Soule, Ph.D.; Matt Settles, Ph.D.

Department Administrator: Eva Top, Ph.D.

December 2017

Authorization to Submit Thesis

This thesis of David Streett, submitted for the degree of Master of Science with a major in Bioinformatics and Computational Biology and titled "HTStream: A Toolkit for High Throughput Sequencing Analysis," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor: _____Date_____
Barrie Robison, Ph.D.

Committee Members: _____Date_____
Samuel Hunter, Ph.D.

_____Date_____
Terrence Soule, Ph.D.

_____Date_____
Matt Settles, Ph.D.

Department Administrator: _____Date_____
Eva Top, Ph.D.

Abstract

HTStream is a suite of applications developed in C++11 that helps users streamline analysis of High Throughput Sequencing data. Current analysis "pipelines" are often rigid and difficult to use, whereas HTStream was designed to enhance the users' experience. To do this, we introduces a new tab delimited FASTQ format which which allows users to combine applications with the use of simple Linux pipes. This ability creates a paradigm shift toward analysis. This usage of pipes reduces storage space, execution time, since no intermediate files are created. Additionally, this reduction in time and reduction in difficulty while doing analysis creates ease of use for end users while concurrently giving them more power, since integration of de facto Linux tools is straightforward.

Acknowledgements

I would like to thank Joe Angell for his tireless code review, many excellent suggestions, patience and dedication to excellent design. HTStream would not be the exceptional software it is today with his knowledge and care.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 HTStream

Over the last fifteen years, high-throughput sequencing data is becoming cheaper, more accessible, and more essential to research [1]. This means reducing analysis time is becoming more critical. Removing non-essential I/O functions, reducing storage space, and having fast, tested applications is critical to successfully reducing the analysis burden of this ever growing data.

Modern high throughput sequencing machines produce millions of short sequences which, by themselves, are essentially meaningless. Even worse they suffer from a number of sources of noise which can confuse downstream analysis. Properly removing these sources of noise is critical for high quality downstream analysis [2–4].

Herein, I introduce HTStream, a suite of applications developed in C++11 and designed to streamline the preprocessing steps in high throughput sequence data analysis. HTStream was intentionally developed in concordance with recommendations for software development best practices as outlined by Leprevost et al. 2014 [5]. Each application relies on a set of extendible shared libraries for which we have developed comprehensive unit tests, leveraging the Google Test framework (https://github.com/google/googletest). In addition the code is managed through GitHub, and has been subjected to rigorous code review and critique.

The current set of applications include q-window-trimmer (a sliding window quality trimmer), cut-trim (a static trimmer), super-deduper (a reference-free PCR duplicate removal tool), phix-remover (a contaminant read remover tool), overlapper (a paired-read merging tool), adapter trimmer (a paired end adapter removal tool), polyATtrim (a poly A/T tail trimmer), and n-remover (an ambiguity base remover tool).

Each application uses a shared library, which minimizes the code base to increase maintainability of the code, reduce the chance of bugs, and to take advantage of a standardized

$$Q = -10log_{10}(p) + 33$$

Figure 1.1: Quality Score Calcuation
These are how quality scores are derived, where Q is the quality score, and p is the probability the bases is called incorrectly

data format that enables low-overhead streaming communication between tools, making pipeline construction simple and straightforward. In order to achieve this, we have transitioned away from the FASTQ format, and instead represent data as a tab delimited table. This tab delimited FASTQ format allows tools to skip the creation of non-essential intermediate files, significantly reducing storage requirements and lowering server I/O load, while enabling cheap parallelization between tools, seamless integration of paired-end and single-end reads into preprocessing pipelines, and easier integration with de facto Linux tools. This not only has very tangible time and storage savings, it also enhances user experience with standardization across tools, formats, and in creating pipelines.

1.2 High Throughput Sequencing Data

High throughput sequencing (HTS) data in this context refers to data produced by Illumina sequencers, which are the most established platforms [6]. The current nucleotides support by Illumina HTS data are A, C, T, G, and N (ambiguity base). Each base call in a sequence will have a corresponding quality score (q-score). The q-score is an ASCII character that corresponds to the likelihood the base being called correct. Figure 1.1 shows the equation for this [7].Since it is ASCII encoded, numbers are roun

If the quality score is a #, that means, the likelihood of that base being incorrect is approximately 68%. ASCII quality scores do not go above 40, and are always above 0. These nucleotides and quality scores are stored in FASTQ. FASTQ is the de facto "raw" HTS data format [8]. Fastq format can contain two types of information: paired end (PE) data or (SE) single end data. PE data contains sequences from both ends of a DNA molecule

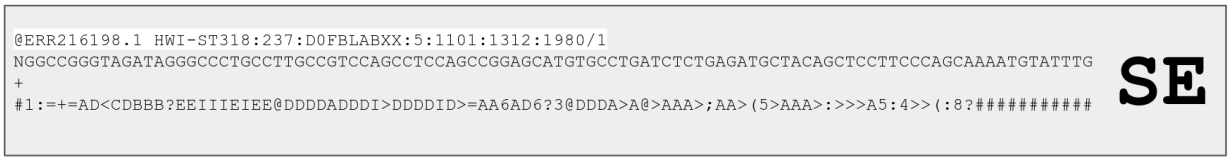

Figure 1.2: Single End Read Example
This is the orientation and template of a single end read.

Figure 1.3: Paired End Read Example
This is the orientation and template of a paired end read.

whereas SE data is done from one end of a DNA molecule. format is as follows - an ID line, the sequence, a plus sign (+), and the quality score, which are all contained on different lines, seen in Figure 1.2 and 1.3. A PE file will have two FASTQ files associated with it, where corresponding reads are on the same relative line in the two files, seen in Figure 1.3. An interleaved FASTQ file, means PE reads are stored in one file - with lines 1-4 being read one, and lines 5-8 being read two as seen in Figure **??** Interleaved. A SE read, will be stored in one file in a standard FASTQ format with only one file Figure 1.2 [7].

1.3 N-Removing Background

Under conditions where the Illuimina base caller is unable to determine which based was incorporated during a flow cycle, an "N" will be reported, representing an ambiguity base. Ns can produce issues in downstream analysis because some software applications are incompatible with them, resulting in a software crash, or in reads with Ns being ignored or the N arbitrarily being changed to a different base [4]. This can be a significant frustration during analysis, can produce unexpected loss of information or create biases in the data which is difficult to detect.

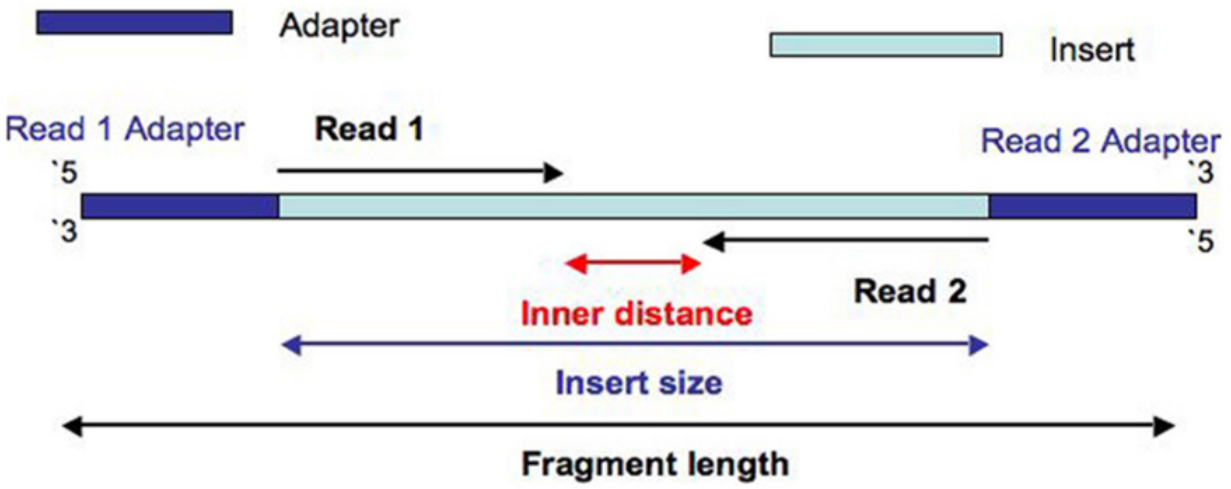

Figure 1.4: No Overlap Example
This is a DNA template that shows the setup of a non-overlapping insert (NOLIns). Image Credit: Turner et al. [9]

1.4 Overlapping and Adapter Trimming Background

In HTS data, paired end reads can overlap in a variety of ways depending on the insert length of the template DNA molecule. Insert length refers to the length of the DNA sequenced (Illumina Sequencing Manual, n.d.). Based on the this length, are classified (PE) reads into 3 different categories: No overlap long insert (NOLIns), long insert (LIns), short insert (SIns). Nolins mean the insert length is greater than the sum of read one and read two lengths (the reads do not overlap). As seen in Figure 1.4, this means the reads are far enough apart where no overlap will happen between read one and read two. Lin reads are quantified by the insert length being less than sum of the two reads length, but greater than the length of the shorter of the two reads lengths. As seen in Figure 1.5 (A), this means part of the reads will overlap. Sin reads are quantified by the insert length being less than the short of the two reads. As seen in Figure 1.5 (A), sins must sequence into the adapter of the read.

Sin reads are a special case of overlaps that applications such as AdapterRemoval [10], PEAR [11], Trimmomatic [12], use for additional quality control. If a sin read is present, the

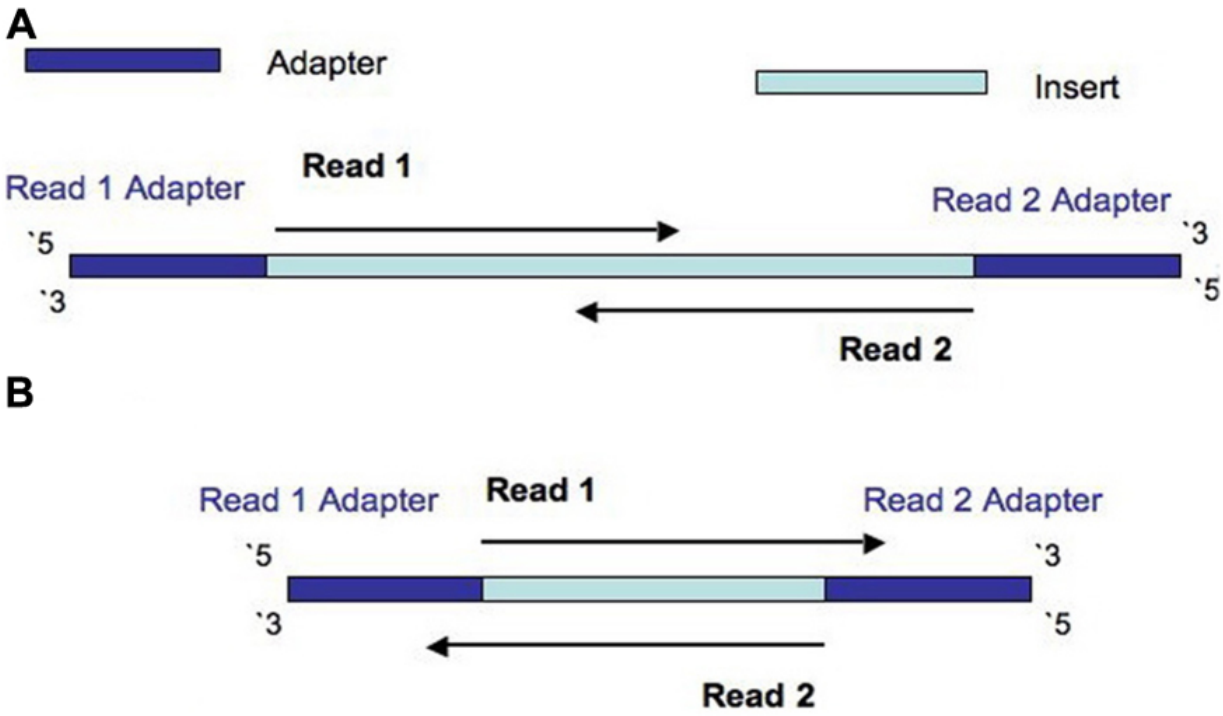

Figure 1.5: Overlap Example
These are two DNA templates that show the setup of a long insert (A) and a short insert (B). Image Credit Turner et al. [9]

edges of the read must contain adapter sequences - which are sequences that are added during library preparation. These adapters can cause inaccurate variant calling, assemblies, and mapping, since they are not derived from the sample, but can confuse tools used for analysis, making the removal of these adapters important for downstream analysis [4]. Olgorithms that usse overlapping regions include, FLASH [13], CASPER [14], and PANDAseq [15]. All overlappings are important because Lins can be used to longer reads help the assembly of genomes [16].

1.5 Phix Removing Background

Contaminate removing is an important part of quality control of HTS data [3]. Contaminates can cause inaccurate final results of analysis (in assembly, differential RNA expression and SNP calling). A common contaminant is Enterobacteria phage phiX (phiX), since it is typically added into Illumina Sequencing Runs to add sequence library diversity, as well as act as a quality control measure [17]. Introduced phiX can result in accidental inclusion of phiX contigs in de novo assemblies, and inaccurate mapping statistics if not removed.

A common approach to contaminant removal is a mapping based approach [**?**]. If the contaminate sequence has a reference, it is possible to map reads against the reference and retain only reads that cannot be mapped. To do this, read mapping tools such as Bowtie2 [18] or BWA [19] are often used. These tools produce a BAM [20] formatted file which can then be processed to extract the unmapped reads. Because the file contains all mapped and unmapped reads it can be quite large, resulting in a significant computational investment and an impediment to the analysis process. Additionally, few dedicated tools exist to resolve this problem, and where solutions do exist, they often involve home-brew scripts or strategies which are slow or cumbersome to include in analysis.

1.6 Poly A/T Tail Removal Background

Polyadenylation is a common feature of mRNA transcripts in eukaryotes. This set of multiple adenosine monophosphates is added to the transcript as part of the processing associated with producing a mature messenger RNA, and results in HTS reads containing stretches of A or T bases. RNAseq library preparation methods sometimes take advantage of this polyadenylation to specifically select for mature mRNA transcripts and avoid sequencing ribosomal RNAs which typically present as a majority fraction of the RNA in eukaryotic cells [21, 22]. Unfortunately, although beneficial for sequencing library preparation, these stretches of homopolymers present a challenge for analysis of the RNAseq data because some mappers and assemblers are unable to recognize the non-genomic bases and handle them properly [23, 24]. Different mappers and assemblers have been built to take these polymers into account, such as STAR [25], this is not a guaranteed by all mappers. An alternative is to trim these homopolymers as a preprocessing step prior to analysis. The necessity of this operation has been recognized within the field for many years, and has lead bioinformatics tool developers to include algorithms supporting it in a variety of tools, including Lucy [26], Cutadapt [27], and UrQT [28], and many others.

1.7 Quality Trimming Background

Reads sequenced on Illumina sequencers have an estimated probability of error associated with each base call within each read [29]. Typically, the ends of the reads have lower quality, with increasing quality in the middle of the read. Most quality trimming applications take advantage of this to remove sections of the read that. are error prone and may have a negative impact on de novo assembly, variant calling, and high confidence mapping of reads against a reference. A large number of quality trimming applications have been developed, some of these include UrQt [28], sickle [30], ngsShoRT [31], NGS QC Toolkit [32], FASTx Toolkit [33], Trimmomatic [12], cutadapt [34], SelexaQA [35]. These application use a variety

of strategies to clean and filter reads; some use a window trimming algorithm, while others simply filter and remove low quality reads. A common theme among the plethora of quality trimming tools is the ability to set parameters in order to trim reads according to the specific requirements of the study. For example, in a study where data is plentiful, it might be beneficial to trim reads aggressively, or even discard a large fraction of the reads in order to reduce noise and improve the results of the analysis. In other situations where data is limited, better results may be obtained by only trimming adapters and the poorest quality bases.

1.8 Cut Trim Background

Because reads vary in quality, and are generated from template sequences of varied length, few quality trimming algorithm guarantee a constant length unless specifically instructed to do so. One example is Trimmotatic which allows the user to set a desired length with the CROP argument [12]. Depending on downstream analysis, constant length across all reads may be required. A notable example of an application that requires constant length read lengths is STACKS, which is a suite of tools commonly used in RAD sequence analysis and population genomics [36]. This tool only supports datasets where all reads are of the same length. In some other cases, library design may incorporate a synthetic barcode sequence into the read, making it necessary to trim a specific number of bases prior to analysis.

1.9 PCR Duplicate Removal Background

Many types of Illumina sequencing libraries require PCR amplification of DNA material as part of the library preparation [37]. This amplification serves to increase the quantity of DNA available for sequencing to insure proper clustering and a successful sequencing run. However, this causes a technical artifact resulting in DNA from the same parent strand being sequenced multiple times. Some “PCR free” library strategies attempt to ameliorate this

problem. However, because of the amplification step common to all Illumina platforms during cluster generation, optical duplicates [38] can still arise. A second class of optical duplicates has recently been identified in the most recently released Illumina sequences (those using patterned flow cells) which further exacerbates the issue of duplicates being generated [39, 40]. It is standard to remove all but one of these PCR duplicates to increase the quality of downstream analysis and remove biases associated with representing a single parent template multiple times in the sequenced data [41]. The impact of not removing duplicates is most notable in variant calling [42]. Applications such as FASTQuniq [43], SAMBLASTER [44], Samtools rmdup [45], and Picard Markduplicates [46] have been been developed to address this problem. Tools attempt to solve the problem of identifying and removing duplicates in two distinct ways. Tools such as Picard MarkDuplicates SAMBLASTER, samtools rmdup, require that reads are first mapped against a reference, and then infer duplicates based on mapping location. Other tools attempt to identify duplicate using raw read similarity to identify duplicated reads and avoid the necessity of mapping to a reference sequence. In the case of marking duplicates in already-mapped reads, all reads are retained, however duplicates are excluded from further analysis. Alternatively, when duplicate reads are identified prior to mapping, the duplicated reads can be discarded, speeding the mapping process and reducing storage space requirements while providing the same benefits. Picard Markduplicates is arguably the de facto tool to for identifying duplicates, especially in human sequencing data. However, it requires a reference genome since mapping location is required, and because of this it is a poor choice when no reference sequence is available.

CHAPTER 2

Methods

2.1 HTStream

HTStream is a suite of applications developed in C++11 and designed to streamline the preprocessing steps often required for high throughput sequence data analysis. HTStream was intentionally developed in concordance with recommendations for software development best practices as outlined in On best practices in the development of bioinformatics software [5]. Although HTStream introduces a handful of innovations for preprocessing Illumina data, a number of applications already exist which are able to address most of the issues commonly encountered during analysis. Instead of trying to make significant improvements on any one of these specific issues, the objective of HTStream was to address a separate, but no less significant problem, which is the existing difficulty in doing exploratory analysis using the hodgepodge of tools and pipelines that are currently available. In HTStream, we attempted to implement high quality solutions to common preprocessing problems in an integrated, flexible way which facilitates exploring the problem space by allowing the user to rapidly implement or modify a preprocessing pipeline to fit the specific needs of a dataset while simultaneously reducing computational load.

We accomplish this task by using the fundamental, yet powerful linux principle of pipes and streaming. Streaming data with current FASTQ formats is extremely difficult (sometimes impossible) because different data types are incompatible in the same file, and like data is spread through multiple files. Instead of creating complexity and rigidity in HTStream by using existing formats, we use a new tab delimited FASTQ format. We use a tab delimited FASTQ format. This format allows us to stream data simply by putting a pipe between two applications. This piping is possible because one line of data represents a grouping of reads (either PE or SE). If a SE read is present, the tab delimited file will have 3 columns (the id, the sequence, and the quality score) in that particular line. If a paired end read is

present then the file will have 5 columns will be present (the id, read one's sequence, the read one's quality score, read two's sequence, and read two's quality score) in that particualar line. This new format creates a different way of viewing analysis of HTS data. This new format allows us to reduce analysis and storage space on machines, since reads are sent directly from to application instead of having to be written to and then re-read from the hard disk. Not creating these non-essential intermediate files, significantly reduces storage requirements and lowers server I/O load, and enables cheap parallelization between tools. Not only are there very tangible storage and time savings benefits, the end user experience is greatly enhanced. The end user no longer has to keep track of reading and writing files from application to application, which can be tedious and frustrating. Additionally, reading and writing of files creates rigidity in the end users' "pipelines". If a user would like to reorder an anlysis pipeline, they must make sure the file names match up in the new application pipeline. If the end user also does not need to look at multiple files to see all the data, since this format allows seamless integration of paired-end and single-end reads.

Since this format can leverage the power of Linux pipes it creates easy integration with de facto Linux tools such as awk, tr, sed, grep, and others. A simple, yet important example is searching through paired end reads for a specific sub sequence in a paired end FASTQ files. A user would have to grep file one and two with string of the subsequence. The user would then have to collect that sequence, as well as the ID above that sequence, and the following two lines (the '+' and the quality score). In addition to that step, the corresponding lines from the other file would have to pulled out and kept track of. This becomes extremely difficult with just the command line. In all likelihood a small python or PERL script would have to be created. Once these reads have been gathered, they would have to be written to the hard disk and the following analysis can finally be called.

This same analysis is easy with HTStream, and shows how powerful Linux tools can be integrated into directly into a pipeline. A user could pipe tab delimited reads from an application, directly into grep, which would output all sequences containing this subsequence

to standout, that could be collected and processed by the next application. There is no need for speciality scripts, intermediate files, or multiple file handles. Continuing with this example, the traditional FASTQ command is multiple lines, and would have to contain an additional script file that could get forgotten when sharing the analysis. HTStream on the other hand is stored all in one line, and relies on applications within HTStream and an ubiquitous Linux tool. This means ease in reproducibility of analysis.

HTStream also offers the benefit of experimentation with analysis. Pipelines are often rigid and removing, adding, or changing the order of analysis can mean significant "book-keeping" of input and output files. Simple spelling and ordering mistakes can cause the entire pipeline to break, which can be frustrating and tough to diagnose from the end user's persepective. HTStream offers the benefit of not keeping track of files, since applications are connected together with pipes. This means applications can be removed, reordered, or deleted with minimal effort. This ease of use encourages users to trying different combinations of applications to see what produces the "best" result.

Figuring out what pipeline is the "best" is difficult, and often requires looking at meta-data. Different tools have a variety of different ways to output metadata, and custom parsers have to be created to gather (if meta data is even reported on). All HTStream applications have standard and custom statistics outputted by default. All statistics are outputted to a JSON formatted file, which any HTStream application can write or append to. This gives the user the option for all metadata, from all applications in the pipeline, to be stored in one file, and parsed simply.

Though it is best practice with HTStream to stream data, we understand that not all tools and analysis support this. In addition to reading and writing (to both the hard disk and stdin and stdout) tab delimited FASTQ files, HTStream can read and write (from/to the hard disk), interleaved files and standard FASTQ files. Additionally, it can write out in unmapped SAM. Not only are there a common set of statistics, and a common set of handled formats, there is also a common set of parameters used in every application of

HTStream. This is because HTStream relies on a set of extendible shared libraries. Each application uses a shared library, which minimizes the code base to increase maintainability of the code, reduce the chance of bugs, and to take advantage of a standardized data format. This standardization of tools helps end users feel more at ease using any tool after just using and learning one.

All code in HTStream, including this shared library, have comprehensive unit tests, leveraging the Google Test framework (https://github.com/google/googletest). In addition the code is managed through GitHub, and has been subjected to rigorous code review and critique. Modern boost libraries and standard template libraries are also leveraged in for speed, general readability, and best practice. All design and code implementations were designed for end user experience. Standardization of formatting, parameters, and statistics was designed to make the user feel comfortable using all tools in HTStream. Streaming analysis removes frustrations and tediousness of dealing with traditional FASTQ files. The shared code base and unit test creates a level of trust and confidence with the users, where they know the applications are doing what they should be doing. All these things create a more productive, faster, and enjoyable analysis for the end user.

2.2 N-Remover Algorithm

Here, were introduce the application "n-remover". N-remover removes ambiguity bases (N) from HTS reads. N-remover uses a simple, O(N) complexity algorithm where N is the length of the read. N-removers goes through the entire read and returns the largest sequence that does not contain an N. The algorithm loops through every base in the sequence and keep tracks of last location of N, the current location, and the locations of the best stretch of Non-N sequences seen so far. The algorithm will check each base pair in order and see if the base pair is an N. If it is an N, the algorithm will check to see if that stretch is longer than the previous stretch. The algorithm will continue until it reaches the last base in the sequence. The algorithm will check to see if the end to the previous N seen is the longest

non-N containing sequence

2.3 Overlapper and Adapter Trimmer Algorithm

Here, we introduce the applications "overlapper" and "adapter-trimmer". These both uses an algorithm that overlaps PE reads, increases confidence in the overlapped region, and removes adapters. This overlapped region is the seen in Figure 1.5 in the blue insert region.

Both overlapper and adapter trimmer uses the same overlapping algorithm. The algorithm looks at the two reads and takes the longest read (in the case where both reads are the same length, read one will be selected). The longest read will be broken into k-mers based on user input (kmer size and kmer overlap). Kmer size will determine what k size is, and kmer overlap will determine what distance between the kmers. These kmers are stored in an unordered map with the key being the kmer and the value being location in the read. The second read (the shorter of the two) is be reverse complimented. This reverse complement of the second read will get a kmer profile of the 5 and 3 prime ends. If a kmer is seen in both the edge of the second read, and in the first read (stored in an unordered map), an overlap is attempted. If the kmer hit in the second read is on the 3 prime end, the read is a sin, otherwise, it is a lin. The overlapped section of the two read each base will be compared against each other. If the bases are the same, the q-score will be modified to the summation of the two bases. If the bases are different, the base with the highest q-score will be selected (in the case of a tie, read one's base will be selected). The modified q-score will be the subtraction of the higher q-score and the lower q-score. In addition, this mismatch will increment the mismatch number. Once a percentage threshold is met of mismatches, the overlap is no longer considered valid and the attempted it exited. The algorithm will continue to attempt all kmers until all kmers are tried, or a valid overlap happens.

The difference between adapter-trimmer and overlapper is the outputted reads. In both overlapper and adapter-trimming, if a read overlaps, adapters will be trimmed (if a sin), and bases and q-scores will be modified (in the overlapped region). In overlapper, the reads

will be merged into a single end read. In adapter-trimmer, reads will be kept as paired end reads. The usage of either of these tools depends on downstream analysis. If a single end read is handed to either of these applications, it is ignored, unless it is under the minimum length, then it will be discarded.

2.4 Phix Removing Algorithm

Here, we introduce the application "phix-remover". Phix-removal algorithm is based upon comparing kmers within a read, to the kmers seen within a reference. The program creates a lookup table (an unordered set) by breaking up a sequence (default sequence is phiX), into kmers of a given size (default is a 11-mer). We then map the base pairs within this kmer to binary using the encoding seen in Table 2.1. One thing to note, this encoding does not allow for ambiguity bases (N's) to be within the kmer of interest. Kmers that have N's within them will be ignored.

	bp	binary
1	A	00
2	T	11
3	G	10
4	C	01

Table 2.1: Binary Encoding of Base Pairs
A maps to 0 (00), T maps to 3 (11), G maps to 2 (10), and C maps to (01). The base compliments are not's of each other. Note: N's have no mappings and are ignored.

Both the forward and reverse complement binaries of the kmer are made and the larger number of the two are added to the lookup table. For example, Table 2.2 shows what the table would look like with a simple reference sequence of "AGGATCA" make with a simple 5-mer.

The observed hit ratio (Figure 2.1) is then compared to the user input. If the observed hit ratio is above the threshold ratio, it is considered a match and will not be written, otherwise, the read is not considered a match and is written out.

	5-mer	Binary Encoding	RC Binary Encoding	Selected Encoding
1	AGGAT	0010100011	0011010111	0011010111
2	GGATC	1010001101	1000110101	1010001101
3	CATCA	0100110100	1110001110	1110001101

Table 2.2: Binary Encoding of Sequence
The sequence AGGATCA final kmer binary encoding

$$ObservedHitRatio = \frac{hits}{length(Read) - length(kmer)}$$

Figure 2.1: Phix Hit Ratio
This is how the hit threshold is derived for phix-remover.

Our this phix-remover tools is a fast, memory efficient, and accurate application that removes contaminates from HTS data, specifically phiX and other contaminates with small genomic sizes.

2.5 Poly A/T Removal Algorithm

Here, we introduce the application "polyATtrim". This poly A/T tail remover algorithm uses a simple O(N) algorithm that checks the ends of the reads for either poly-A or poly-T tails. The algorithm checks bases on both the 3 and 5 prime ends of the read and work inward. The algorithm checks if the base is either an A or a T. If the bases is not an A an A mismatch variable will be incremented. If the base is not a T a T mismatch variable will be incremented. The algorithm will store the last location of an A or a T. Once both the A and T mismatch variables are over the maximum mismatches allowed, the algorithm will trim at the location of the last A or T - if that length is above the minimum required.

2.6 Quality Trimming Algorithm

Here, we introduce the applicaton "q-window-trimmer". This quality trimmer removes low quality bases from the ends of the read. It uses a simple O(N) algorithm that loops

through the quality score to remove low quality bases.

Q-window trim uses a sliding window approach to remove low quality data. The sliding window will start on one end of the read's and look at each bases quality score. The quality score seen will be added into window summation as each quality score is seen. Once the window length reaches the maximum allowed size, the quality score from window length ago is subtracted from the window sum. Before that point, the window length is treated as the number of quality scores the algorithm has seen.

The algorithm looks at the average quality score in the window (q score summation / window length). If this average quality is above the threshold, the read is trimmed prior to the last base look at.

	Position	Window Sequence	Window Quality	Mean Quality	Action
1	1	G	#	2	Trim
2	2	GA	##	2	Trim
3	3	GAT	###	2	Trim
4	4	GATA	####	2	Trim
5	5	GATAC	####(	3	Trim
6	6	ATACA	###((	4	Trim
7	7	TACAG	##(((	5	Trim
8	8	ACAGA	#(((@	10.8	Trim
9	9	CAGAA	(((@@	16.6	Trim
10	10	AGAAT	((@@@	21.4	Keep
11	11	GAATG	(@@@@	26.2	Keep
12	12	AATGA	@@@@D	31.8	Keep
13	13	ATGAT	@@@DD	32.6	Keep
14	14	TGATT	@@DDD	33.4	Keep
15	15	GATTG	@DDDD	34.2	Keep
16	16	ATTGC	DDDDD	35	Keep
17	17	TTGCA	DDDDD	35	Keep

Table 2.3: Quality Trimmer Example
Here is a step by step process of how q-window-trimmer's algorithm works with a sequence of "ACGTTAGTAAGACATAG" and a quality sequence of "DDDDDD@@@@(((####".

Table 2.3 shows a simple walk through of the algorithmic steps.

2.7 Cut Trim Algorithm

Cut-trim is a constant O(N) trimming algorithm. It should be used instead of q-window-trimmer to remove a constant amount of base pairs from the 3' and 5' end. Cut-trim will remove a constant number of bases from the ends of the reads (based on user input) and return that sub sequence.

2.8 PCR Duplicate Remover Algorithm

Here, we introduce "super-deduper". Super Deduper uses the binary kmer strategy described in the Phix-Remover algorithm (Table 1 and Table 2). Instead of using perfect overlapping kmers, though, Super Deduper uses a single kmer from the start of both reads in PE, and a single kmer in SE reads. These kmers are at a set location for each read, creating a binary id for that read. Each read is processed and the binary ID is added to an unordered map, along with the read. If another read comes along with the same binary ID, several things can happen. If the read's summation of its q-score is higher than that of the previous read (with the same ID), the read with the larger summation q-score will replace the smaller one. If the new read is greater than the q-score threshold (based on user input), the read is automatically written out. In this case, the unordered map only stores the binary ID and the read memory is freed with no other read with that binary ID considered. Lastly, the read could be a lower summation q-score and does not replace the read. Once all reads are processed, all reads in the unordered map are written out (unless they have already passed the q-score threshold).

2.9 Meta-statistics

Each application will generates informative statistics. These statistic are output in a JSON format. The structure of the outputted file is a key of the application with an

underscore followed by the PID of the applications, and a value of a dictionary (which is full of the statistics). Each application will write out statistics in a JSON format, and can append to previous JSON files. An entire pipeline of applications can write to the same output statistic file showing what order analysis was ran, if any biases were introduced into the data, and keeping the data all in one place. These statistics include the input and output counts for total number of reads, total number of single end reads, and total number of paired end reads.

CHAPTER 3

Results

3.1 HTStream Profiling

HTStream has many benefits including lack of intermediate files and no unneeded disk I/O operations. Though this time comparison is not a comparison against other similar tools, it is a comparison of processing data of different apps with intermediate files, versus piping results to each other. We used the command /usr/bin/time -v (app) to get time and memory usage for each program. User time and maximum resident set size were used to get time, and memory usage.

3.2 N Remover Validation

To validate n-remover was working properly, a small python script was developed. The python script split a randomly generatored read on N's. The script then sorted the split values in the list, longest to shortest. The longest sequence (or the first entry in the list) was compared to the result of n-remover. They matched perfectly analyzing 10 million randomly generator reads when comparing the python script and n-remover.

3.3 Overlapping and Adapter Trimming Validation

We took 25 M paired end reads of phiX and did sensitivity and specificity analysis. We used ground truth as bwa mapping data. Since phiX is a circular genome, any mappings within the sum of read one length and read two lengths are not considered (abouot 20 PE reads). A true positive are reads that overlaps with our overlapping algorithm and by its mapping location. A true negative are reads that do not overlap with our overlapping algorithm and do not overlap based on mapping location. A false positive is a read that overlaps with our overlapping algorithm but does not overlap based on mapping locations.

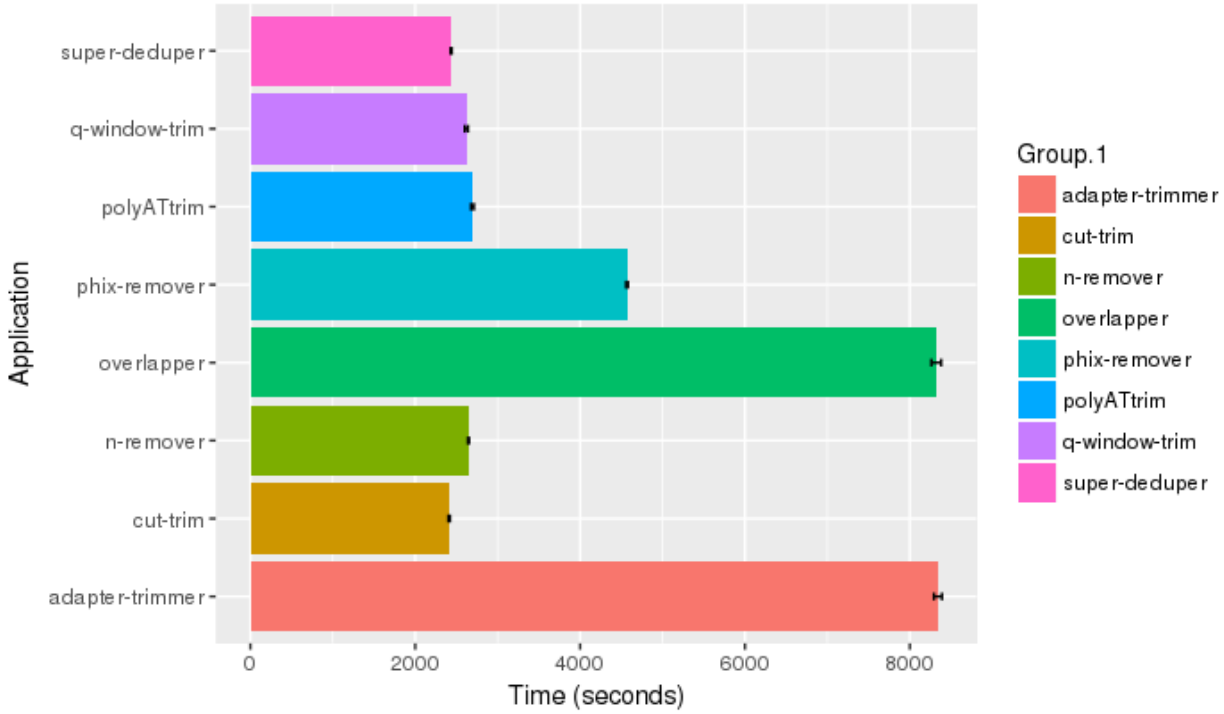

Figure 3.1: Individual Application's Processing Time
Each application in HTStream was ran against approximately 300 million reads and the time was collected.

A false negative is read that does not overlaps with our overlapping algorithm, but does overlap based on the mapping locations.

Sensitivity with defaults was high with 96.68% and a specificity of 96.16%. Most that were false positives and false negatives had low quality and mapping scores. Reads that were true positives were then used to run a T-test. Overlap length was compared against calculated insert length derived from mapping locations. To calculate mapping lengths, the lin algorithm is as follows. If read one is in a reverse orientation compared to read two - read length is equal to the sum of the left most position of read one and the length of read one's sequence minus left most position of read two. If read two was in a reverse orientation compared to read two - read length is equal to the sum of the left most position of read two and the length of read two's sequence minus left most position of read two.

The sin algorithm follows the same structure, however, requires the CIGAR string to look at soft clipped ends. Soft clips are indicative of Illumina adapter sequence (which is a

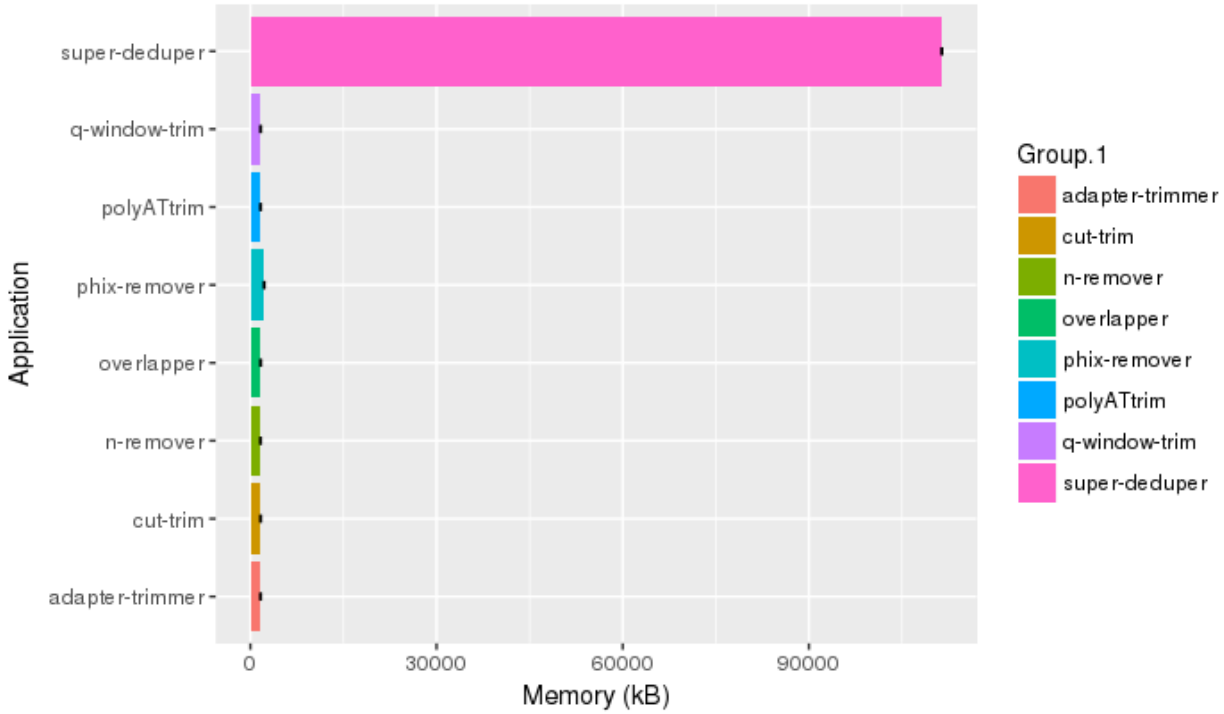

Figure 3.2: Individual Application's Memory Overhead
Each application in HTStream was ran against approximately 300 million reads and the RAM used was collected.

technical artifact). If read one is in a reverse orientation - soft clips should happen on the 3' end of read one and the 5' end of read two. If read two is in a reverse orientation, soft clips should happen on the 5' end of read one and the 3' end of read two. Soft clips size should be approximately the same size on each end. If soft clips are seen on both size, the maximum soft clip size is taken and subtracted from the insert length calculated for lin.

Results of the Welch Two Sample t-test showed a p-value of less than 2.2e-16, with a mean overlapper insert length of 365.40 and mean calculated insert length of 365.66.

3.4 Phix Removal Validation

To test phix removal we took 10 million single end reads from a PhiX data set (reads that properly mapped to phiX), 10 million simulated single end human reads (simulated from the latest version of the human genome with wgsim). We then ran specificity sensitivity analysis

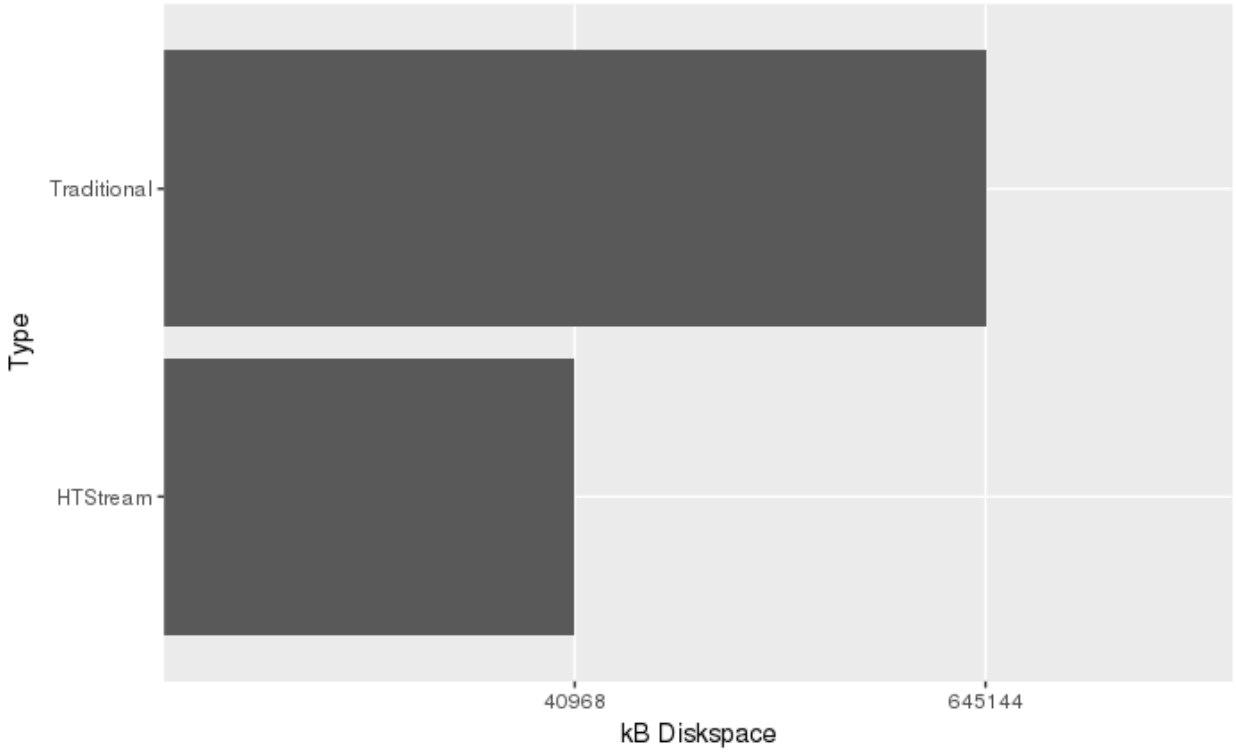

Figure 3.3: Traditional Versus Streaming Storage Space
Overall storage usage of traditional analysis versus streaming analysis (27 million reads)

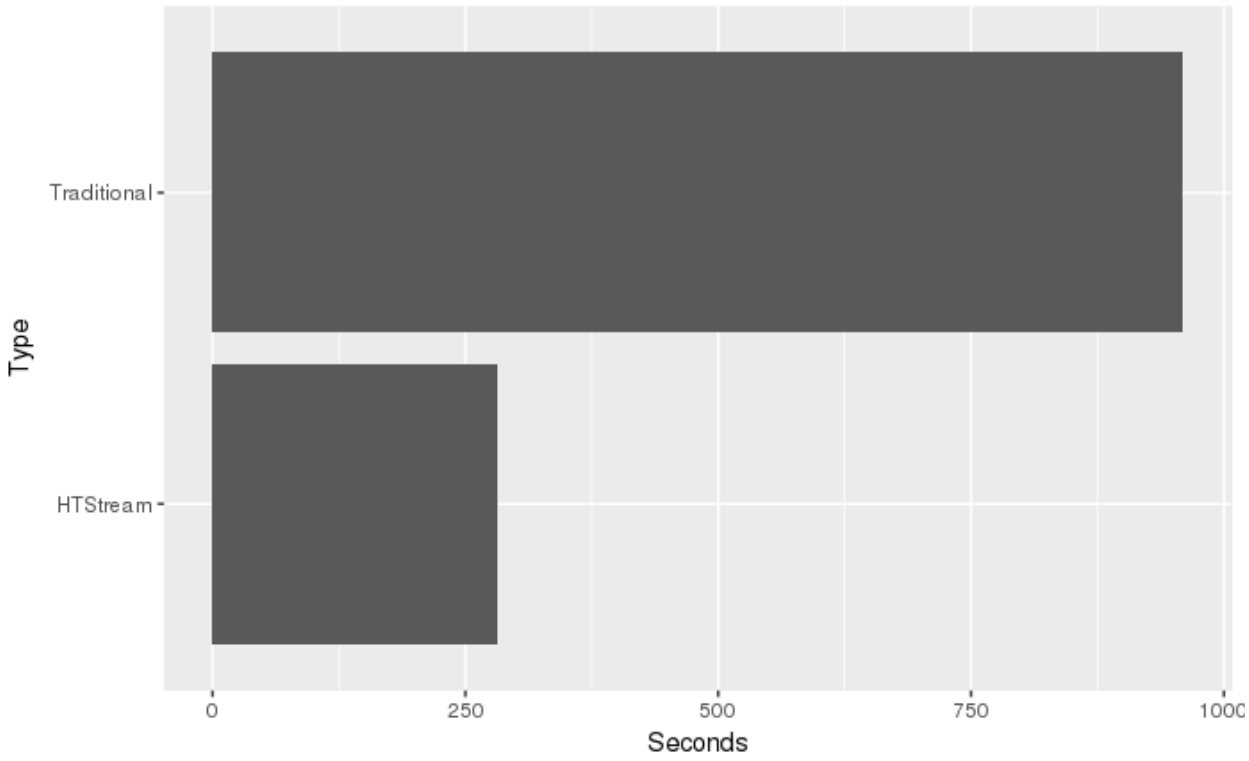

Figure 3.4: Traditional Versus Streaming Processing Time
Processing time of traditional analysis versus streaming analysis (27 million reads).

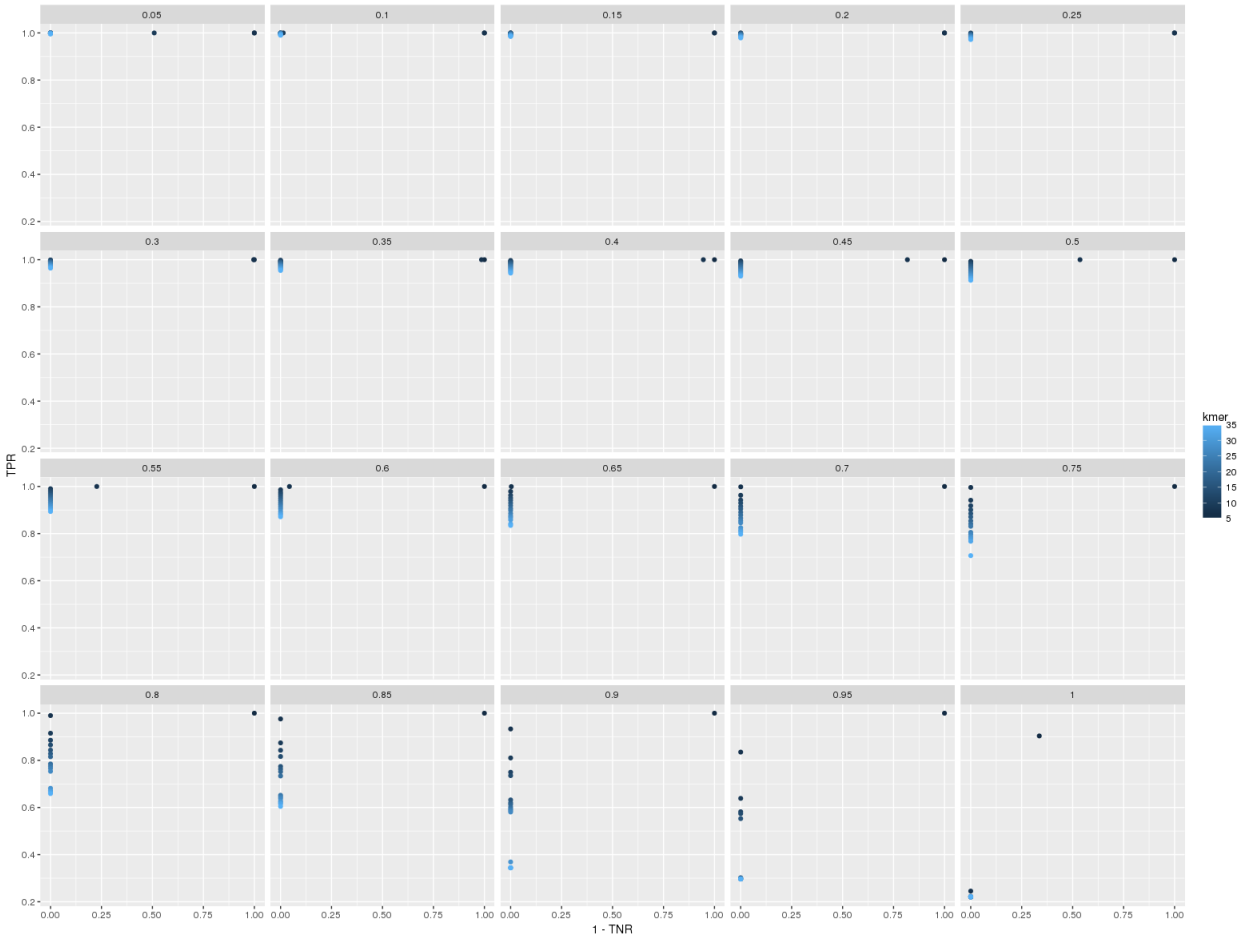

Figure 3.5: Phix ROC Curve
Phix was ran with varying length and percent hit threshold to determine the optimal parameters. Each gray box is a different percentage ran with every kmer 2 through 35, skipping two.

on the resulting data set. A phiX read that is removed is considered a true positive. A human read that is removed is considered a false positive. A phiX read that is not removed is considered a false negative, whereas and a human read that is not removed is considered a true negative.

3.5 Poly A/T Removal Validation

To validate poly AT tail trimmer, we analyzed all reads from a RNA seq experiment where read one had at least a poly A or poly T region at the start or end of the read at least

5 bp long. We then ran polyAT-trim on this dataset on approximately 1.5 million SE reads. As seen in Figure 3.6 there, removal of these poly AT tails normalized the distribution of nucleotides.

3.6 Quality Trimming Validation

The following analysis was done with approximately 25 million reads sequenced on a Miseq. Only Read 2 was ran (since read two typically has a lower quality). Figure 3.7 shows coverage per call for different kmer length. Figure 3.8 shows mean quality score per call site with different kmer trim lengths. For all analysis the quality threshold is a q score of 20, since that is a standard quality threshold.

The same analysis was done comparing sickle and q-window trimmer, since they have a very similar algorithm. Sickle was run under default conditions and q-window-trimmer was run with a threshold q score of 20 and a window size of 50. The depth per call and average quality score was compared between raw reads, sickle, and q-window-trim.

3.7 Cut Trim Validation

A small Linux/Unix script was developed to validate that cut-trim was working properly. 10 million randomly generator reads were created and ran through both the script and cut-trim. The results were exact matches.

3.8 Super-Deduper Validation

A de facto PRR and optical duplicate removal application is Picard's Markduplicates application. This is because Markduplicates uses mapping locations of reads, hence duplicate detection is easier. Both Markduplicate and super deduper behaved the same when validated against perfect simulated data. With both behaving this way with a simulated data, we

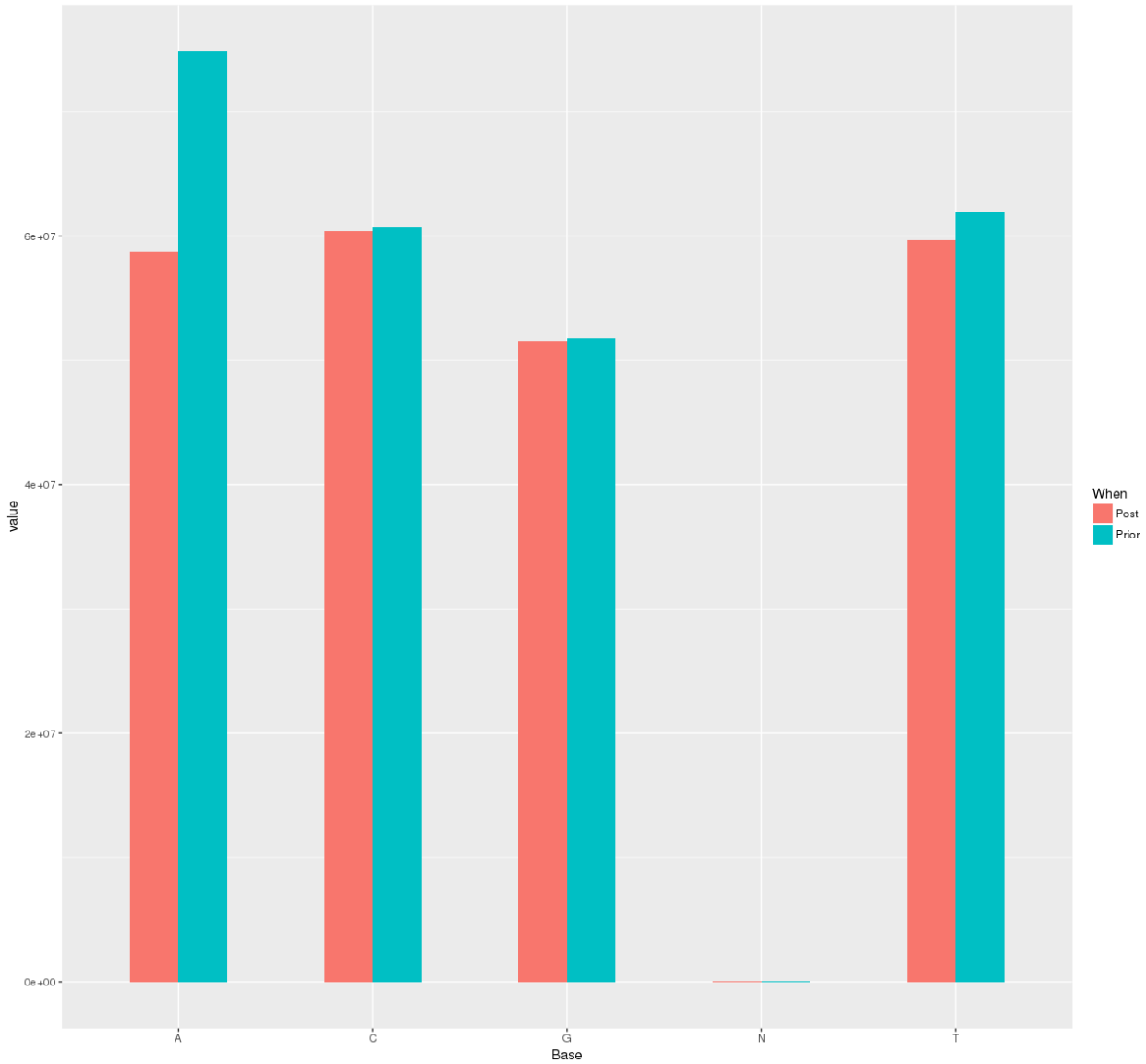

Figure 3.6: Poly A/T Nucleotide Distribution
Nucleotide distribution pre and post Poly A/T tail trimming.

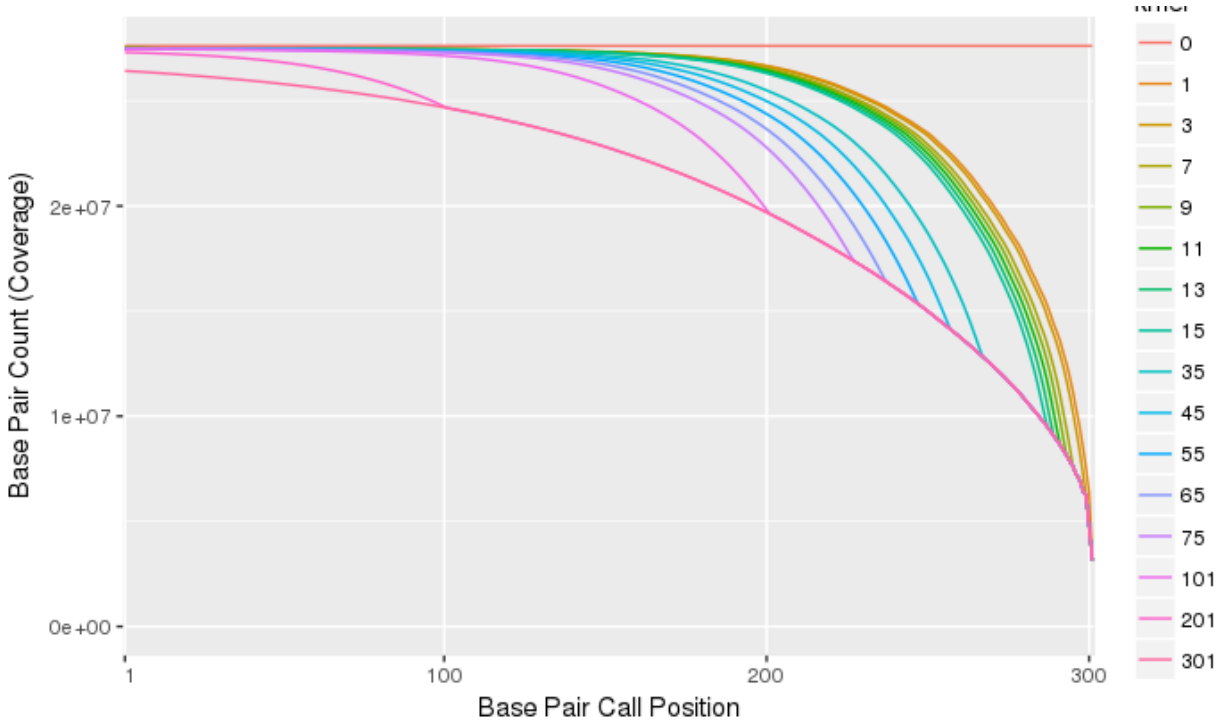

Figure 3.7: Quality Trim Call Site Coverage
The coverage per call site with different kmer windows. Only 5' trimming was used (left trimming), though default is both 3' and 5' trimming.

moved on to real data. Since Markduplicates is a program that is used often, we use its results as the "ground truth" in these experiments.

We used a targeted human dna capture data set which is available on the NCBI SRA [27] with the accession number of SRR1304297. After cleaning and filtering for low quality reads (these reads cleaned with q-window-trimmer with a window length of 50 and a minimal acceptable length of 50, and trimmed from only the 5' end), there were approximately 5 million properly paired, properly mapped reads. We took these reads and ran them through Picard Markduplicates as well as super deduper under various parameters. We then looked at the resulting outputted reads from both of them to determine true positive and true negative rates. We refer to a duplicate as the grouping of duplicates. This means the DS and DI tags (arguments TAG_DUPLICATE_SET_MEMBERS) for duplicates and the PG flag for singletons in SAM/BAM format. For super-deduper, super-deduper-debug (on the SD-DEBUG branch) was used to determine duplicate groups. Note, duplicate is not in the

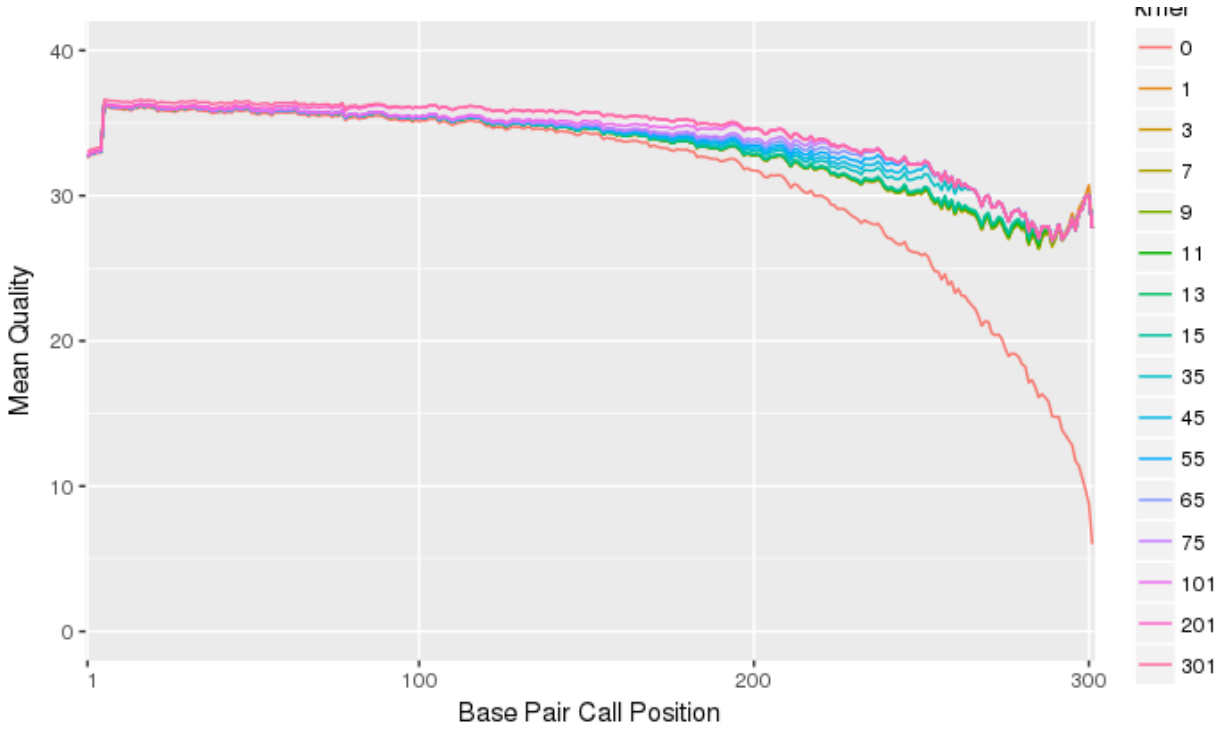

Figure 3.8: Quality Trim Mean Quality
The mean quality score per call site with different K-mer trimmings. Only 5' trimming was used (left trimming), though default is both 3' and 5' trimming.

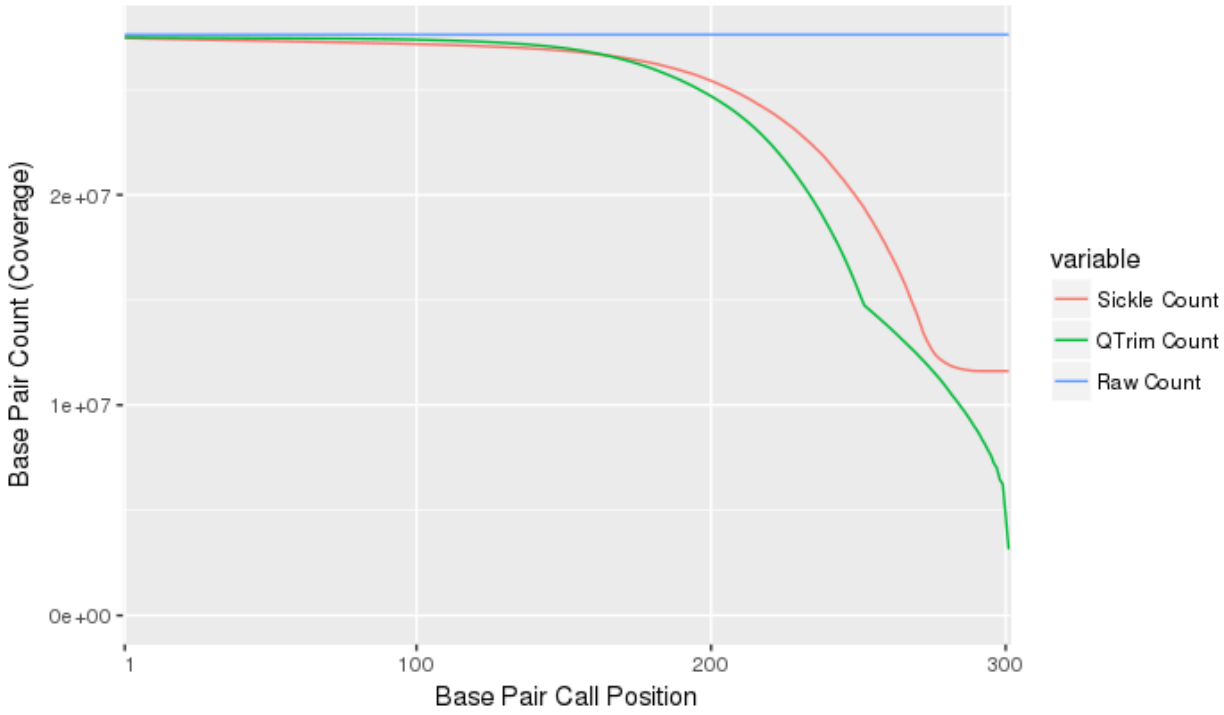

Figure 3.9: Quality Trim Versus Sickle Coverage
The coverage per call position between sickle, q-window-trim (with a window length of 50), and raw reads. Each application uses a quality threshold of 20.

bitwise field in the sam/bam format in this case (bit 0x400). Markduplicates will use this field to mark all duplicates, minus the one with the highest sum of quality score (arguments DUPLICATE_SCORE_STRATEGY; default SUM_OF_BASE_QUALITIES). We refer to a singleton as a read that did not undergo a duplication event.

We then did a sensitivity specificity analysis looking at optimal start and kmer lengths to create the binary kmer ID using markduplicates as the ground truth. A true positive was quantified by both super deduper and markduplicates calling a read a PRC duplicate. A false positive was quantified by both super deduper and markduplicate calling a read a singleton (no duplicate event happened). A false positive was quantified by super deduper calling a read a duplicate, but was a singleton in markduplicates. A false negative was quantified by super deduper calling a read a singleton. We test start locations between 5 and 25 and kmers between 6 and 25.

We found that starting location varied the results little and that it was more critical

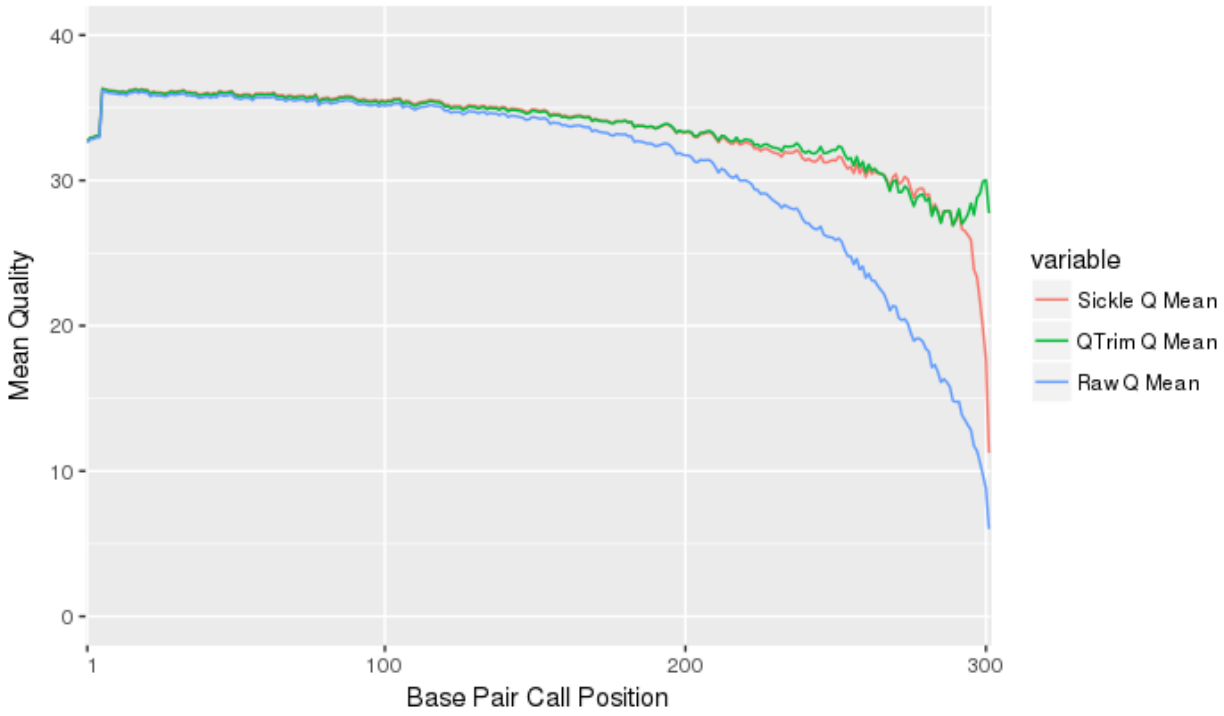

Figure 3.10: Quality Trim Versus Sickle Mean Quality
The average quality score per call position between sickle, q-window-trim (with a window length of 50), and raw reads. Each application uses a quality threshold of 20.

the length of the sequences. Larger binary lengths created higher number of false negatives, while true positives remain relatively constant with varied lengths. We are also interested in the concordance of the outputted reads when comparing markduplicates and super deduper. Mark duplicate will select one read out of each duplicate grouping to represent that duplicate group. Using the parameters of a start location of 10 and a length of 10 (defaults), we saw concordance of outputted reads were fairly high with a TPR of 94.56

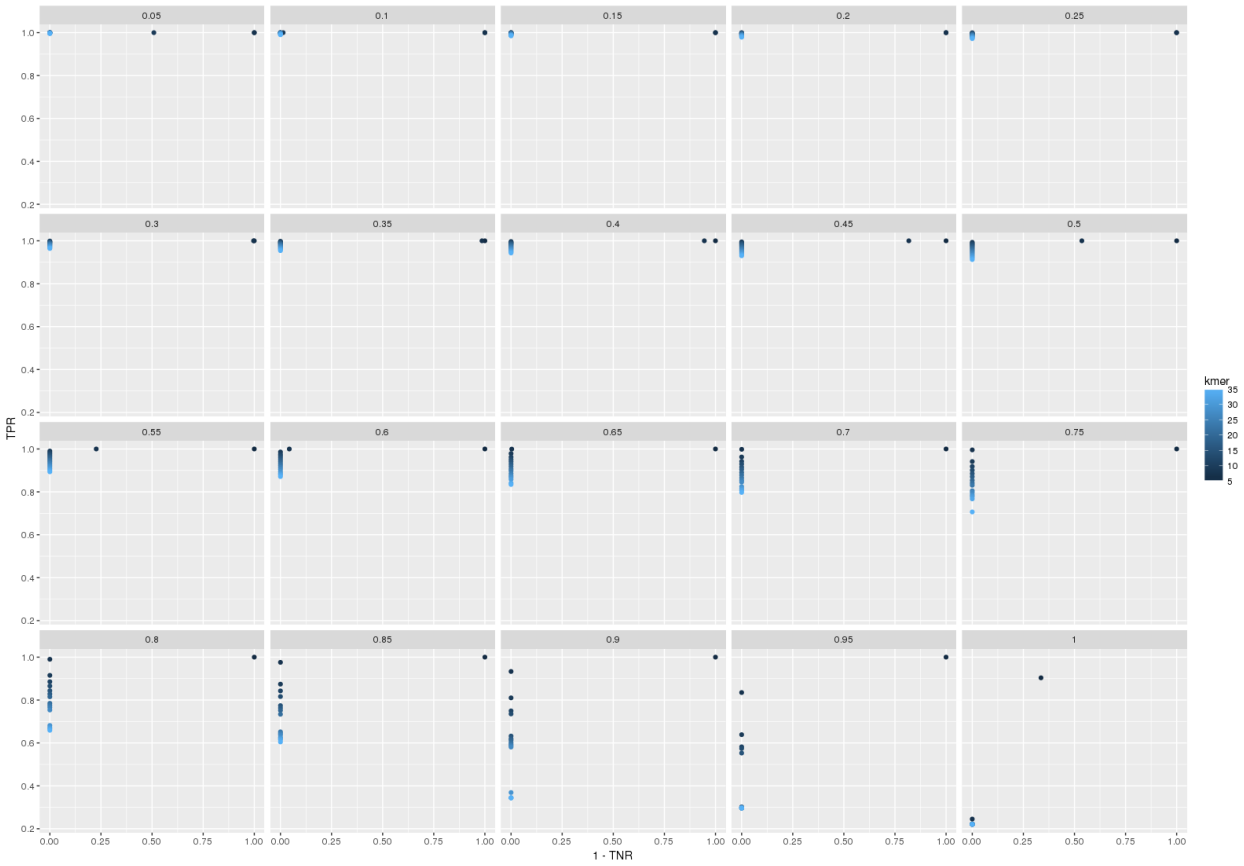

Figure 3.11: Super Deduper ROC Curve
This ROC curve looks at the optimal kmer length and start location of binary ID. Each grouping is the facet of the Binary ID, and there is a colour gradient of start locations.

CHAPTER 4

Discussion

4.1 HTStream

As seen in Figure 3.1 most HTStream applications are fairly quick and a predominantly I/O bound. Some notable examples that deviate from this trend are Phix Remover, Overlapper, and Adapter Trimmer. This is because these applications have to break the read up into kmers and check it against a map. In the case of Phix Remover, the unordered set is created for all the reads. In overlapper and adapter trimmer, an unordered map must be created for each PE read. This creates substantial overhead in this algorithm and makes it relatively slower than the other algorithms. The memory usage for all applications is very low as well (Figure 3.2, with the exception of super deduper. Super Deduper must store all reads in memory to determine which ones are duplicates and which ones are not. This can be a challenge with very large data sets with low duplication rates. Because of this super deduper has an argument that allows reads to be written out immediately, if the read is above a certain quality threshold. The tab delimited format power is shown in Figures 3.4 and 3.3. We see a reduction by about a half in 3.3 of storage space on machines. Most of that memory are non-critical files that could be deleted. In in practice, however, these intermediate files often take up large amounts of space on servers. Figure 3.3 shows the reduction in time from the traditional methods to the streaming method. The savings in on parallelization of the tools and removal of unnecessary I/O.

4.2 N Remover

Creating N remover in python was a trivial task, however, was significantly slower than C++11. N remove as part of HTStream is a fast quality control script that fits into the HTStream toolkit. N remover has been validated with both the python script that tested 10 million reads, but also unit test.

4.3 Overlapping and Adapter Trimming

The sensitivity and specificity of approximately 96 percent seen in real, raw data was a fairly positive result. As stated before, the 4We then looked if the insert lengths of these overlaps were correct. With the t-test we see the predicted mapping results and overlapping/adapter-trimmer results are almost the exact same. The slight variation comes from soft clipping. Soft clipping can be a few base pairs off, if the adapter matches the reference close enough. This means slight variation between overlapper and predicted mapping was expected.

4.4 Phix Removal

With the sensitivity specificity analysis we see large kmers produce high sensitivity (removal of Phix) even at low percentages. This shows the robustness of the kmer approach and the flexibility of kmer size and percentage thresholds. As seen in Figure 3.5, from about 10The phix-remover was given raw data. This means the quality of the some of the data could have been low. Ideally, the kmer approach should be robust enough to still remove this low quality, raw data. That is why we went with a kmer length of 17 and a percentage of 10. We felt it would remove most of the phiX, while not removing any of the non-phix reads. We feel this phiX remover can also be used to remove small contaminants as well. As long as the contaminates are only a tens of kilobases long, this strategy should be fairly successful.

4.5 Poly A/T Removal

As seen in Figure 3.6, the nucleotide distribution post AT tail trimming was normalized. This normalization of these homopolyers helps downstream analysis in mapping and assemblies. Though it might seem like a significant lost of data, the data that was removed was

redundant, non-informative bases.

4.6 Quality Trimming

As seen in Figure 3.7 q-window-trimmer, larger kmers trim more aggressively. Larger Kmer sizes will reduce overall call coverage, however, will slightly improve quality scores as seen in Figure 3.8. In the comparison against sickle (Figures 3.10 and 3.9, we see because of the adaptive window sliding algorithm used in q-window-trim, we guarantee the average q-score will be greater than the q-threshold (if it follows Illumina error profile), whereas sickle does not. This means higher coverage from sickle at the end of the reads, and higher confidence at the end of the reads of q-window-trim.

4.7 Cut Trim

Cut trim was validation and unit test all behaved as expected. Cut trim is an utility program that allows for more applications to be used in downstream analysis.

4.8 Super Deduper

As seen in Figure 3.11, 8-mer and above offer a high sensitivity and a relatively high specificity up to the 25-mer. Start location impacts the results minimally, with the exception of very small (1-8) start location. This is because start location at the very start of the read tend to be lower quality than the middle of the read, as seen in Figure 3.8.

CHAPTER 5

Conclusion

We have illustrated that HTStream is a fast, high quality, and reliable set of applications for analysis on HTS data. This with its ability to use streaming tab FASTQ format decreases time spent doing analysis, reduces the overall storage, and creates ease of use for the end users.

Bibliography

[1] Thomas P Niedringhaus, Denitsa Milanova, Matthew B Kerby, Michael P Snyder, and Annelise E Barron. Landscape of Next-Generation sequencing technologies. — *Anal. Chem*, 83:4327–4341, 2011.

[2] Chuming Chen, Sari S Khaleel, Hongzhan Huang, and Cathy H Wu. Software for pre-processing illumina next-generation sequencing short read sequences. *Source Code Biol. Med.*, 9:8, 3 May 2014.

[3] Hong Kai Lee, Chun Kiat Lee, Julian Wei-Tze Tang, Tze Ping Loh, and Evelyn Siew-Chuan Koay. Contamination-controlled high- throughput whole genome sequencing for influenza a viruses using the MiSeq sequencer. *Nature Publishing Group*, 2016.

[4] Cristian Del Fabbro, Simone Scalabrin, Michele Morgante, and Federico M Giorgi. An extensive evaluation of read trimming effects on illumina NGS data analysis. *PLoS One*, 8(12):e85024, 23 December 2013.

[5] Felipe da Veiga Leprevost, Valmir C Barbosa, Eduardo L Francisco, Yasset Perez-Riverol, and Paulo C Carvalho. On best practices in the development of bioinformatics software. *Front. Genet.*, 5:199, 2 July 2014.

[6] Jason A Reuter, Damek V Spacek, and Michael P Snyder. High-throughput sequencing technologies. *Mol. Cell*, 58(4):586–597, 21 May 2015.

[7] Illumina Inc. Casava manual 1.8.0. `https://support.illumina.com/content/dam/illumina-support/documents/myillumina/354c68ce-32f3-4ea4-9fe5-8cb2d968616c/casava1_8_changes.pdf`.

[8] Peter J A Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. The sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res.*, 38(6):1767–1771, April 2010.

[9] Frances S Turner. Assessment of insert sizes and adapter content in fastq data from NexteraXT libraries. *Front. Genet.*, 5:5, 30 January 2014.

[10] Stinus Lindgreen. AdapterRemoval: easy cleaning of next-generation sequencing reads. *Lindgreen BMC Research Notes*, 5, 2012.

[11] Jiajie Zhang, Kassian Kobert, Tomá Š Flouri, and Alexandros Stamatakis. PEAR: a fast and accurate illumina Paired-End read merger. 30(5):614–620, 2014.

[12] Anthony M Bolger, Marc Lohse, and Bjoern Usadel. Trimmomatic: a flexible trimmer for illumina sequence data. *Bioinformatics*, 30(15):2114–2120, 1 August 2014.

[13] Tanja Mag and Steven L Salzberg. FLASH: fast length adjustment of short reads to improve genome assemblies. 27(21):2957–296310, 2011.

[14] Sunyoung Kwon, Byunghan Lee, and Sungroh Yoon. CASPER: context-aware scheme for paired-end reads from high-throughput amplicon sequencing. *BMC Bioinformatics*, 15 Suppl 9:S10, 10 September 2014.

[15] Andre P Masella, Andrea K Bartram, Jakub M Truszkowski, Daniel G Brown, and Josh D Neufeld. PANDAseq: paired-end assembler for illumina sequences. *BMC Bioinformatics*, 13:31, 14 February 2012.

[16] Elsa Góngora-Castillo and C Robin Buell. Bioinformatics challenges in de novo transcriptome assembly using short read sequences in the absence of a reference genome sequence. *Nat. Prod. Rep.*, 30(4):490, 2013.

[17] Supratim Mukherjee, Marcel Huntemann, Natalia Ivanova, Nikos C Kyrpides, and Amrita Pati. Large-scale contamination of microbial isolate genomes by illumina PhiX control.

[18] B. Langmead and S. L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, 9(4):357–359, Mar 2012.

[19] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[20] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, Aug 2009.

[21] Wei Zhao, Xiaping He, Katherine A Hoadley, Joel S Parker, David Neil Hayes, and Charles M Perou. Comparison of RNA-Seq by poly (a) capture, ribosomal RNA depletion, and DNA microarray for expression profiling. *BMC Genomics*, 15:419, 2 June 2014.

[22] Dinghai Zheng and Bin Tian. Sizing up the poly(a) tail: insights from deep sequencing. *Trends Biochem. Sci.*, 39(6):255–257, June 2014.

[23] Mihai Pop. Genome assembly reborn: recent computational challenges. *Brief. Bioinform.*, 10(4):354–366, July 2009.

[24] Ana Conesa, Pedro Madrigal, Sonia Tarazona, David Gomez-Cabrero, Alejandra Cervera, Andrew McPherson, Michał Wojciech Szcześniak, Daniel J Gaffney, Laura L Elo, Xuegong Zhang, and Ali Mortazavi. A survey of best practices for RNA-seq data analysis. *Genome Biol.*, 17:13, 26 January 2016.

[25] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 1 January 2013.

[26] H H Chou and M H Holmes. DNA sequence quality trimming and vector removal. *Bioinformatics*, 17(12):1093–1104, December 2001.

[27] Rasko Leinonen, Hideaki Sugawara, Martin Shumway, and International Nucleotide Sequence Database Collaboration. The sequence read archive. *Nucleic Acids Res.*, 39(Database issue):D19–21, January 2011.

[28] Laurent Modolo and Emmanuelle Lerat. UrQt: An efficient software for the unsupervised quality trimming of NGS data. *BMC Bioinformatics*, 16(1), 1 December 2015.

[29] Kensuke Nakamura, Taku Oshima, Takuya Morimoto, Shun Ikeda, Hirofumi Yoshikawa, Yuh Shiwa, Shu Ishikawa, Margaret C Linak, Aki Hirai, Hiroki Takahashi, Md Altaf-Ul-Amin, Naotake Ogasawara, and Shigehiko Kanaya. Sequence-specific error profile of illumina sequencers.

[30] FASS Jn. Joshi NA. sickle.

[31] Fernando Garc-Alcalde, Konstantin Okonechnikov, Jos? Carbonell, Luis M Cruz, Stefan G?tz, Sonia Tarazona, Joaqu?n Dopazo, Thomas F Meyer, Ana Conesa, M Roberts, G Marçais, M Pop, J A Yorke, P Hugenholtz, N C Kyrpides, J T Simpson, N A Fonseca, İ Birol, T R Docking, I Y Ho, D S Rokhsar, R Chikhi, D Lavenier, G Chapuis, D Naquin, N Maillet, M C Schatz, D R Kelley, A M Phillippy, and S Koren. Qualimap: evaluating next-generation sequencing alignment data. *Bioinformatics*, 28(20):2678–2679, 15 October 2012.

[32] Ravi K Patel and Mukesh Jain. NGS QC toolkit: A toolkit for quality control of next generation sequencing data. *PLoS One*, 7(2):e30619, 1 February 2012.

[33] [PDF]FASTX-Toolkit - command line usage - InsideDNA.

[34] Marcel Martin. Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.journal*, 17(1):10–12, 2 May 2011.

[35] Murray P Cox, Daniel A Peterson, and Patrick J Biggs. SolexaQA: At-a-glance quality assessment of illumina second-generation sequencing data. *BMC Bioinformatics*, 11(1):485, 27 September 2010.

[36] Julian Catchen, Paul A Hohenlohe, Susan Bassham, Angel Amores, and William A Cresko. Stacks: an analysis tool set for population genomics. *Mol. Ecol.*, 22(11):3124–3140, June 2013.

[37] An introduction to next-generation sequencing technology. `https://www.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf`.

[38] Nava Whiteford, Tom Skelly, Christina Curtis, Matt E Ritchie, Andrea Löhr, Alexander Wait Zaranek, Irina Abnizova, and Clive Brown. Swift: primary data analysis for the illumina solexa sequencing platform. *Bioinformatics*, 25(17):2194–2199, 1 September 2009.

[39] Steven Wingett. QC fail sequencing illumina patterned flow cells generate duplicated sequences. `https://sequencing.qcfail.com/articles/illumina-patterned-flow-cells-generate-duplicated-sequences/`. Accessed: 2017-7-27.

[40] J. Increased read duplication on patterned flowcells- understanding the impact of exclusion amplification. `http://core-genomics.blogspot.com/2016/05/increased-read-duplication-on-patterned.html`. Accessed: 2017-7-27.

[41] Matthew N Bainbridge, Min Wang, Daniel L Burgess, Christie Kovar, Matthew J Rodesch, Mark D'Ascenzo, Jacob Kitzman, Yuan-Qing Wu, Irene Newsham, Todd A Richmond, Jeffrey A Jeddeloh, Donna Muzny, Thomas J Albert, and Richard A Gibbs. Whole exome capture in solution with 3 gbp of data. *Genome Biol.*, 11(6):R62, 17 June 2010.

[42] Emma M Quinn, Paul Cormican, Elaine M Kenny, Matthew Hill, Richard Anney, Michael Gill, Aiden P Corvin, and Derek W Morris. Development of strategies for SNP detection in RNA-seq data: application to lymphoblastoid cell lines and evaluation using 1000 genomes data. *PLoS One*, 8(3):e58815, 26 March 2013.

[43] Haibin Xu, Xiang Luo, Jun Qian, Xiaohui Pang, Jingyuan Song, Guangrui Qian, Jinhui Chen, and Shilin Chen. FastUniq: a fast de novo duplicates removal tool for paired short reads. *PLoS One*, 7(12):e52249, 20 December 2012.

[44] Gregory G Faust and Ira M Hall. SAMBLASTER: fast duplicate marking and structural variant read extraction. *Bioinformatics*, 30(17):2503–2505, 1 September 2014.

[45] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 15 August 2009.

[46] Broad Institue. picard.