12-13-2017 11:00 AM

Correction of DNA Sequencing Data with Spaced Seeds

Stephen Lu
The University of Western Ontario

Supervisor
Ilie, Lucian
The University of Western Ontario

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science
© Stephen Lu 2017

Abstract

The advent of next-generation sequencing technologies has allowed for the bridging of wet lab work and large data analysis into a cohesive work flow; with the increasing speed and efficiency of sequencing organisms, it becomes imperative that we are able to ensure the data that is produced is correct.

We designed and implemented a new algorithm, QUESS, which based on using multiple spaced seeds to correct DNA sequencing data from Illumina MiSeq, HiSeq and NextSeq machines using C++ and OpenMP for parallel computing. We compared our method with ten leading programs, producing consistently better overall results for most tested measures. QUESS has the best average performance for all programs tested and is also competitive in terms of time and space complexity.

Keywords: DNA sequencing error correction, spaced seeds

Acknowlegements

This thesis has been a product of all of the support I have gotten in the past years. I would like to express my deep thanks for all the people that have made this journey possible. Firstly, I would also like to thank my family for being supportive of all my decisions. Secondly, I would like to thank my colleagues for their insight into many topics that I would not have known prior. Finally, I would like to thank my supervisor, Dr. Lucian Ilie, for his guidance, his knowledge and his patience. Without him, I would not be able to complete this thesis in a timely manner.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Biologic Relevance

Traditional study of evolutionary genetics has taken its roots from Charles Darwin's theory of evolution by natural selection, which is primed on the postulate that traits are heritable and that favourable traits persist through generations due to selection [4]. These organisms that have traits that aid in its survival fare better against its environment versus organisms that do not have these traits. In an extension to this idea, Mendelian inheritance observes that the phenotype, or physical properties of offspring organisms are bound by laws of genetic inheritance that are a function of parental genetics, or genotype (Figure 1.1). Such observations would spur the birth of the field of genetics and bioinformatics, due to the need to study the physical cause of this inheritance.

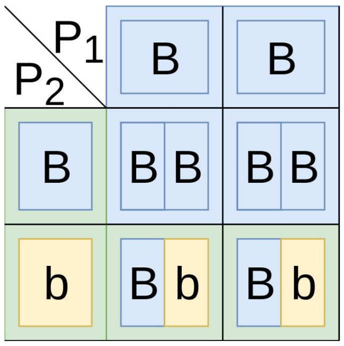


Figure 1.1: **Sample Punnett Square.** The box surrounding each letter is representative of the allele that attributes towards the phenotype, and the background represents the overall phenotype of the specimen. If the genotype for parent 1 **(top row)** is BB resulting in a blue specimen, and the genotype for parent 2 **(left column)** is Bb resulting in a green specimen, the genotype of a resulting offspring can be either BB or Bb, resulting in either a blue or green specimen.

Chemists would come to deduce that this information existed in sequences called deoxyribonucleic acid (DNA) comprised of sub-units called deoxynucleotide triphosphates (dNTP). DNA is synthesized in a process known as polymerase chain reaction (PCR) in the 5' to 3' direction by a biological catalyst (enzyme), DNA polymerase – the 5' denotes the carbon attached to a phosphate group, and the 3' denotes the carbon attached to the hydroxyl group, which is the active region for DNA polymerase – by addition of one of four variants of dNTP: adenine (A), guanine (G), and cytosine (C). Sequencing of DNA was first made possible through Sanger sequencing [16], which relied on the use of di-deoxynucleotide triphosphates (ddNTPs) – a similar molecule to dNTP, but does not elongate DNA – to terminate the DNA early and radioactive tags to identify sequence DNA. Sanger sequencing used four DNA sequencing pools in where each pool conducted PCR, with the caveat that each pool had in addition a ddNTP variant corresponding to each dNTP. As the DNA in each pool replicated, some replication cycles would be cut short due to the appending of ddNTP, resulting in each pool having all DNA strands that end in its corresponding ddNTP. Sanger sequencing could then deduce the sequence through gel electrophoresis – a process in which the DNA is run through a porous gel, causing larger DNA strands to go through slower than smaller faster DNA – as the longer strands take more time to bypass through the porous gel, which would result in radioactive bands corresponding to the relative position of termination in reverse (Figure 1.2 has a sequence GGCAAGCTCGGCGACTCGGCGA). This experimentally derived sequence is called a read.

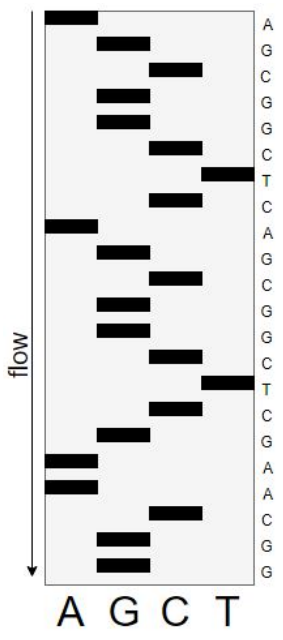


Figure 1.2: **Sanger Sequencing.**

This allowed biologists to observe the presence of genomic sequences, and linked prior observations of parental inheritance to the physical phenomenon that is the difference in DNA sequence. But these technologies were primitive and slow, and the reads produced were short. The need for faster and more efficient sequencing has became a necessity for the growing field of bioinformatics; a shift in interest which resulted in the dramatic drop in cost for sequencing (Figure 1.3), and has effectively shifted the bottleneck of the advance of biology to data analysis [17].

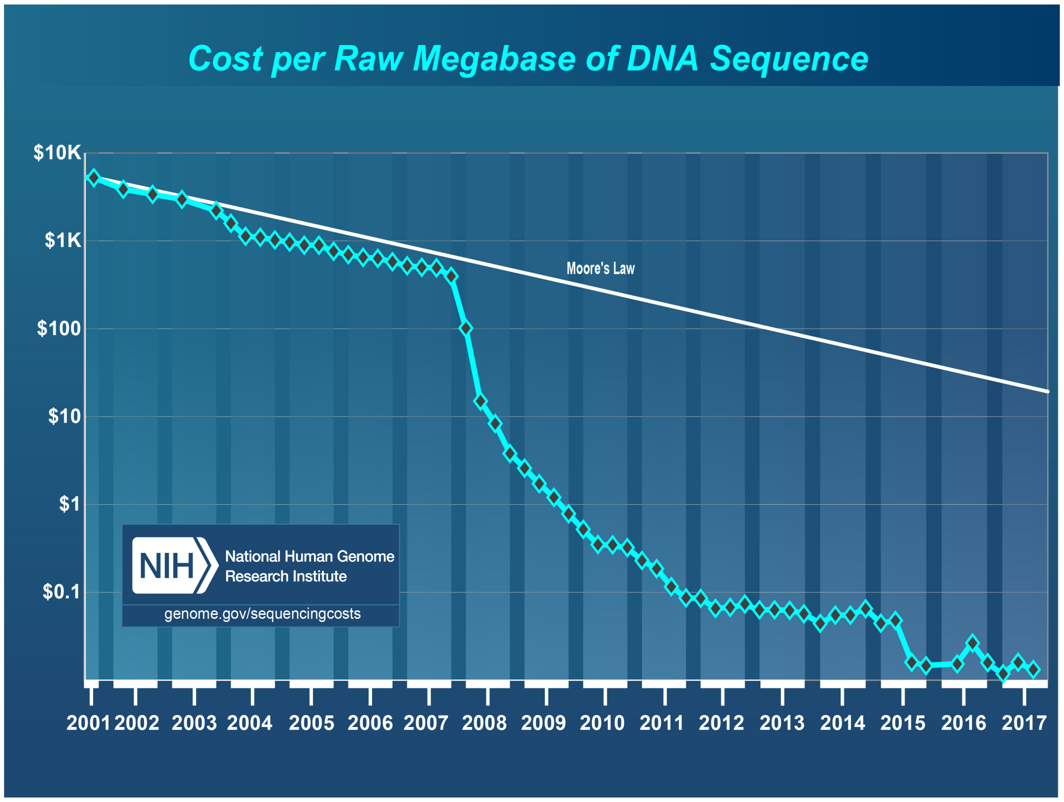


Figure 1.3: **Cost of Sequencing.** The cost of sequencing against time. The cost of sequencing DNA has decreased over the past 16 years. [22]

1.1.1 DNA Structure

DNA contains a subset of four nucleotides (a subset of dNTP): adenine, guanine, cytosine, and thymine, which will represented in literature as A, G, C, and T. DNA exists as a double helix in its native state, in where adenine A binds with thymine T, and guanine G binds with cytosine C. In this double helical state, DNA maintains a reversely complementary scheme in where the sequence on the oppo-

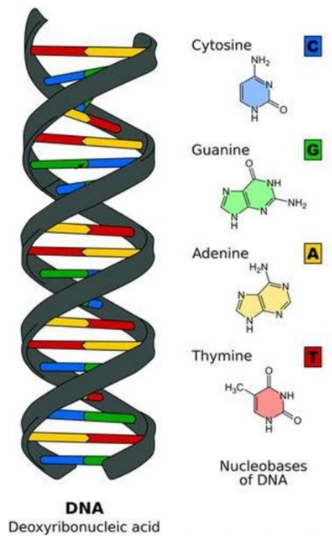


Figure 1.4: **DNA Structure.** [21]

site strand will have the corresponding binding nucleotide in reverse order. This reversely complementary scheme is important as it is the state of DNA which achieves the lowest level of energy – this means that two pieces of DNA that are reversely complementary will bind together – and man-made primers – sequences of DNA that are designed to bind to target segments of DNA – are created to emulate their nature-created counterpart.

1.2 Next Generation Sequencing

The advent of next-generation sequencing (NGS) technologies has allowed for the bridging of wet lab work and large data analysis into a cohesive work flow; with the ability to generate longer reads from test specimen, it becomes a reality to identify the hidden trends that underlie the message that is the DNA. Illumina, the current leader of such sequencing technologies, offers multiple models of machines that produce a range of reads based on target size and price [9]. The HiSeq machines are designed for high output (>15GB), have longer runtimes, and purposed to sequence larger organisms and produce reads typically around 100 base pairs (bp). The MiSeq machines are designed to be benchtop machines, handling smaller jobs (<15GB), have shorter runtimes, and run on smaller organisms producing reads of around 250-300 bp. The newer NextSeq machines are a flexible series of machines designed to be high output benchtop machines, allowing for larger jobs (<120GB), while producing reads around 100 bp.

These machines start by taking sample DNA and breaking into smaller fragments of 200 to 600 bp. Primers are subsequently added, which bind to the 5' and 3' ends of the DNA. The primer-bound DNA is subsequently replicated a few cycles and bound to a flow cell containing obligos – lengths of DNA – complementary to the primer sequences (Figure 1.5). Recall that DNA exists in a native state in where it is reversely complement, and the primer will want to bind to a sequence that is reversely complement, which is the case with the flow cell-bound oligo. These flow cell-bound fragments are used as a template for repeated cycles of PCR, resulting in a forest of identical fragments. The resulting clusters of fragments are then snapshot at every base pair with use of fluorescent tags, allowing for the sequence to be deduced by consensus of each cluster.

These reads are stored within a file that is usually of one or two formats: FastA or FastQ. FastA files contain only reads (Figure 1.6a)– consecutive base pairs that span anywhere from 35 bp to 300 bp – whereas FastQ files contain not only the reads, but also the corresponding quality of each base pair, a measure of certainty of that particular base within the read, which is represented by an ascii value (Figure 1.6b). With respect to computer scientists, these reads are strings of characters from set { adenine, guanine, cytosine, tyrosine } or in short by { AGCT }; the character N is used to represents a failure to determine which of the set that particular base belongs to.

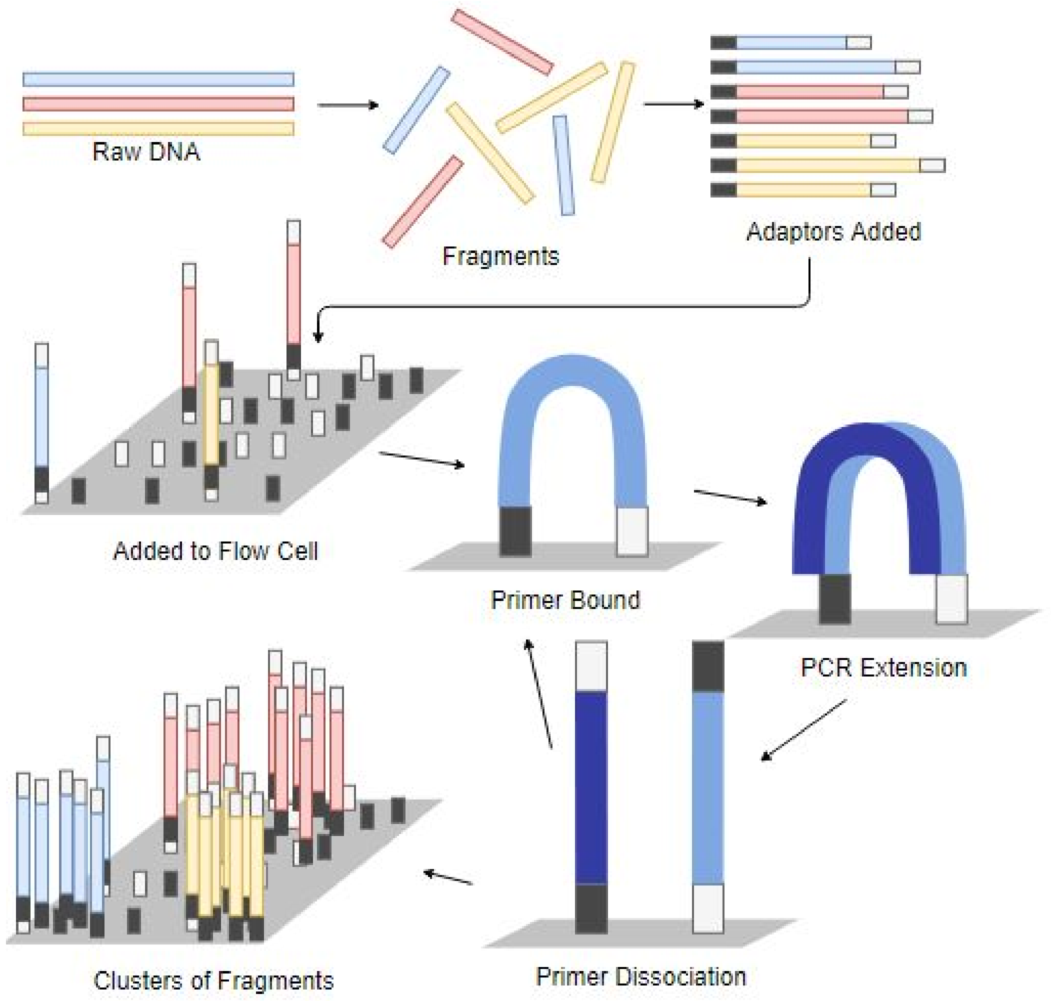
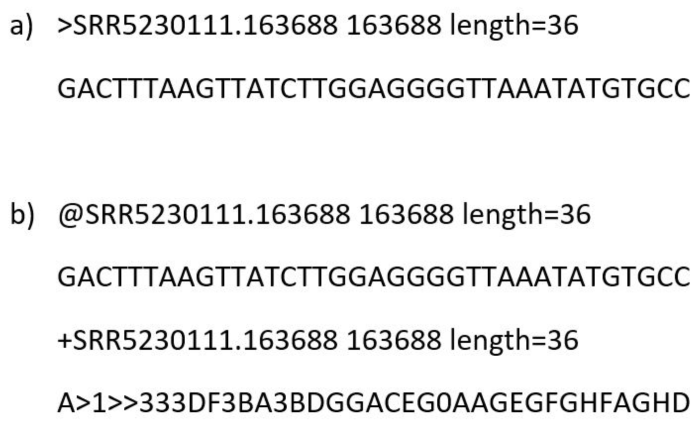


Figure 1.5: **Outline of Illumina genome sequencing process.** Raw DNA is fragmented, and subsequently added with primers. These templates are then added to a flow cell, where primers are added, then the template is replicated and dissociated from the primer. This process of adding the primer, extending the primer, and dissociating the primer is repeated multitudes of times to achieve a forest of reads.

With the aforementioned method of producing these reads, it can be noted that errors can occur during the process, which results in one of three types of errors: insertion errors, in where the product sequence has extra base pairs compared to the true sequence; deletion errors, in where the product sequence is missing base pairs compared to the true sequence; and substitution errors, in where the product sequence has base pairs that are different than the true sequence. It should be noted that with Illumina data, substitution errors predominate [14], as base by base addition of base pairs disfavours insertions and deletions, which are not chemically as stable, leading to premature termination of reads.



```
a)  >SRR5230111.163688 163688 length=36
    GACTTTAAGTTATCTTGGAGGGGTTAAATATGTGCC

b)  @SRR5230111.163688 163688 length=36
    GACTTTAAGTTATCTTGGAGGGGTTAAATATGTGCC
    +SRR5230111.163688 163688 length=36
    A>1>>333DF3BA3BDGGACEG0AAGEGFGHFAGHD
```

Figure 1.6: **Sample Read.** Illumina input read datasets would have thousands to billions of these reads, of longer lengths. **a)** A FastA file, denoted by the $'>'$ at the beginning of every other line. The first line is a comment describing the read (usually relating to its order). The second line dictates the nucleotide base identity, the read. **b)** The FastQ file, denoted by the $'@'$ at the beginning of every fourth line. The first two lines are the same as FastA files, the third line is a comment about the quality of the reads that are present on the fourth line. The quality of the reads is denoted in ascii value.

The resulting product contains thousands to billions of reads that are to be interpreted by the researcher and applied downstream to a number of applications such as *de novo* genome assembly, metagenomics, and DNA-protein interactions. The magnitude of downstream uses implies the need for a stand-alone error correction program that is able to reliably produce the best results; is time and space efficient to integrate into a workflow; and is simple and easy to use to allow for easily reproducible results.

1.3 Error Correction

Error correction is predicated upon the underlying redundancy of data; the machines produce data that is several multitudes larger than the true sequence, and with sufficient coverage – the number of times an input read dataset would overlap the reference genome (Figure 1.7) – many error correction programs can determine and correct erroneous sequences into true sequences, as according to their reference genomes.

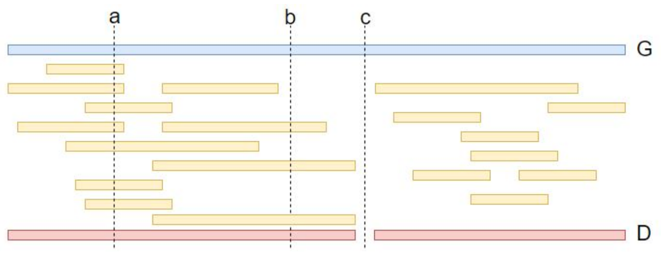


Figure 1.7: **Example Genome Coverage.** **(blue)** The reference genome. **(yellow)** Fragment input read dataset. **(red)** The read breadth for the fragment reads compared to the reference genome. **(a)** An area of high coverage, has a depth of 7. **(b)** An area with low depth coverage, has a depth of 2. **(c)** An area of no coverage. The coverage of the input read dataset **(yellow)** is equal to how many times it overlaps the reference genome **(blue)** on average.

There are multiple trains of thought behind error correction, existing algorithms can be classified into three categories: (i) k-mer counting algorithms, such as ACE [18] and RACER [8]; (ii) k-mer spectrum algorithms, such as BLESS 2 [6], Blue [5], Lighter [20], and Musket [12]; and multiple sequence alignment algorithms, such as BFC [10], Karect [1], and SGA [19]. Each branch of thought has its advantages and disadvantages; k-mer spectrum algorithms tend to use less time and space, but also do not correct as well, multiple sequence alignment algorithms are slow and tend to use up a lot of space and time, and k-mer counting programs can also take up a lot of space due to its counting of k-mers. The end goal of such implementations is to aid researchers to better analyze and interpret their information.

We present QUESS, an error correction program that utilizes multiple optimal spaced seeds to correct errors found within input reads. While traditional error correction programs use continuous seeds of to determine likeness between two sequences between reads, QUESS utilizes multiple spaced seeds, which ignores set positions within the seed, which increases the amount of hits – matches to the database – that each seed incurs [3]. QUESS offers the following advantages: (i) it is the best performer amongst all the aforementioned programs; (ii) it is reasonably time and space efficient compared to the other top performers; and (iii) unlike some k-mer spectrum algorithms, it does not require optimized parameters upon initial runtime.

Chapter 2

Existing Algorithms

The majority of programs utilize k-mers – all the possible substrings of length k that are contained in a string – to correct (Figure 2.1), and categorized into one of three major categories: k-mer counting, k-mer spectrum, and sequence alignment. K-mer counting posits that given multiple variations of a k-mer – a sequence of consecutive base pairs spanning k base pairs – that are not too different from one another, the correct variant of these k-mers is the one with the highest count. In contrast, k-mer spectrum techniques utilize thresholds to ascertain the correctness of a specific k-mer – based on the deviance from the strongest value – usually through two phases; they will tally up k-mers and create a k-mer occurrence frequency histogram and set a threshold and evaluate correctness of k-mers based on this histogram. Alignment algorithms attempt to search for similar k-mers by attempting to bridge the difference between two or more different k-mers into a consensus k-mer in the least costly way. We describe in this chapter nine leading error correction programs.

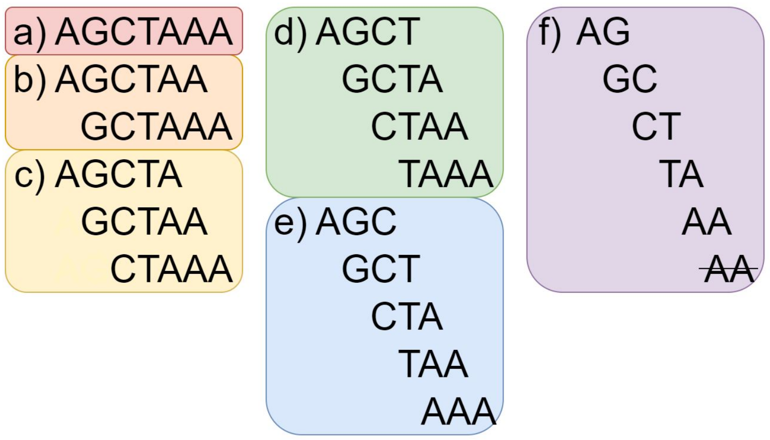


Figure 2.1: **K-mer Example. a)** The original 7-mer. **b)** The 6-mers. **c)**The 5-mers. **d)** The 4-mers. **e)** The 3-mers. **f)** The 2-mers, note that AA appears twice.

2.1 K-mer Counting

2.1.1 ACE

ACE [18] organizes k-mers into tries – prefix trees in where the edge sequence denoted from the parent node to the root node is the prefix of a child node – storing 2-bit encoding of nucleotides { A,G,C,T } with 2 bits { 00,10,01,11 }. Given the ith nucleotide of a k-mer, where $1 \leq i < k$, the ith nucleotide would have been represented as the edge connecting the current node and its parent node and the (i+1)th nucleotide of the k-mer would be represented by the base identity of the (i+1)th nucleotide in binary as an edge linking to a child node (Figure 2.2). The nodes between edge each denote count of i-mers that ACE has encountered.

ACE iterates through two steps: Construct_Subtrie() and Detect_and_Correct(). Construct_Subtrie() applies prefix-based insertion of k-mers from the input read bank to construct the trie where each edge corresponds to the 2-bit encoded identity of a base pair of the k-mer, and each node corresponds to the frequency of that particular variant of the k-mer (Figure 2.2). Detect_and_Correct() detects substitutions from within the sub-tries and attempts to search for alternatives for the branch nodes with the lowest frequencies. ACE mutates the aforementioned branch nodes into all 3 alternative bases for these branches and if such a change results in a variant with a high frequency, it will conduct a 1-change. If such a high-frequency variant is not found, it will conduct a 2-change, in where two bases are mutated, for which nine variants of the low-frequency branch node is tested.

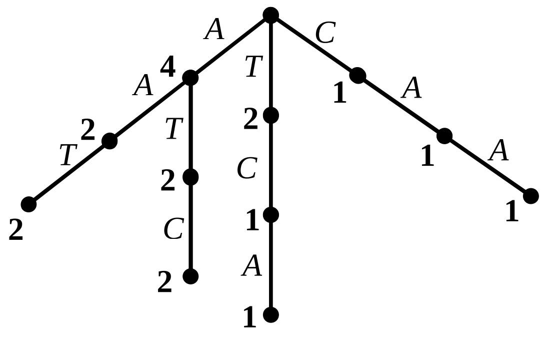


Figure 2.2: **ACE Trie Data Structure.** K-mers are utilized in ACE as a trie, where each edge corresponds to the 2-bit encoded nucleotide base pair, and each node corresponds to the count of the k-mer at that level.[18]

2.1.2 RACER

RACER [8] utilizes 2-bit encoding of nucleotides of reads and represents each k-mer as a 64-bit integer, storing them in a hash table along with their count. A threshold t is computed to discern correct k-mers versus incorrect k-mers. For each k-mer, the flanking nucleotides are denoted as a and b, and these nucleotides are replaced if there exists a variant in the hash table that exceeds the count of the current variant, and is greater than t.

2.2 K-mer Spectrum

Bloom Filters

Most of the best performing k-mer spectrum algorithms utilize Bloom filters, which are probabilistic data structures that are designed to ascertain membership of an input string within a set [2]. To insert a key into the dataset, the key would have be hashed by k hash functions, and the Bloom filter would set the resulting bits in the Bloom filter to 1. To determine if an element is within the Bloom filter, the key would be hashed by these k hash functions, and if all the resulting bits would be 1, it is assumed to be a member within the set defined by the Bloom filter (Figure 2.3). The advantage that comes from use of a Bloom filter is its space efficiency, but it can be subject to false positive matches. All algorithms that utilize Bloom filters are inherently probabilistic due to the nature of Bloom filters erroneously accepting membership of an erroneous entry. This false positive rate can be modeled as $(1 - e^{kn/m})^k$ in where e is the natural logarithm, k is the number of hash functions, n is the number of elements inserted, and m is the size of the Bloom filter.

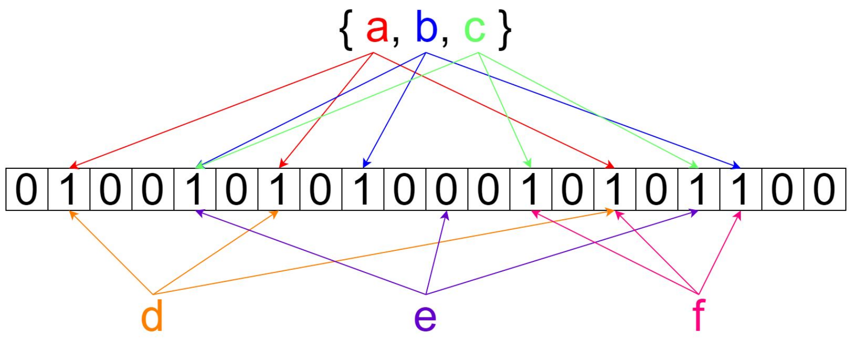


Figure 2.3: **Bloom Filter Example.** Set **{ a,b,c }** hashed into the 1 bits of the Bloom filter (the array of 1s and 0s) using k = 3 hash functions. **d (orange)** belongs in the set due to hashing into bits of 1. **e (purple)** does not belong in the set, having hashed into a bit of 0. **f (pink)** hashes into bits of 1, but is a false positive.

2.2.1 BLESS 2

BLESS 2 [6] is an enhanced parallelized version of BLESS that implements improved methods of detecting weak k-mers. BLESS 2 corrects input reads by first constructing a k-mer occurrence histogram as well as a quality score histogram. K-mers that occur more than a computed threshold are kept as solid k-mers, and these k-mers are used by a Bloom filter by each node to correct the input reads. BLESS 2 corrects weak k-mers and k-mers with low quality scores by cross-referencing them with the solid k-mers.

2.2.2 Blue

Blue [5] uses read context to choose between alternative replacement k-mers. Blue first tiles the input reads to produce a k-mer occurrence histogram, in which it uses to determine what thresholds to use to correct the reads. Then it uses a partitioned hash table to hold consensus k-mers corresponding to the peaks in the aforementioned k-mer occurrence histogram. It corrects its data in the least number of possible fixes such that the resultant fix is a good k-mer, as dictated by the hash table generated prior. Blue calculates appropriate threshold levels based on the harmonic mean of each read rather than the entire dataset, allowing for effective error correction for datasets of lower coverage. TryHealingRead() will subsequently attempt to fix each read from left to right, attempting to fix any k-mers below an acceptable threshold level. Once it encounters such a k-mer it attempts to recursively take the k-mer and perform a depth-first search of possible fixes with FindPlausibleVariants(), and select an appropriate change that is dependent on not affecting the quality of the remainder of the read.

2.2.3 Lighter

Lighter [20] utilizes three passes through multiple Bloom filters to determine the trusted k-mers and transform similar k-mers into those trusted k-mers. In the first pass, Lighter examines each k-mer and stores with a probability of α. Lighter assumes that if a distinct k-mer a occurs N times within the dataset, reduction of α would reduce the survivability of a should it not be a correct k-mer, since correct k-mers are assumed to be more numerous. In the second pass, Lighter stores trusted k-mers into a Bloom filter B, then sets a threshold such that if not enough k-mers overlap a position, it will be marked untrusted. Inversely, if it surpasses this threshold, it will be marked trusted. Lighter then looks for regions of consecutive trusted positions and stores these k-mers within a Bloom filter B. In the third pass, it utilizes a greedy error correction algorithm that looks for the region of longest trusted k-mers that appear in Bloom filter B, and attempts to stretch this region by introducing substitution to the adjacent positions. If such substitutions would result in a change to a k-mer present in Bloom filter B, the position is changed. If more than one candidate is present, the following position will be marked ambiguous and the algorithm continues.

2.2.4 Musket

Musket [12] constructs a k-mer occurrence histogram by counting all non-unique k-mers using both a hash table and a Bloom filter in parallel utilizing a multiple master-slave model, in where the masters are dedicated to task designation, and slaves are dedicated to assigned work. With this model, each master fetches reads in parallel from the input file, and feeds them to their respective slaves, which all hold a local Bloom filter and hash table. Each slave captures the k-mers and performs a membership look up to include within its own Bloom filter. Musket will then repeat this step, but this time, incrementing non-unique k-mers and deleting all unique k-mers. Musket then estimates a coverage cut-off based on this histogram.

When correcting, Musket utilizes three techniques: a two-sided conservative correction, one-sided aggressive correction and voting based refinement. In two-sided correction, bases are classified as trusted or untrusted. If an untrusted base is found on either border of two k-mers, these will be prime targets for correction. One-sided aggressive correction occurs when two or more sequencing errors occur within a single k-mer. For each region of trusted bases, Musket will conduct aggressive replacement with alternative bases to form trusted k-mers, ending only once trusted regions aggregate. Voting-based refinement corrects a base by looking at the number of occurrence of possible fixes, choosing the one with the largest number of occurrences.

2.3 Multiple Sequence Alignment

2.3.1 BFC

BFC [10] utilizes a non-greedy algorithm – in where it explores a greater search space in order to be more accurate – that defines correction as a spectral alignment problem. BFC defines a set of trusted k-mers and attempts to extend the trusted k-mer within a read with such minimal corrections such that all k-mers within the error becomes trusted. BFC utilizes a solution consisting of a 4-tuple of values: the current position i, the last (k-1)-mer W ending at i, the set of previous corrected solutions C, and the penalty associated with the set of solutions p. With each new base, it will retrieve states of minimal penalty from C, and put into context with the new base on a new solution with a new penalty.

2.3.2 Karect

Karect [1] corrects a read by taking all similar reads to a target read r, and stores them in a partial order graph. Karect aligns each similar read in the graph against r and notes the possible changes to similar reads within the graph to recreate r. Each change is noted, and a consensus is derived.

2.3.3 SGA

SGA [19] error correction is based on k-mer frequencies. SGA assumes that k-mers covering a position of error in a read will incur a low frequency of those k-mers. SGA scans a read from left to right to identify problematic bases that yield low frequencies of k-mers that are less than a frequency of c. It will iterate over the other three possibilities of the problematic position, and if such a possibility that exists yields k-mers with frequency c, it will make the change. SGA also stores the entire read dataset using an FM-index – which is a compressed substring index based on the Burrows-Wheeler transform of an input string – to greatly reduce the peak memory usage.

Chapter 3

QUESS: QUality Error correction using Spaced Seeds

3.1 Introduction

QUESS is an error correction program that utilizes multiple optimal spaced seeds to correct errors found within input reads. In this chapter, we will describe in depth the concept of spaced seeds, and give an in-depth explanation as to the mechanisms behind QUESS.

3.2 Spaced seeds

Traditional seed matching is when a region of consecutive bases is matched to a region in the dataset. However, Califano and Rigoutsos posit that use of non-consecutive matches allows for increase chance to find similarities [13] [3]. These spaced seeds have an increased hit – a match from the seed against a sequence within the dataset – rate as opposed to a larger seed of the same length, allowing for a specific seed to be comparing more regions.

In addition, with the introduction of multiple optimal spaced seeds, we can reach Smith-Waterman sensitivity [11], the sensitivity that can be derived by use of the much slower dynamic programming.

a) 10 11 11 00 01 01 11 10 10 00 11 11 01
b) 11 11 01 01 00 10 11 01 00 10 10 11 11

Figure 3.1: **Spaced Seed Example.** **(a** and **b)** Two spaced seeds of seed length 26, and seed weight 16.

A spaced seed is a bit array consisting of 1-bits equal to the seed weight, and 0-bits for the rest of array (Figure 3.1). Compared to other programs which utilize the concept of k-mers – strings of length k – QUESS error correction introduces a new concept called s-mers, which are the strings denoted by the bitwise operation AND between a spaced seed and the input strings. When attempting to match against a sequence of the dataset, the s-mer would only compare bits that have the value 1, and ignore bits with the value 0. QUESS error correction is dependent on use of multiple optimal spaced seeds [7] – each having a seed length of 26 and a static weight of 10-26 – to generate s-mers and their s-gaps, which are the matches against the dataset of the ignored 0 bits. QUESS then derives a canonically correct s-gap, which is the highest count of all variant s-gaps of each s-mer, in which it uses to correct all other similar s-gap variants. QUESS utilizes a seed weight of 16 or 20, dependent on the size of the dataset, which was derived experimentally. Table 3.1 reflects one of many experimental measures used to decide on final seed weight; in this measure, while seed weight of 17 does the best on average across number of seeds equal to 4 and 8, seed weight of 16 does the best on the iteration of number of seeds equal to 8 and does better with all other measures, so it was chosen. It was found that performance due to seed weight variance was correlated to size of dataset, and the final seed weight for the smaller datasets was chosen to be 16, and the final seed weight for the larger datasets was chosen to be 20.

	QUESS ReadDepthGain												
	ReadDepthGain	W15	W16	W17	W18	W19	W20	W21	W22	W23	W24	W25	W26
SRR5230111	K4	28.83	31.17	31.26	29.91	31.31	30.24	31.24	30.21	30.76	29.58	30.30	29.27
SRR5075712	K4	45.58	46.40	46.16	46.13	46.17	45.89	45.72	46.12	45.11	45.35	40.51	40.68
SRR4417428	K4	19.44	20.12	19.68	19.71	19.99	19.59	19.61	19.60	20.01	20.01	19.77	19.76
SRR5181392	K4	25.98	26.08	26.11	26.05	26.09	26.02	25.68	25.68	25.57	25.57	24.89	24.85
SRR5164004	K4	31.20	31.50	31.47	31.39	29.78	29.68	29.58	29.70	29.35	29.38	27.77	27.79
SRR5110368	K4	50.86	60.55	62.77	61.99	62.47	62.24	62.47	62.84	61.73	61.64	62.93	60.68
SRR4423181	K4	79.66	82.19	82.45	82.83	83.56	83.56	83.60	83.60	83.60	83.61	82.07	82.05
ERR1485307	K4	85.15	85.59	85.72	85.30	85.42	85.35	85.44	85.37	85.36	85.32	84.61	84.57
SRR2317595	K4	52.03	52.03	53.14	44.49	44.95	45.00	44.76	53.69	53.37	53.37	51.21	51.37
SRR5146462	K4	67.44	71.33	72.11	72.19	72.38	72.32	72.35	72.36	72.25	72.24	71.04	71.04
SRR5230111	K8	29.44	31.72	31.71	30.80	31.77	30.90	31.75	30.85	31.67	31.70	31.39	31.35
SRR5075712	K8	57.63	57.72	57.57	57.45	57.55	57.29	57.52	57.15	57.10	56.98	54.69	54.70
SRR4417428	K8	0.00	20.25	19.95	19.97	20.25	19.97	19.99	19.99	20.23	20.24	20.08	20.02
SRR5181392	K8	26.66	26.66	26.66	26.64	26.67	26.61	26.55	26.49	26.48	26.46	25.89	25.83
SRR5164004	K8	32.70	32.79	32.66	32.68	32.30	32.17	32.28	32.08	32.09	32.13	30.98	30.92
SRR5110368	K8	61.30	64.46	64.53	64.66	64.56	64.58	64.49	64.56	64.36	64.56	64.06	63.93
SRR4423181	K8	82.69	85.13	85.48	85.62	85.61	85.68	85.66	85.73	85.68	85.73	85.37	85.40
ERR1485307	K8	86.79	86.80	86.79	86.76	86.81	86.78	86.85	86.78	86.78	86.74	85.90	85.82
SRR2317595	K8	62.09	63.40	63.30	55.68	61.09	55.94	56.81	62.93	62.86	62.93	61.60	61.70
SRR5146462	K8	71.77	73.42	73.53	73.54	73.58	73.56	73.58	73.46	73.44	73.41	72.50	72.46
	Average K4	48.62	50.70	51.09	50.00	50.21	49.99	50.05	50.92	50.71	50.61	49.51	49.21
	Average K8	51.11	54.24	54.22	53.38	54.02	53.35	53.55	54.00	54.07	54.09	53.25	53.21
	Average Total	49.86	52.47	52.65	51.69	52.12	51.67	51.80	52.46	52.39	52.35	51.38	51.21

Table 3.1: **QUESS Seed Weight Selection.** Comparison on relative seed weight performance on MiSeq datasets comparing the ReadDepthGain measure. Darker colors indicate better results.

3.3 QUESS - overview

QUESS utilizes multiple spaced seeds to correct errors found within input reads. While established algorithms utilize seeds with consecutive bases, QUESS masks these bases with the spaced seeds to produce an s-mer and an s-gap. With a bank of sufficient s-mers and their corresponding optimal s-gap, derivation of the most correct variant s-gap is obtained and correction of the input dataset is accomplished by comparison of s-gaps of input reads versus the computed correct s-gaps. An overview of the program is given in Algorithm 1.

Algorithm 1 Main Algorithm

```
input: $R_I$, $L$                                   ▷ $R_I$ – input dataset, $L$ – genome size
output: $R_O$                                       ▷ $R_O$ – output dataset
move $R_I reads\ to\ R_T$                            ▷ $R_T$ – temp reads
computeTC( $L$ )
initialize hash table $H$, file $R_{TC}$             ▷ $R_{TC}$ – empty read file
for each seed $S_I$ do
    if $I <= 3$ then
        $T_E = max(\ 2, T_C/4\ )$, $T_{DIFF} = 4$
    else
        $T_E = T_C/2$, $T_{DIFF} = 2$
    insertSMers( $S_I$, $R_T$, $H$ )
    rehashFrequentSMers( $H$, $T_C$ )
    insertSGaps( $S_I$, $R_T$, $H$ )
    removeAmbiguousSMers( $T_C$, $T_E$, $H$ )
    correct( $S_I$, $T_C$, $T_E$, $T_{DIFF}$, $R_T$, $R_{TC}$, $H$ )
    clear $H$
    move $R_{TC}\ to\ R_T$
move $R_T\ to\ R_O$, export $R_O$
```

3.4 Computing parameters

QUESS has two parameters that depend on the specifications of the input dataset. T_C and T_E are threshold values that adjust based on the coverage of the dataset against the estimated size of the reference genome. These values will be important in determine the correctness of the input s-mers and s-gaps and are calculated based on the Poisson distribution, which posits that the probability of any given event occurring within a fixed interval of time or space is a constant rate and is independently determined (3.1).

$$\begin{aligned} P(\ k,\ I\ ) &= e^{-\lambda}\frac{\lambda^k}{k!} \\ P(\ k,\ I\ ) &= \ Poisson\ distribution\ of\ k\ events\ in\ I\ time\ or\ interval \\ e &= Euler's\ number \\ \lambda &= \ the\ event\ rate \end{aligned} \tag{3.1}$$

To calculate T_C, we have to define λ and I, which are function dependent on the average lengths of reads $|r|$, the number of reads within a dataset N, the estimated length of the reference genome L, the estimated rate of error of the dataset ϵ, and the weight of the s-mer w.

To calculate λ_C, we can posit (1-ϵ) as the probability of a base being correct, and (1-ϵ)$^{\mathrm{w}}$ is the probability of a s-mer being correct, $\frac{|r|N}{L}$ as the average coverage of read r, and $\frac{wN}{L}$ as the average coverage of a s-mer. We then can expect that the likelihood of a s-mer being correct as a function of the difference of coverage of $|r|$ versus s multiplied by the likelihood that an arbitrary s-mer of weight w is correct: $\lambda_C = (|r| - w)\frac{N}{L}(1-\epsilon)^{\mathrm{w}}$. In addition, we can express the likelihood that an arbitrary s-mer of weight w is incorrect at one base: $\lambda_E = (|r| - w)\frac{N}{L}(1-\epsilon)^{w-1}\epsilon\frac{1}{3}$. We can then calculate the Poisson (λ_C, T_C) and Poisson (λ_E, T_C) as a function of λ_C, λ_E and T_C (3.2).

$$
\begin{aligned}
&P(\ \lambda_C,\ T_C\ )\ is\ the\ likelihood\ of\ a\ s-mer\ being\ correct\\
&P(\ \lambda_E,\ T_C\ )\ is\ the\ likelihood\ of\ a\ s-mer\ being\ incorrect\ at\ one\ base\\
&\qquad\lambda_C = (|r| - w)\frac{N}{L}(1-error)^{w}\\
&\qquad\lambda_E = (|r| - w)\frac{N}{L}(1-error)^{w-1}error\frac{1}{3}\\
&P(\ \lambda_C,\ T_C\ ) = \frac{e^{-\lambda_C}\lambda_C^{T_C}}{T_C!}\\
&P(\ \lambda_E,\ T_C\ ) = \frac{e^{-\lambda_E}\lambda_E^{T_C}}{T_C!}
\end{aligned}
\tag{3.2}
$$

It can be noted that Poisson(λ_E, T_C) is greater than Poisson (λ_C, T_C) at $T_C = 1$, which denotes the attribution of an arbitrary s-mer of weight w to be correct or not, given no context. A low T_C indicates that a lower threshold to s-mer correctness, resulting in more erroneous s-mers being interpreted as correct. Inversely, if T_C is set too high, not enough s-mers are interpreted as correct, and correction is minimal. Therefore, T_C is set to a significant value greater than where the number of average correct s-mers equals the number of average incorrect s-mers. This ensures that the T_C value is not too low to pick up too many ambiguous s-mers, and not too high as to not pick up any s-mers at all.

Using some datasets M1, M2, and M5 (Figure 4.1), we can illustrate selection of T_C by computing the Poisson($\lambda_C,\ T_C$) and Poisson($\lambda_E,\ T_C$) on multiple T_C (3.2). We can see for dataset M1, that the point of intersection of when Poisson($\lambda_C,\ T_C$) exceeds Poisson($\lambda_E,\ T_C$) is at $T_C = 18$. This means that at 18 observations of an arbitrary s-mer, we can reason it to be more likely to be fully correct as compared to being incorrect at one base. QUESS will select the T_C value that is 2 greater than this, being at 20 for dataset M1, resulting in a Poisson($\lambda_C,\ T_C$) value that is about 1000x greater than Poisson($\lambda_E,\ T_C$), illustrating that at 20 observations of an arbitrary k-mer in the dataset, it is about 1000 more likely to be fully correct than incorrect at exactly one base.

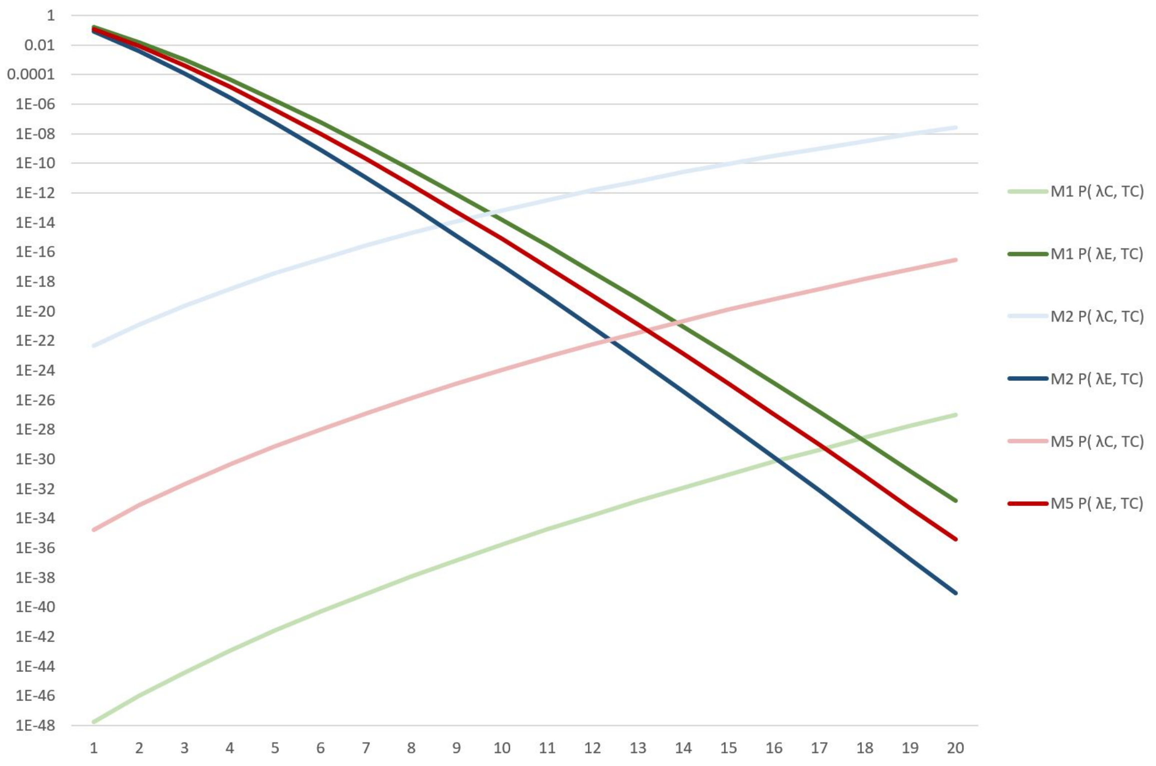


Figure 3.2: **$\mathbf{T_C}$ Selection**. Each dataset has two lines mapping the Poisson distribution of λ_C and λ_E where the lighter shade is Poisson($\lambda_C,\ T_C$), and the darker shade is Poisson($\lambda_E,\ T_C$). The point of intersection is when it is more likely that a s-mer is likely to be correct compared to correct except for one base.

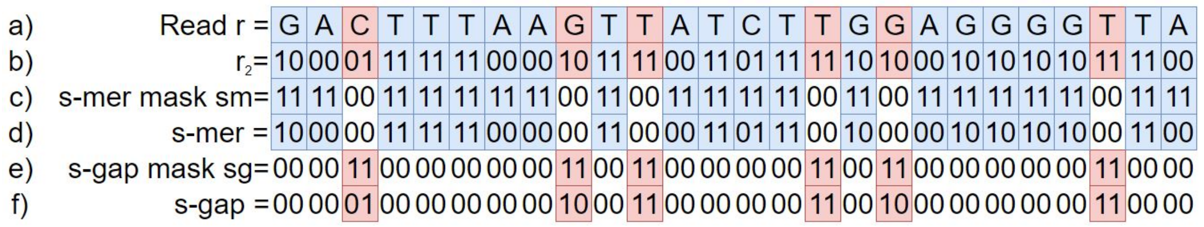
Algorithm 2 computeTC(L)

```
input: $L$                                              ▷ $L$ – genome size
output: $T_C$                                           ▷ $T_C$ – threshold for correctness
let $r$ denote a read
let $|r|$ denote average read length
let $w$ denote s-mer of weight w
let $N$ represent number of reads from input dataset
let $\epsilon$ represent error
$\lambda_C = (|r| - w)\frac{N}{L}(1-\epsilon)^w$.
$\lambda_E = (|r| - w)\frac{N}{L}(1-\epsilon)^{w-1}\epsilon\frac{1}{3}$
while $Poisson(\lambda_C, T_C) < Poisson(\lambda_E, T_C)$ do      ▷ $4 \leq T_C \leq 254$
    $T_C + +$
$T_C$+=2;
```

In where T_C is more related to the expected s-mer frequency, T_E is the threshold that is related to the expected s-gap frequency. For each s-mer found within the dataset, there will be a number of variant s-gaps, and if the count of multiple s-gaps exceed T_E, then it might dictate that the s-mer is inherently ambiguous to begin with. A low T_E dictates that the s-mer will be very trusted, as it is more easily classified as ambiguous, and a T_E of 1 forces every s-mer to be classified as ambiguous given any variant s-gap at all. Inversely, a high T_E dictates that the s-mer will be less trusted, due to an acceptance of variant s-gaps. QUESS computes T_E as a function of T_C to allow for conservative correction in the first iterations, by computing T_E as $\frac{1}{4}$ of T_C with the first four spaced seeds, and $\frac{1}{2}$ of T_C with the last four spaced seeds, with a floor of $T_E = 2$, to not force all s-mers to be classified as ambiguous.

3.5 Counting s-mers and s-gaps

QUESS refers to a s-mer as the collective 1-bits of a specific spaced seed, and a s-gap as the collective 0-bits corresponding to the gaps in the seed (Figure 3.3). It can further be related that these s-mers will be the anchor points in where QUESS accumulates variant s-gaps counts to ascertain the most frequent s-gap for each corresponding s-mer. Recall that base pairs can be denoted by the set { A,G,C,T }, which can be represented by 2 bits per base pair (Figure 3.3). With QUESS, we denote A as 00, T as 11, C as 01, and G as 10, where the reverse complement of A is T and C is G. Recall in biology, these are the complementing base pairs that bind to each other (A binds to T, G binds to C), which allows for the inference of the opposite read. We only characterize a s-mer and its reversely complementing pair as the same, as they are redundant and represent the same s-mer.



a)	Read r =	G	A	C	T	T	T	A	A	G	T	T	A	T	C	T	T	G	G	A	G	G	G	G	T	T	A
b)	r_2=	10	00	01	11	11	11	00	00	10	11	11	00	11	01	11	11	10	10	00	10	10	10	10	11	11	00
c)	s-mer mask sm=	11	11	00	11	11	11	11	11	00	11	00	11	11	11	11	00	11	00	11	11	11	11	11	00	11	11
d)	s-mer =	10	00	00	11	11	11	00	00	00	11	00	00	11	01	11	00	10	00	00	10	10	10	10	00	11	00
e)	s-gap mask sg=	00	00	11	00	00	00	00	00	11	00	11	00	00	00	00	11	00	11	00	00	00	00	00	11	00	00
f)	s-gap =	00	00	01	00	00	00	00	00	10	00	11	00	00	00	00	11	00	10	00	00	00	00	00	11	00	00

Figure 3.3: **S-mer Example.** a) The input read r of size 26, also known as a 26-mer. b) The binary encoding of r, r_2, note that A=00, T=11, C=01, G=10. c) The s-mer mask. d) The s-mer given r_2 and s-mer mask (highlighted in blue). e) The s-gap mask (inverse of s-mer mask) f) The s-gap given r_2 and the s-gap mask (highlighted in red).

In the next step after parameter computation, insertSMers() fetches the reads r from the filtered input dataset R_T in parallel and encodes each r into their 2-bit encoding r_2. QUESS subsequently utilizes a window along r_2 along with two masks: the s-mer mask and the mask of its reverse complement. This works due to DNA's reverse complementary binding nature, in where it binds inversely in the opposite direction, and we can therefore not ignore the reverse complement sequence. QUESS will use the bit-wise function AND to extract the s-mer and the corresponding reverse complement. The smaller of the two resulting 64-bit integers is added to the hash table using a hash function dependent on its 64-bit integer representation, or increment an existing entry.

Algorithm 3 insertSMers($S_I,\ R_T,\ H$)

```
input: $S_I$, $R_T$, $H$        ▷ $S_I$ – spaced seed, $R_T$ – input dataset, $H$ –hash table
output: $H$                     ▷ $H$ – updated hash table
let $r$ represent a read, $RC$ stand for reverse complement
while ( !EOF( $R_T$ ) & !full ( $H$ ) ) do
    for $t$ threads in parallel do      ▷ Each $t$ works on $|R_T|/t$ reads of $R_T$
        $r_2 = r$                        ▷ $r_2$ – binary coding of $r$
        $s - mer = r_2\ \&\ mask_{s-mer}$
        $s - mer_{RC} = r_2\ \&\ mask_{RC(s-mer)}$
        $s - mer$ inserted into $H$ using double hashing
        min( $s - mer$ , $s - mer_{RC}$ ) inserted into $H$ using double hashing
if full ( $H$ ) then
    increase hash table size and restart from beginning of file
```

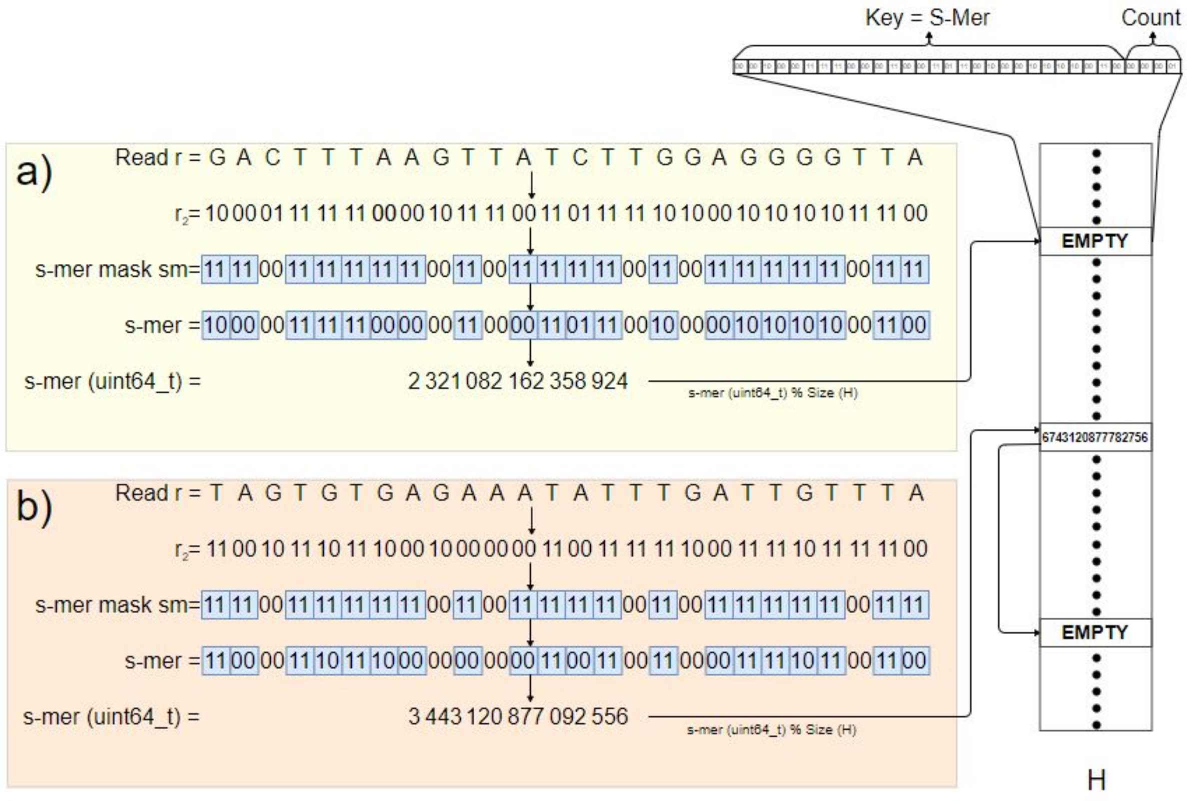

Key = S-Mer
Count
a)
Read r = G A C T T T A A G T T A T C T T G G A G G G G T T A
r₂= 10 00 01 11 11 11 00 00 10 11 11 00 11 01 11 11 10 10 00 10 10 10 10 11 11 00
s-mer mask sm= 11 11 00 11 11 11 11 11 00 11 00 11 11 11 11 00 11 00 11 11 11 11 11 00 11 11
s-mer = 10 00 00 11 11 11 00 00 00 11 00 00 11 01 11 00 10 00 00 10 10 10 10 00 11 00
s-mer (uint64_t) =
2 321 082 162 358 924
s-mer (uint64_t) % Size (H)
EMPTY
6743120877782756
b)
Read r = T A G T G T G A G A A A T A T T T G A T T G T T T A
r₂= 11 00 10 11 10 11 10 00 10 00 00 00 11 00 11 11 11 10 00 11 11 10 11 11 11 00
s-mer mask sm= 11 11 00 11 11 11 11 11 00 11 00 11 11 11 11 00 11 00 11 11 11 11 11 00 11 11
s-mer = 11 00 00 11 10 11 10 00 00 00 00 00 11 00 11 00 11 00 00 11 11 10 11 00 11 00
s-mer (uint64_t) =
3 443 120 877 092 556
s-mer (uint64_t) % Size (H)
EMPTY
H

Figure 3.4: **Data Structure 1: S-Mer Insertion.** An overview of how s-mers are inserted. **a)** A s-mer is inserted with a key 2 321 082 162 356 924 finding an empty spot in H. It is inserted, where the first 28 bits contain its s-mer, and the last 8 bits contain its count. **b)** A s-mer is inserted with a key 3 443 120 877 092 556 and encounters a collision. It will then use double hashing to find another empty spot in H.

After initial insertion of s-mers, rehashFrequentSMers() utilizes T_C to disambiguate and reduce the size of the hash table. A correct s-mer is expect to appear at least T_C times throughout the dataset, and QUESS uses these s-mers to form a new smaller hash table that contains only unambiguous s-mers, along with their count. These s-mers are considered strong s-mers.

Algorithm 4 rehashFrequentSMers(H, T_C)

```
input: $H$, $T_C$                         ▷ $H$ –hash table, $T_C$ – threshold for correctness
output: $H$                                            ▷ $H$ – updated hash table
initialize $Freq$              ▷ $Freq$ – counter for number of frequent S-Mers
for $i = H$.size do
    if $H[i]$.count  $T_C$ then
        $Freq$ ++
initialize $H'$ of approximately $Freq$
for $i = H$.size do
    if $H[i]$.count  $T_C$ then
        $H'$.insert( $H[i]$ )
replace $H$ with $H'$
```

QUESS then utilizes insertSGaps() to add the s-gaps in the same way as s-mer insertion. However, s-gaps are directly related to their corresponding s-mer, due to their interlinking nature. This means that they are not added to the hash table as a function of their own hash function, but as an additional element, linked to the original s-mer hash key. Variant s-gaps can be added to the same s-mer, so long as no s-mer has two s-gaps that exceed T_C or if multiple s-gaps exceed T_E – a measure based on T_C that denotes the threshold of erroneous s-gaps allowed to exist – since this would denote an ambiguity of either the s-mer or s-gap. Similar to the way s-mers were inserted, insertion of new s-gaps inserts an element to linked to the s-mer, and insertion of existing s-gaps increments the count of pre-existing s-gaps.

Algorithm 5 insertSGaps(S_I, R_T, H)

```
input: $S_I$, $R_T$, $H$              ▷ $S_I$ – spaced seed, $R_T$ – input dataset, $H$ –hash table
output: $H$                                                    ▷ $H$ – updated hash table
let $r$ represent a read, $RC$ stand for reverse complement
while ( !EOF( $R_T$ ) ) do
    for $t$ threads in parallel do                 ▷ Each $t$ works on $|R_T|/t$ reads of $R_T$
        $r_2 = r$                                              ▷ $r_2$ – binary coding of $r$
        $s - mer = r_2$ & $mask_{s-mer}$
        $s - mer_{RC} = r_2$ & $mask_{RC(s-mer)}$
        $s - gap = r_2$ & $mask_{s-gap}$
        $s - gap_{RC} = r_2$ & $mask_{RC(s-gap)}$
        $s - mer$ inserted into $H$ using double hashing
        if $H(\min(s - mer, s - mer_{RC}))$ exists and is not marked ambiguous then
            corresponding $s - gap$ inserted into $H$ under key = $s - mer$
            if more than 1 s-gap variant $T_C$ for $H(s - mer)$ then
                mark $H(s - mer)$ as ambiguous
```

3.6 Error correction

Before correction can occur, the s-gaps present in H must be disambiguated. With removeAmbiguousSMers(), QUESS iterates through the hash table and assigns the s-gap with the largest count as the canonical correct s-gap for its associated s-mer. The strength of this s-gap is denoted as a function of its count versus the count of the next strongest variant.

Algorithm 6 removeAmbiguousSMers(T_C, T_E, H)

```
input: $T_C$, $T_E$, $H$   ▷ $T_C$ – threshold for correctness, $T_E$ – threshold for extra s-gaps,
$H$ –hash table
output: $H$                                                    ▷ $H$ – updated hash table
for $i = H$.size do
    if $H[i].s - mer$ exists and not ambiguous then
        check count of all variant S-Gaps
        if 1 variant count $= T_E$ or no variant count $T_C$ then
            Mark $H[i].s - mer$ as ambiguous
        if $H[i].s - mer$ is not ambiguous then
            Take highest count $s - gap$ to be most correct $s - gap$ for that $s - mer$
```

After ambiguous s-mers and s-gaps have been removed from the hash table, correct() then corrects the reads by iterating through the input dataset in parallel. If it encounters an s-mer that matches an s-mer or its reverse complement in H, it will compare the s-gap from the input dataset with the canonically correct s-gap present in H. These s-gaps may have different bits, and only if the input s-gap is different to the correct s-gap by less than T_{DIFF} – defined as the allowance of difference of bits between the s-gaps of the input dataset and computed one – will the program correct it.

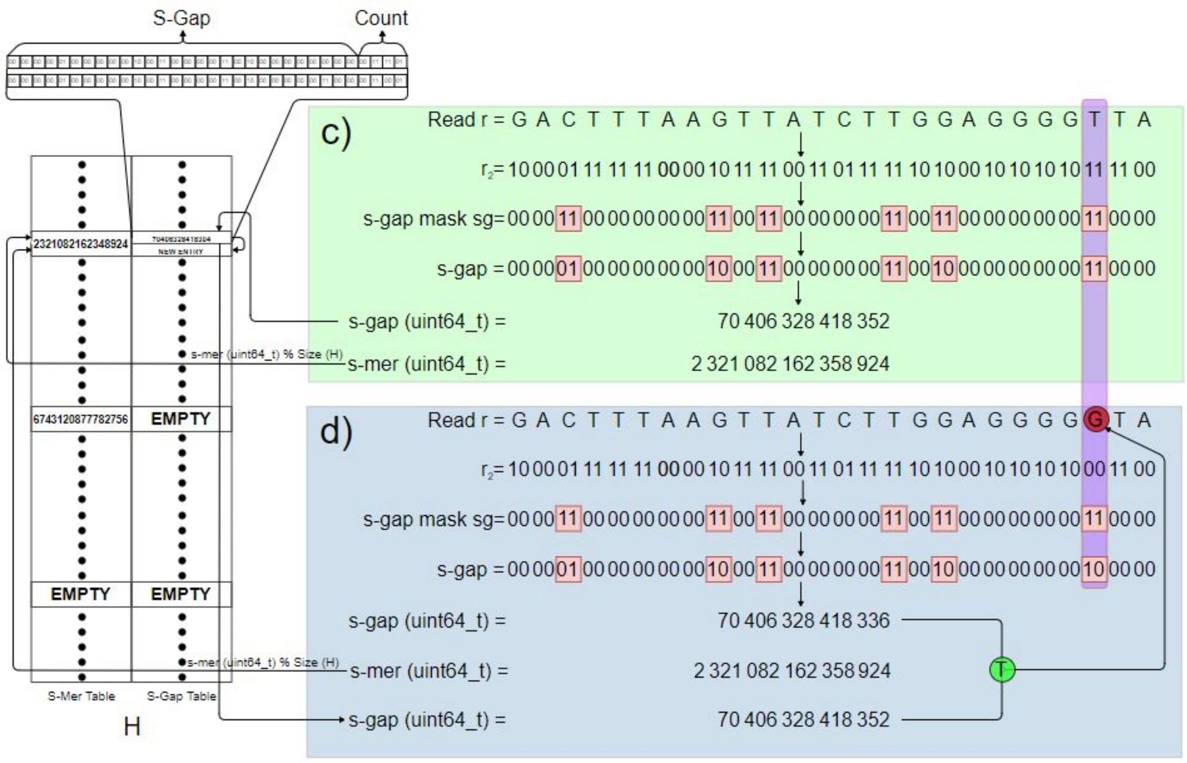

S-Gap
Count
c)
Read r = G A C T T T A A G T T A T C T T G G A G G G G T T A
r2= 10 00 01 11 11 11 00 00 10 11 11 00 11 01 11 11 10 10 00 10 10 10 10 11 11 00
s-gap mask sg= 00 00 11 00 00 00 00 00 11 00 11 00 00 00 00 11 00 11 00 00 00 00 00 11 00 00
s-gap = 00 00 01 00 00 00 00 00 10 00 11 00 00 00 00 11 00 10 00 00 00 00 00 11 00 00
s-gap (uint64_t) = 70 406 328 418 352
s-mer (uint64_t) = 2 321 082 162 358 924
s-mer (uint64_t) % Size (H)
2321082162348924
70406328418304
NEW ENTRY
6743120877782756
EMPTY
EMPTY
EMPTY
d)
Read r = G A C T T T A A G T T A T C T T G G A G G G G G T A
r2= 10 00 01 11 11 11 00 00 10 11 11 00 11 01 11 11 10 10 00 10 10 10 10 00 11 00
s-gap mask sg= 00 00 11 00 00 00 00 00 11 00 11 00 00 00 00 11 00 11 00 00 00 00 00 11 00 00
s-gap = 00 00 01 00 00 00 00 00 10 00 11 00 00 00 00 11 00 10 00 00 00 00 00 10 00 00
s-gap (uint64_t) = 70 406 328 418 336
s-mer (uint64_t) = 2 321 082 162 358 924
s-mer (uint64_t) % Size (H)
s-gap (uint64_t) = 70 406 328 418 352
T
S-Mer Table
S-Gap Table
H

Figure 3.5: **Data Structure 2: S-Gap Insertion and Correction.** An overview of s-gap insertion and correction. **c)** A s-gap is inserted with the s-mer as its key, 2 321 082 162 358 924 and finds that its s-gap, 70 406 328 418 352, is different from the s-gap associated with the s-mer, 70 406 328 418 304. It adds another entry under the same s-mer. **d)** A read probes the dataset with its s-mer, 2 321 082 162 358 924, and finds that the correct s-gap associated with it is 70 406 328 418 352. This is different (highlighted in red) from its own s-gap, 70 406 328 418 336, and we proceed to change the dataset s-gap to the correct s-gap.

Algorithm 7 correct(S_I, T_C, T_E, T_{DIFF}, R_T, R_{TC}, H)

```
input: $S_I$, $T_C$, $T_E$, $T_{DIFF}$, $R_T$, $R_{TC}$, $H$          ▷ $T_C$ – threshold for correctness
                              ▷ $T_E$ – threshold for extra s-gaps, $H$ –hash table
                                  ▷ $T_{DIFF}$ – difference threshold for s-gaps
output: $R_{TC}$                     ▷ $R_T$ – input dataset, $R_{TC}$ – output dataset
let $r$ represent a read, $RC$ stand for reverse complement
while ( !EOF( $R_T$ ) ) do
    for $t$ threads in parallel do            ▷ Each $t$ works on $|R_T|/t$ reads of $R_T$
        $r_2 = r$                                       ▷ $r_2$ – binary coding of $r$
        $s - mer = r_2$ & $mask_{s-mer}$
        $s - mer_{RC} = r_2$ & $mask_{RC(s-mer)}$
        $s - gap = r_2$ & $mask_{s-gap}$
        $s - gap_{RC} = r_2$ & $mask_{RC(s-gap)}$
        if $H(\min(s - mer, s - mer_{RC}))$ exists and is not marked ambiguous and (difference of $s - hap_{RT}$ and $s - hap_H) < T_{DIFF}$ then
            change $s - gap_{RT}$ to $s - gap_H$
        write $r$ into $R_{TC}$
```

QUESS will subsequently reiterate through using the newly corrected dataset with more variant seeds of the same weight and length. Note that as QUESS runs on multiple seeds, T_E and T_{DIFF} are different between iterations. In earlier iterations, T_E will have smaller values, which will correspond to more s-mers being marked ambiguous, leading to only having strong s-mers in the pool by the time insertSGaps() is invoked. In addition, T_{DIFF} will be larger, which allows for a greater number of input dataset reads to be corrected by a stronger s-mer. This allows for later iterations to be more aggressive in their correction, as their initial input reads will have been conservatively corrected.

3.7 Data structures and Parallelization

QUESS uses a hash table to store and correct k-mers from an input dataset. The input 28-mers, originally represented by a string of char is reinterpreted as an encoding of 2 bits, which is then masked to a resulting s-mer. Initially, each entry of the hash table is double hashed in as an 64 bit integer entry consisting of an s-mer (2 bit $* 28 = 56$ bits), and a 8 bit count (Figure 3.4). Additionally, an associated s-gap table is created, where the parent s-mer identity is used as a key, and the value consists of an array of 64 bit integers (2 bit $* 28 = 56$ bits s-gap and 8 bit count) (Figure 3.5).

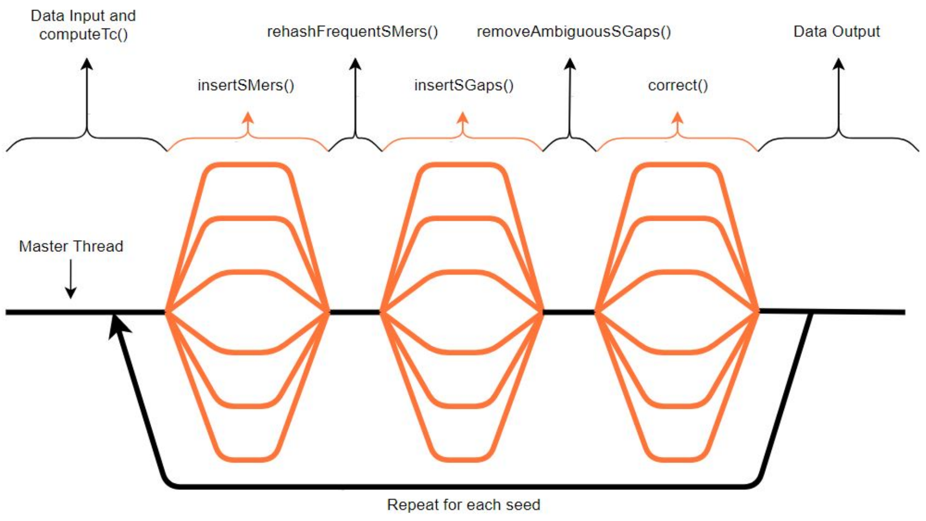


Figure 3.6: **QUESS Parallelization Model.** Fork-join model depicting work done by QUESS on various stages of the program. The orange lines depicts threads, in where the test cases will have 12 threads.

OpenMP was the choice of parallelization going forward due to the flexibility of evaluating serial versus parallel code during testing. While work stealing protocol is not explicit in documentation, the asymmetric nature of the input reads may cause unintended bottlenecks during parallelization. OpenMP support of dynamic loop scheduling and simple use of directives made it the first choice in implementation, although other parallelization concurrency platforms may be considered in future studies. Speedup is measured and is sufficient given 12 cores, as given in Table 3.2.

Speedup (Ts/Tp)					Efficiency (Speedup/P)				
Miseq	Cores				Miseq	Cores			
Dataset	2	4	8	12	Dataset	2	4	8	12
SRR5230111	1.896764	3.381868	7.19883	9.848	SRR5230111	0.948382	0.845467	0.899854	0.820667
SRR5075712	1.836268	3.442244	6.379205	9.438914	SRR5075712	0.918134	0.860561	0.797401	0.786576
SRR4417428	1.597847	3.052336	5.853047	7.965854	SRR4417428	0.798924	0.763084	0.731631	0.663821
SRR5181392	1.650585	3.216524	6.037433	8.09319	SRR5181392	0.825292	0.804131	0.754679	0.674432
SRR5164004	1.874001	3.380266	5.444643	7.817949	SRR5164004	0.937001	0.845067	0.68058	0.651496
SRR5110368	1.510607	2.613924	5.241117	7.145329	SRR5110368	0.755304	0.653481	0.65514	0.595444
SRR4423181	1.682958	3.185841	6.226415	9.875312	SRR4423181	0.841479	0.79646	0.778302	0.822943
ERR1485307	1.942128	3.364486	8.497854	8.497854	ERR1485307	0.971064	0.841121	1.062232	0.708155
SRR2317595	1.747573	2.948805	5.482234	7.309645	SRR2317595	0.873786	0.737201	0.685279	0.609137
SRR5146462	1.236264	2.312139	4.017857	8.275862	SRR5146462	0.618132	0.578035	0.502232	0.689655
Average	1.6975	3.09	6.04	8.43	Average	0.84875	0.77	0.75	0.70

Table 3.2: **QUESS Speedup.** Comparison on program performance on use of different number of cores on MiSeq data. Darker colors indicate better results.

3.8 My Contributions

My contributions towards implementation of QUESS can be categorized into a three main categories: design and implementation of algorithm; optimization of run-time parameters; and dataset retrieval and analysis.

I implemented the algorithm using C++ and OpenMP with a design based on quality, speed and memory. With regards to quality, there was a lot of deliberation on initial test parameters such as seed weight, number of seeds, and the computed parameters T_C, T_E and T_{DIFF}. Seed weight and number of seeds were experimentally derived as shown in Figure 3.1. T_C was derived through calculation of optimal breakpoints as denoted by Figure 3.2, in where the point of intersection was initialized as T_C.

Subsequent optimization demonstrated that an additional 2 units allowed for significant increase in performance due to the additional representation of correct s-mers over incorrect s-mers at this level of T_C. In addition, educated choices were made with regards to T_E and T_{DIFF} that correlated them to fit the overall heuristic of the program, in where iterations of initial seeds allows for more aggressive correction of the DNA sequencing data on the later iterations of the program.

Some optimizations were experimented but did not enhance performance and did not make it into the thesis. In particular, I tried to incorporate the quality scores of FastQ datasets into the correction heuristic in a way akin to how BLESS 2 deals with quality, but the results were negligible with an increase time and memory. In addition, I tried to test acceptance of correctness of 2-to-many variant s-gaps given sufficient support above T_C, as well as trying to reduce the amount of ambiguous s-mers by setting higher thresholds for T_E; but these changes tended to decrease performance and were subsequently discontinued.

With regards to speed, I implemented the algorithm with use of OpenMP due to its ease of implementation, and it was shown to have good speed up on 12 cores (Table 3.2). Initially, QUESS was utilizing about 2-3 times more memory than the program at the time of thesis submission. To mitigate this problem, I minimized the threshold in which the s-gap hash table could accept new s-gaps depending on the seed number; when the program would run unchanged, there was tens to hundreds of erroneous s-gaps that were added to the s-gap hash table, which ended up increasing hash table size by about 2 to 3 fold. QUESS allowed for greater variation of variant s-gaps for the initial seeds – due to having a larger pool of variant s-gaps due to the relative incorrectness of the dataset – while the later seeds had less leniency because the dataset should be mostly correct at that point. This allowed for reduction of hash table size – which was a bottleneck in terms of memory usage of QUESS – while not compromising quality. This shifted the burden of memory usage to a mixture between the hash table sizing and the parallel functions acting upon the input reads.

To test our results, it was often not useful to rely on other documented sources, as they were outdated and/or simulated. Because of this, with aid from my background in biology, I compiled a list of a large variety of organisms spanning varying genome sizes, with datasets having various coverage levels, read lengths, and error levels. This was not a simple task, as datasets often did not correspond well with the available reference genomes; I compiled datasets that were of high quality and were good candidates for downstream applications (as most datasets were relevant only for a specific application).

I installed competing programs onto SHARCNet and created scripts to automatically run everything, collect data, and organize into a cohesive format to transfer onto a spreadsheet for interpretation. Running programs took a significant time, as some datasets were exceedingly large (>200GB), and some competing programs took either too much time (>14 days) to execute or too much memory (>1TB) to execute. I have also compiled multi-paged spreadsheets summarizing the results of all aforementioned optimizations and experiments.

Chapter 4

Results

4.1 Datasets

We have performed testing on a variety of datasets spanning different models of machines, different read lengths, genome sizes and coverage levels. Unaltered real datasets are used in favour of artificial datasets since errors from artificial datasets can be localized. With the case of real datasets, there are often reads that very difficult to correct due to their inherent ambiguity, and a reference genome may not always be reliable or readily available. In order to improve correction numbers, discarding of these difficult reads that do not align to a reference genome allow for programs to inflate their performance. However, if there is such a scenario in where the reference genome is unavailable (such as in *de novo* assembly), reliance on these shortcuts do not aid in performance, and in these cases, it can be seen that evaluation of real datasets is the only method that allows for unbiased results. The accession numbers and details for each of the 10 MiSeq datasets, 10 HiSeq datasets, and 7 NextSeq datasets are given in Table 4.1.

Dataset	Organism	Accession Number	Read Length	Number of Reads	Total bp	Coverage	Reference Genome	Genome Length
M1	*Campylobacter jejuni*	SRR5230111	251	949,460	219,003,621	133.94	NC_017279.1	1,635,045
M2	*Escherichia coli*	SRR5075712	251	1,239,176	299,280,330	64.48	NC_000913.3	4,641,652
M3	*Streptococcus pneumoniae*	SRR4417428	301	1,037,914	311,910,269	153.00	NC_003098.1	2,038,615
M4	*Listeria monocytogenes*	SRR5181392	251	1,589,398	387,240,543	131.51	NC_003210.1	2,944,528
M5	*Salmonella enterica*	SRR5164004	251	2,059,666	488,021,390	98.56	AE006468.2	4,951,383
M6	*Staphylococcus aureus*	SRR5110368	251	2,259,206	567,060,706	200.99	NC_007795.1	2,821,361
M7	*Mycobacterium tuberculosis*	SRR4423181	300	3,033,410	626,299,841	141.97	NC_000962.3	4,411,532
M8	*Vibrio cholerae*	ERR1485307	301	3,341,186	646,401,481	160.26	NC_002505.1	4,033,464
M9	*Saccharomyces cerevisiae*	SRR2317595	250	3,386,250	678,196,775	55.79	NC_001133.9	12,157,105
M10	*Klebsiella pneumoniae*	SRR5146462	300	2,578,642	773,592,600	136.14	NC_016845.1	5,682,322

Dataset	Organism	Accession Number	Read Length	Number of Reads	Total bp	Coverage	Reference Genome	Genome Length
H1	*Staphylococcus aureus*	SRR5226582	125	899,444	112,430,500	39.85	NC_007795.1	2,821,361
H2	*Shigella flexneri*	SRR5029830	100	3,399,844	339,984,400	70.41	NC_004337.2	4,828,820
H3	*Streptococcus pneumoniae*	ERR1764059	125	2,964,194	370,524,250	181.75	NC_003098.1	2,038,615
H4	*Escherichia coli*	SRR5177719	100	5,520,000	552,000,000	118.92	NC_000913.3	4,641,652
H5	*Campylobacter jejuni*	SRR4451711	100	5,520,588	552,058,800	337.64	NC_017279.1	1,635,045
H6	*Arabidopsis thaliana*	ERR1904924	101	129,341,488	13,063,490,288	109.17	NC_003070.9	119,667,750
H7	*Caenorhabditis elegans*	ERR294487	100	130,706,970	13,070,697,000	130.33	NC_003279.8	100,286,401
H8	*Drosophila melanogaster*	SRR3939099	151	337,747,590	50,999,886,090	354.84	NC_004354.4	143,726,002
H9	*Homo sapiens*	SRR5408851	150	952,436,102	143,817,851,402	44.41	NC_000001.1	3,238,442,024
H10	*Mus musculus*	DRR082909	151	952,649,182	143,850,026,482	51.23	NC_000067.6	2,807,715,301

Dataset	Organism	Accession Number	Read Length	Number of Reads	Total bp	Coverage	Reference Genome	Genome Length
N1	*Staphylococcus aureus*	ERR1845153	151	1,675,018	240,239,363	85.15	NC_007795.1	2,821,361
N2	*Escherichia coli*	SRR5336197	150	2,994,552	393,966,302	84.88	NC_000913.3	4,641,652
N3	*Listeria monocytogenes*	SRR5088142	151	2,751,302	398,032,839	135.18	NC_003210.1	2,944,528
N4	*Saccharomyces cerevisiae*	SRR5251561	75	13,381,854	1,003,639,050	82.56	NC_001133.9	12,157,105
N5	*Salmonella enterica*	SRR5381280	151	7,745,830	1,110,882,098	224.36	AE006468.2	4,951,383
N6	*Klebsiella pneumoniae*	SRR5093762	151	10,240,128	1,537,618,483	270.60	NC_016845.1	5,682,322
N7	*Drosophila melanogaster*	SRR3285619	151	62,547,046	9,444,603,946	65.71	NC_004354.4	143,726,002

Table 4.1: **Datasets.** Summary of all datasets. **(Top)** MiSeq (M1 – M10). **(Middle)** HiSeq (H1 – H10). **(Bottom)** NextSeq (N1 – N7). Each dataset includes the organism; the accession number to access the dataset from NCBI; the number of reads within the input read bank; the number of base pairs within the input read bank; the coverage level (total base pairs / genome length); the reference genome accession number; and the genome length.

4.2 Correction Evaluation

There are two branches of thought that have emerged behind evaluation of program performance: one method considers individual base correction [23], whereas the other considers evaluation of reads and k-mers [15]. However, one must consider not only individual base pair corrections in the context of *de novo* genome assembly and general correction, as correction of an arbitrary base pair cannot be equated to the correction of a base pair that allows for a whole read to be interpreted as correct. As such, we will consider the latter evaluation, and the relative performance of QUESS and all tested programs will be evaluated as described in [15]. For completeness, we describe briefly the evaluation method described in [15].

Consider an organism, its reference genome G of length L nucleotide base pairs, and an uncorrected dataset D of N number of reads, r: $D = \{r_i | 1 < i < N\}$, where the length in base pairs of read r can be denoted as $|r|$ or ℓ. We can evaluate the resulting file in comparison to the initial uncorrected file in terms of gains in depth and breadth of coverage (Figure 1.7), where depth of coverage is the average number of times each base position of G is sequenced: $\frac{1}{L}\sum_{n=1}^{N}|r_i|$. The breadth of coverage is the percentage of the genome covered covered by the union of the reads of D. It can be noted that there must be a sufficient overlap of ℓ bases of G to facilitate useful sequences. For reads of size ℓ, we can define the breadth of coverage as a ratio of ℓ–mers in G that appear in D: $\frac{|\ell{-}mer(G) \cap \ell{-}mer(D)|}{\ell{-}mer(D)}$.

4.2.1 Depth of Coverage Gain

A read r is considered correct if it is found in the reference genome, $r = G[i...i + |r| - 1]$ for some arbitrary $1 \leq i \leq L - |r| - 1$, and erroneous otherwise. We can categorize program output in one of four ways; TP = number of reads that were erroneous before and correct after program execution; FP = number of reads that were correct before and erroneous after; TN = number of reads that were correct both before and after; and FN = number of reads that were erroneous before and after. We can then define $\textsc{ReadDepthGain} = \frac{TP-FP}{TP+FN}$, where TP-FP represents the net gain of correct reads after program execution, and TP+FN representing the total possible number of erroneous reads prior to program execution. We can also express the quality of the original dataset as a function of $\textsc{OrigReadDepth} = \frac{N}{N+P} = \frac{TN+FN}{TN+FN+TP+FP}$, and the quality of the corrected dataset as $\textsc{CorrReadDepth} = \frac{TP+TN}{N+P} = \frac{TP+TN}{TN+FN+TP+FP}$.

However, assessing the ratio of corrected versus uncorrected reads is not sufficient as a means to represent a program's overall effectiveness. There is a lot of significance that comes from correctness of consecutive bases; in the natural world, enzymes (which are the natural machines of the natural world) bind to specific sequences on the genomic and proteomic level. Errors within the sequence cause changes in levels of enzymatic binding, and can cause the mechanism to not interact at all. It is therefore imperative that sequences of consecutive bases are judged as a parameter for program evaluation.

A positional k-mer ($k,\ i,\ j$) – a k-mer starting at position j of read r_i – is considered correct if it is found within the reference genome, $r_i[j...j+k-1] = G[\ell...i+k-1]$ for some $1 \leq \ell \leq L-k+1$. In this case, we can define program output similarly as READDEPTHGAIN with TP, TN, FP and FN. We can define KMERDEPTHGAIN= $\frac{TP-FP}{TP+FN}$, where TP-FP represents the net gain of correct positional k-mers after program execution, and TP+FN representing the total possible number of erroneous positional k-mers prior to program execution.

4.2.2 Breadth of Coverage Gain

While evaluation of datasets using the depth of coverage can be helpful in evaluating a program's effectiveness in correcting certain errors, it may represent a programs dependence on correct only certain regions of the genome, rather than the entirety of the genome. Therefore, we can denote KMERBREADTHGAIN as the gain in coverage as a result of correct positional k-mers, where TP, TN, FP and FN are defined by correctness before and after the program execution. Similarly, READBREADTHGAIN is proportional to the number of elements covered after program correction versus before. It can be noted that programs with higher READBREADTHGAIN denote a gain in overall coverage of the reference genome, and programs with higher KMERBREADTHGAIN denote a greater preservation of unique k-mers. We refer the reader for more details to [15].

4.3 Comparison of Programs

All experiments were conducted on the goblin cluster under SHARCNet with an Intel® Xeon®, utilizing 32 cores running at 2.2 GHz and 1 TB RAM, and given a maximum run time of 14 days. The command-line options used for each program – ACE, BFC, BLESS 2, Blue, Karect, Lighter, Musket, RACER, SGA, and QUESS – are determined to be the example options that were given alongside supplementary materials, or default values if not given.

Our results are presented in Tables 4.2 – 4.4. The quality of the original and optimally corrected datasets is given for MiSeq (Table 4.5), Hiseq (Table 4.6) and NextSeq (Table 4.7), where each dataset is evaluated prior to correction, and the programs yielding the highest values for each measure is reported with their values after. The aforementioned evaluation parameters are noted in detail for each program tested in Table 4.2 for MiSeq datasets, Table 4.3 for HiSeq datasets, and Tables 4.4 for NextSeq datasets. Note that some datasets could not be run for specific programs, and the result is annotated at each relevant table.

4.3.1 Comparison of MiSeq datasets

ReadDepthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	RACER	SGA	QUESS
M1	30.37	30.66	29.37	32.11	31.54	29.76	30.60	27.93	31.79
M2	56.92	54.01	45.33	58.95	58.41	47.40	48.19	42.10	59.92
M3	19.97	19.57	17.60	20.26	20.20	17.98	18.46	16.77	20.33
M4	25.88	25.37	22.38	27.08	26.67	22.45	23.23	20.66	27.00
M5	31.87	30.36	26.23	33.14	32.29	27.10	27.63	24.78	32.76
M6	45.98	41.95	42.55	47.58	53.52	40.38	44.71	39.68	64.47
M7	78.40	74.82	78.67	80.45	83.03	74.91	85.32	75.74	84.46
M8	61.70	38.02	36.85	72.03	79.88	48.39	38.89	35.34	87.24
M9	62.37	75.25	56.22	64.61	69.08	61.11	64.35	56.24	63.94
M10	59.90	54.08	44.43	62.43	65.51	45.72	48.31	43.36	73.74
Average	47.33	44.41	39.96	49.86	52.01	41.52	42.97	38.26	54.56

ReadBreadthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	RACER	SGA	QUESS
M1	2.55	2.60	2.71	2.73	2.69	2.51	2.62	2.40	2.71
M2	5.29	5.02	4.79	5.46	5.41	4.42	4.51	3.94	5.56
M3	6.51	6.41	6.19	6.59	6.58	5.94	6.08	5.60	6.61
M4	4.14	4.05	3.95	4.30	4.24	3.62	3.75	3.37	4.32
M5	4.91	4.68	4.56	5.09	4.96	4.20	4.29	3.85	5.03
M6	13.11	12.06	16.49	13.48	15.06	11.65	12.81	11.48	17.76
M7	12.88	12.31	13.93	13.21	13.65	12.30	14.21	12.48	13.91
M8	8.70	5.14	10.01	10.06	11.00	6.72	5.43	4.79	12.07
M9	2.08	2.15	2.15	1.87	2.20	2.06	2.08	1.95	2.18
M10	12.94	11.91	12.90	13.33	13.87	10.41	10.94	10.05	15.18
Average	7.31	6.63	7.77	7.61	7.96	6.38	6.67	5.99	8.53

KmerDepthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	RACER	SGA	QUESS
M1	18.97	17.46	15.33	18.83	18.91	16.57	17.66	12.81	19.43
M2	51.74	40.89	34.75	47.15	48.99	35.83	41.04	19.58	51.42
M3	35.02	31.39	28.06	34.02	34.45	28.58	31.31	19.44	34.86
M4	34.35	27.83	24.39	31.60	32.74	23.99	27.61	14.51	34.10
M5	45.32	37.24	32.34	43.59	43.87	32.79	37.76	21.04	45.41
M6	33.16	23.29	20.03	26.00	36.19	21.74	26.29	16.15	45.55
M7	77.62	76.95	63.85	77.83	81.09	76.14	82.15	73.72	82.53
M8	67.82	18.77	27.81	53.17	67.13	39.23	47.88	14.36	73.32
M9	55.17	52.65	30.74	52.64	58.04	46.59	51.87	32.61	54.30
M10	49.87	37.03	30.09	44.24	49.51	31.32	39.19	17.61	59.59
Average	46.90	36.35	30.74	42.91	47.09	35.28	40.27	24.18	50.05

KmerBreadthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	RACER	SGA	QUESS
M1	-11.86	-3.24	-4.29	-3.45	-3.44	-3.25	-3.59	-2.79	-3.49
M2	-4.09	-1.97	-3.34	-2.25	-2.30	-2.07	-2.45	-1.40	-2.36
M3	-10.56	-9.30	-9.72	-10.05	-10.20	-9.35	-9.96	-6.35	-10.29
M4	-6.68	-5.29	-5.78	-5.99	-6.06	-5.49	-5.84	-3.40	-6.18
M5	-5.22	-4.36	-4.79	-4.71	-4.75	-4.43	-4.68	-3.06	-4.87
M6	-4.55	-2.75	-3.43	-3.08	-3.36	-2.89	-3.49	-2.32	-3.64
M7	-93.82	-7.76	-19.93	-7.20	-8.95	-9.23	-33.66	-5.60	-19.25
M8	-5.65	-2.65	-4.45	-4.55	-5.20	-4.35	-5.43	-1.64	-5.17
M9	-429.84	-2.88	-320.46	-18.72	-3.59	-9.74	-72.34	-0.97	-8.49
M10	-2.00	-1.58	-1.83	-1.74	-1.79	-1.57	-1.74	-0.87	-1.92
Average	-57.43	-4.18	-37.80	-6.17	-4.96	-5.24	-14.32	-2.84	-6.56

Table 4.2: **MiSeq Performance.** Comparison on program performance on MiSeq datasets (M1 – M10). **(Top to Bottom)** READDEPTHGAIN, READBREADTHGAIN, KMERDEPTHGAIN, KMERBREADTHGAIN. Darker colors indicate better results. Musket results are excluded due its very poor performance.

The error correction comparison on MiSeq datasets (M1 – M10) is presented in Table 4.2 for each of the four measures: ReadDepthGain, ReadBreadthGain, KmerDepthGain, and KmerBreadthGain. It should noted that Musket results are not compared as its performance is too subpar compared to the performance of other programs.

With regards to ReadDepthGain results, QUESS performs the best on average with the exception of M9, topping the most datasets, with Blue and Karect following behind, while SGA consistently performs the worst, followed by BLESS 2. This trend appears similarly again with ReadBreadthGain results, with QUESS performing the best, and Karect and BLESS behind it. QUESS is the best performer with the KmerDepthGain measure, followed by Karect and ACE. For KmerBreadthGain results, SGA outperforms all other programs, but it is suspected that the reason it does so well in this measure is due to its relative poor performance in every other measure. ACE is consistently the worst performing program with regards to KmerBreadthGain, and it can be inferred to be much more aggressive compared to the other programs. QUESS has the best average score amounts three of the four measures, except for KmerBreadthGain.

4.3.2 Comparison of HiSeq datasets

ReadDepthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
H1	53.91	25.17	30.83	46.95	53.25	28.77	27.82	40.86	39.33	83.08
H2	65.13	59.48	57.92	65.92	65.55	58.02	57.68	61.05	55.91	67.27
H3	10.55	10.43	10.39	10.63	10.57	10.29	10.27	10.51	10.14	10.64
H4	96.62	96.96	96.31	98.29	98.18	95.04	94.98	96.84	88.33	96.21
H5	44.95	42.62	40.18	42.58	44.86	42.53	38.94	41.25	31.57	45.78
H6	40.24	42.28	40.96	32.94	46.83	36.21	41.58	42.76	34.42	41.44
H7	73.06	74.11	[b]	72.99	76.27	69.16	55.70	74.15	71.18	71.29
H8	26.46	24.52	27.19	32.10	[c]	22.98	22.73	29.07	18.07	26.59
H9	[a]	35.53	30.77	[a]	[c]	27.91	25.09	32.44	34.09	32.85
H10	[a]	52.09	[b]	[a]	[c]	41.18	38.24	45.42	47.44	46.73
Average	51.36	46.32	41.82	50.30	56.50	43.21	41.31	47.43	43.05	52.19

ReadBreadthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
H1	3.26	1.55	4.29	2.74	3.18	1.77	1.71	2.55	2.42	4.84
H2	6.29	5.69	6.20	6.03	6.20	5.69	5.51	5.88	5.47	6.48
H3	3.83	3.80	3.85	3.82	3.83	3.75	3.74	3.82	3.69	3.85
H4	10.56	10.59	10.65	10.59	10.71	10.40	10.38	10.58	9.62	10.50
H5	13.51	13.49	12.83	13.33	13.91	13.21	12.56	12.87	11.30	13.96
H6	6.86	6.84	6.82	4.80	6.97	6.68	6.32	6.74	6.60	6.40
H7	22.66	22.79	[b]	22.53	23.26	21.59	17.35	23.03	22.35	21.78
H8	17.64	16.38	18.99	18.20	[c]	15.30	6.24	17.23	13.53	17.39
H9	[a]	2.77	3.00	[a]	[c]	2.18	1.97	2.52	2.71	2.60
H10	[a]	5.33	[b]	[a]	[c]	4.25	3.95	4.67	4.91	4.83
Average	10.58	8.92	8.33	10.25	9.72	8.48	6.97	8.99	8.26	9.26

ReadKmerGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
H1	50.64	13.80	-4.61	23.10	40.04	14.78	14.04	30.53	17.12	82.84
H2	48.71	43.59	39.72	45.91	48.30	42.44	41.21	44.73	35.23	49.69
H3	9.36	8.91	8.51	9.11	9.16	8.60	8.63	8.87	7.81	9.23
H4	94.34	94.36	92.56	95.80	95.44	92.23	92.45	94.33	84.41	93.73
H5	49.35	39.95	34.36	38.19	45.42	39.74	34.62	38.03	22.66	45.73
H6	42.54	36.62	29.77	32.78	43.85	33.43	37.82	40.32	28.30	40.76
H7	50.05	47.87	[b]	48.89	52.39	45.72	45.23	51.37	39.40	50.00
H8	41.53	30.78	26.51	39.38	[c]	29.45	33.79	39.16	19.32	38.44
H9	[a]	31.80	14.78	[a]	[c]	29.83	29.39	34.20	27.21	35.15
H10	[a]	45.00	[b]	[a]	[c]	41.46	42.76	46.41	34.98	47.19
Average	48.31	39.27	30.20	41.65	47.80	37.77	37.99	42.79	31.64	49.28

KmerBreadthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
H1	-3.46	-0.03	-4.02	-0.30	-0.01	-0.02	-0.10	-0.12	-0.01	-0.02
H2	-3.92	-3.03	-6.73	-4.67	-3.47	-2.75	-3.61	-3.70	-2.24	-3.16
H3	-3.58	-3.44	-3.55	-3.53	-3.51	-3.33	-3.34	-3.49	-3.20	-3.52
H4	-7.14	-0.71	-6.70	-2.20	-0.71	-0.71	-2.47	-2.47	-0.71	-0.71
H5	-39.82	-27.76	-29.72	-28.14	-31.24	-28.93	-27.96	-29.21	-17.90	-31.88
H6	-152.38	-10.87	-97.53	-430.40	-20.96	-26.49	-172.04	-278.15	-7.90	-265.90
H7	-45.78	-13.55	[b]	-20.87	-21.54	-14.21	-20.84	-46.83	-8.51	-25.35
H8	-390.64	-64.76	-575.63	-397.12	[c]	-326.02	-844.72	-638.17	-34.00	-392.20
H9	[a]	-8.24	-11.79	[a]	[c]	-8.26	-10.49	-31.61	-6.61	-9.36
H10	[a]	-9.16	[b]	[a]	[c]	-8.55	-19.75	-42.44	-7.00	-59.76
Average	-80.84	-14.15	-91.96	-110.90	-11.64	-41.93	-110.53	-107.62	-8.81	-79.19

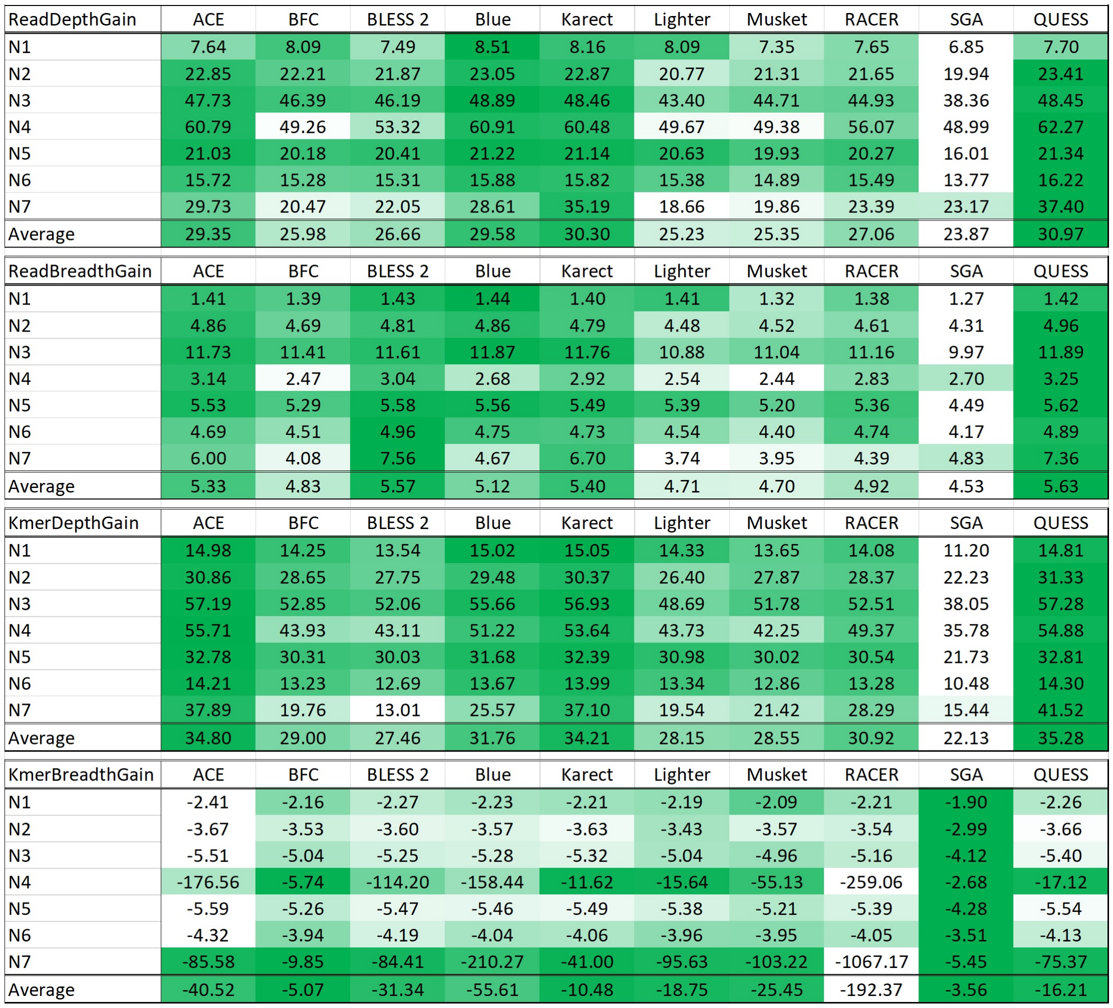
Table 4.3: **HiSeq Performance.** Comparison on program performance on HiSeq datasets (H1 – H10). **(Top to Bottom)** READDEPTHGAIN, READBREADTHGAIN, KMERDEPTHGAIN, KMERBREADTHGAIN. Darker colors indicate better results. [a] Maximum runtime exceeded (14 d). [b] "ERROR: Irregular quality score range". [c] Out of memory (>1TB).

The error correction comparison on HiSeq datasets (H1 – H10) is presented in Table 4.3 for each of the four measures: ReadDepthGain, ReadBreadthGain, KmerDepthGain, and KmerBreadthGain. It should be noted that some programs did not run for specific programs. In the case of ACE and Blue for datasets H9 and H10, the maximum runtime of 14 days was exceeded. BLESS 2 gave an error "Irregular quality score range 33-77" for dataset H7 and "Irregular quality score range 35-75" for dataset H10. Karect exceeded the maximum memory allotted of 1TB for datasets H8-H10.

With regards to ReadDepthGain results, QUESS does the best on average, closely followed up by Karect, which beats it in some datasets. However, Karect was unable to complete the larger jobs in H8–H10, which inflates its average correction. There is a similar trend with ReadBreadthGain results, in where SGA performs the worse, especially for the smaller datasets. However, ACE, Blue, Karect and QUESS are very close in terms of performance, except that the former three are unable to complete some of the larger datasets.

QUESS is the best performer with the KmerDepthGain measure, followed by Karect and ACE, and has a better KmerDepthGain for the largest datasets H9 and H10, in where it did not perform as well in ReadDepthGain. For KmerBreadthGain results, BFC and SGA are the clear leaders, but inferring from their other measures, it can be posited that they are doing relatively less correction. Karect also has an inflated average, due to being unable to correct the dataset that gave the other programs a lot of trouble (H8).

4.3.3 Comparison of NextSeq datasets



ReadDepthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
N1	7.64	8.09	7.49	8.51	8.16	8.09	7.35	7.65	6.85	7.70
N2	22.85	22.21	21.87	23.05	22.87	20.77	21.31	21.65	19.94	23.41
N3	47.73	46.39	46.19	48.89	48.46	43.40	44.71	44.93	38.36	48.45
N4	60.79	49.26	53.32	60.91	60.48	49.67	49.38	56.07	48.99	62.27
N5	21.03	20.18	20.41	21.22	21.14	20.63	19.93	20.27	16.01	21.34
N6	15.72	15.28	15.31	15.88	15.82	15.38	14.89	15.49	13.77	16.22
N7	29.73	20.47	22.05	28.61	35.19	18.66	19.86	23.39	23.17	37.40
Average	29.35	25.98	26.66	29.58	30.30	25.23	25.35	27.06	23.87	30.97

ReadBreadthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
N1	1.41	1.39	1.43	1.44	1.40	1.41	1.32	1.38	1.27	1.42
N2	4.86	4.69	4.81	4.86	4.79	4.48	4.52	4.61	4.31	4.96
N3	11.73	11.41	11.61	11.87	11.76	10.88	11.04	11.16	9.97	11.89
N4	3.14	2.47	3.04	2.68	2.92	2.54	2.44	2.83	2.70	3.25
N5	5.53	5.29	5.58	5.56	5.49	5.39	5.20	5.36	4.49	5.62
N6	4.69	4.51	4.96	4.75	4.73	4.54	4.40	4.74	4.17	4.89
N7	6.00	4.08	7.56	4.67	6.70	3.74	3.95	4.39	4.83	7.36
Average	5.33	4.83	5.57	5.12	5.40	4.71	4.70	4.92	4.53	5.63

KmerDepthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
N1	14.98	14.25	13.54	15.02	15.05	14.33	13.65	14.08	11.20	14.81
N2	30.86	28.65	27.75	29.48	30.37	26.40	27.87	28.37	22.23	31.33
N3	57.19	52.85	52.06	55.66	56.93	48.69	51.78	52.51	38.05	57.28
N4	55.71	43.93	43.11	51.22	53.64	43.73	42.25	49.37	35.78	54.88
N5	32.78	30.31	30.03	31.68	32.39	30.98	30.02	30.54	21.73	32.81
N6	14.21	13.23	12.69	13.67	13.99	13.34	12.86	13.28	10.48	14.30
N7	37.89	19.76	13.01	25.57	37.10	19.54	21.42	28.29	15.44	41.52
Average	34.80	29.00	27.46	31.76	34.21	28.15	28.55	30.92	22.13	35.28

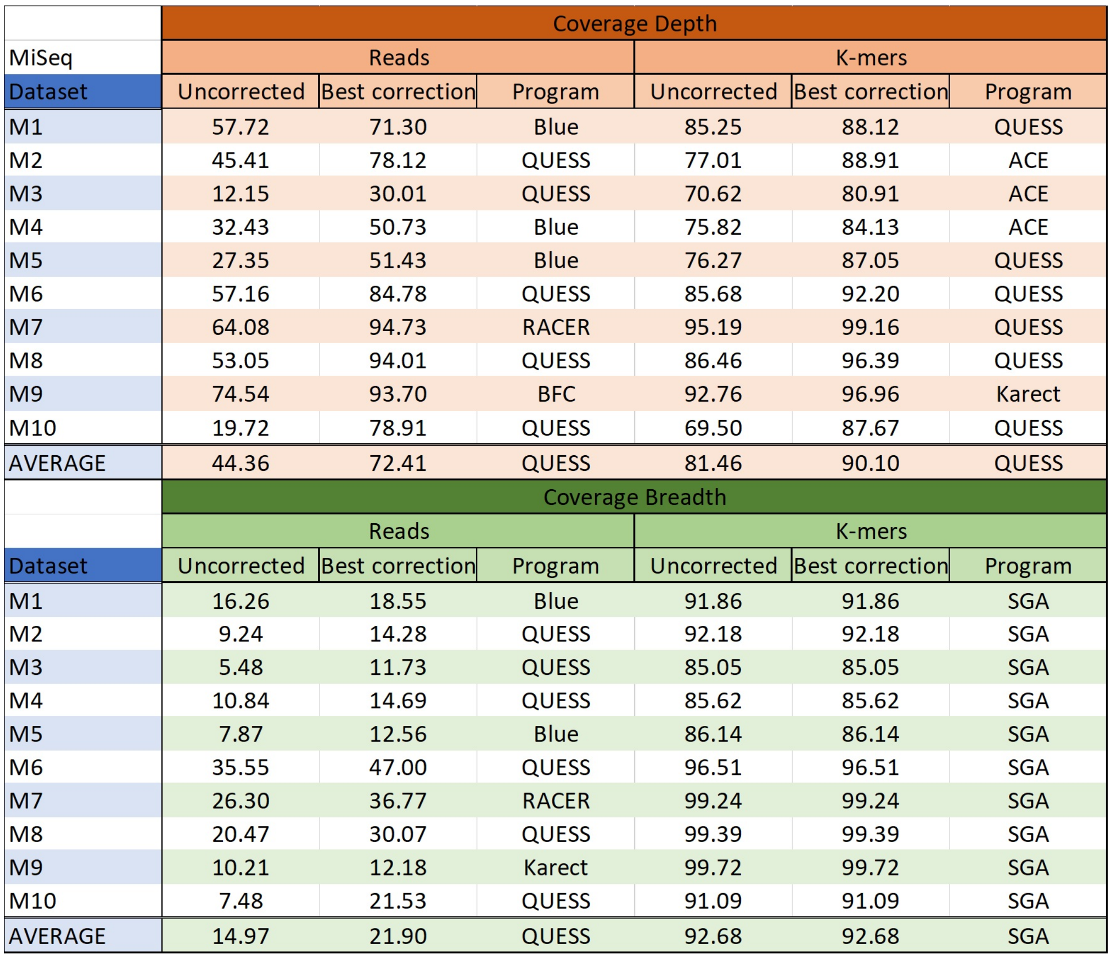
KmerBreadthGain	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
N1	-2.41	-2.16	-2.27	-2.23	-2.21	-2.19	-2.09	-2.21	-1.90	-2.26
N2	-3.67	-3.53	-3.60	-3.57	-3.63	-3.43	-3.57	-3.54	-2.99	-3.66
N3	-5.51	-5.04	-5.25	-5.28	-5.32	-5.04	-4.96	-5.16	-4.12	-5.40
N4	-176.56	-5.74	-114.20	-158.44	-11.62	-15.64	-55.13	-259.06	-2.68	-17.12
N5	-5.59	-5.26	-5.47	-5.46	-5.49	-5.38	-5.21	-5.39	-4.28	-5.54
N6	-4.32	-3.94	-4.19	-4.04	-4.06	-3.96	-3.95	-4.05	-3.51	-4.13
N7	-85.58	-9.85	-84.41	-210.27	-41.00	-95.63	-103.22	-1067.17	-5.45	-75.37
Average	-40.52	-5.07	-31.34	-55.61	-10.48	-18.75	-25.45	-192.37	-3.56	-16.21

Table 4.4: **NextSeq Performance.** Comparison on program performance on NextSeq datasets (N1 – N7). **(Top to Bottom)** READDEPTHGAIN, READBREADTHGAIN, KMERDEPTHGAIN, KMERBREADTHGAIN. Darker colors indicate better results.

The error correction comparison on NextSeq datasets (N1 – N7) is presented in Table 4.4 for each of the four measures: READDEPTHGAIN, READBREADTHGAIN, KMERDEPTH-GAIN, and KMERBREADTHGAIN.

With regards to READDEPTHGAIN results, QUESS performs the best on average, once again topping the most datasets, with Karect and Blue following behind, and SGA performing the worst. With both READBREADTHGAIN results, ACE, BLESS 2, Karect and QUESS have similar performances, with QUESS beating out the other three on average. With KmerDepthGain results, QUESS is the best on average, with ACE and Karect closely behind. With KmerBreadthGain results, as with both MiSeq and HiSeq datasets, SGA performs the best, but performs the worst on the other categories. RACER also appears to handle some datasets poorly, with outlier performances at N4 and N7. QUESS has the best average score amounts three of the four measures, except for KmerBreadthGain.

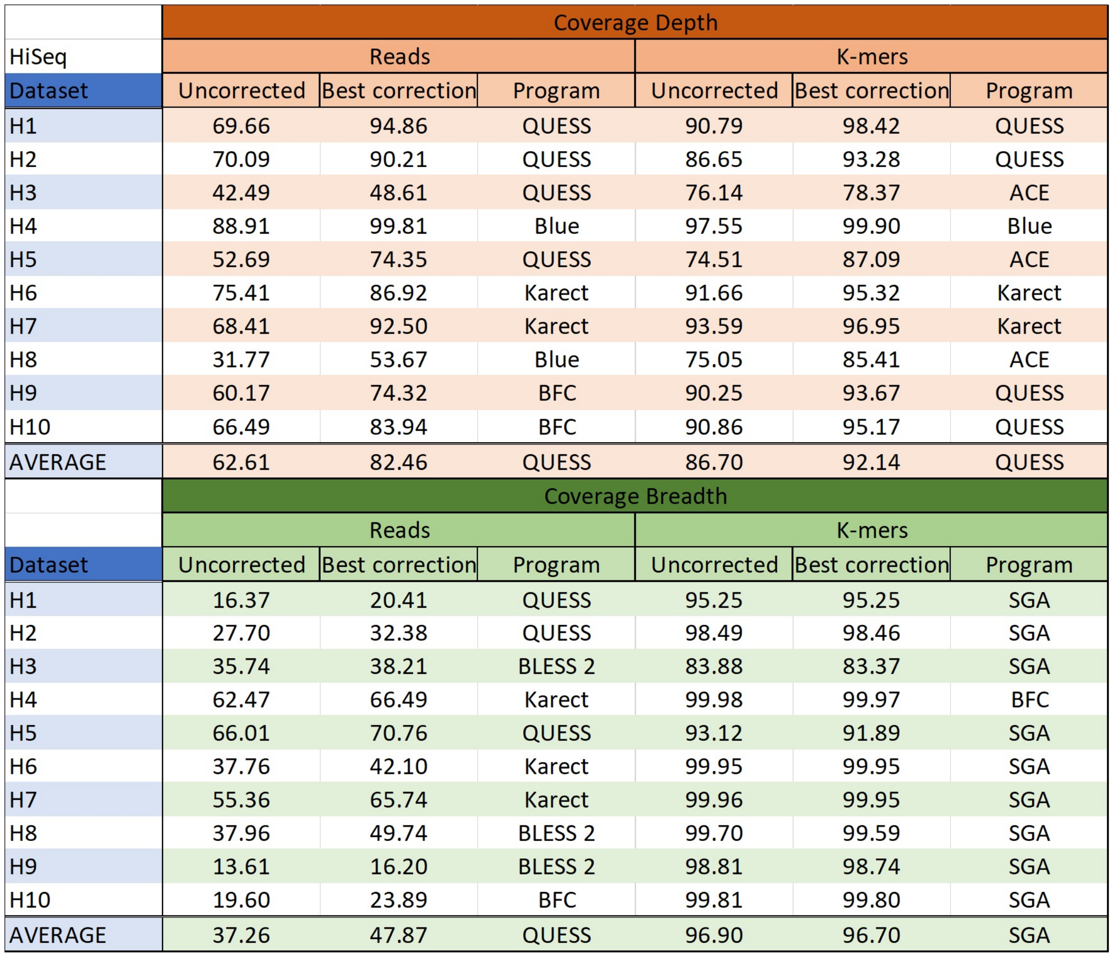
4.3.4 Overall Performance



	Coverage Depth					
MiSeq	Reads			K-mers		
Dataset	Uncorrected	Best correction	Program	Uncorrected	Best correction	Program
M1	57.72	71.30	Blue	85.25	88.12	QUESS
M2	45.41	78.12	QUESS	77.01	88.91	ACE
M3	12.15	30.01	QUESS	70.62	80.91	ACE
M4	32.43	50.73	Blue	75.82	84.13	ACE
M5	27.35	51.43	Blue	76.27	87.05	QUESS
M6	57.16	84.78	QUESS	85.68	92.20	QUESS
M7	64.08	94.73	RACER	95.19	99.16	QUESS
M8	53.05	94.01	QUESS	86.46	96.39	QUESS
M9	74.54	93.70	BFC	92.76	96.96	Karect
M10	19.72	78.91	QUESS	69.50	87.67	QUESS
AVERAGE	44.36	72.41	QUESS	81.46	90.10	QUESS
	Coverage Breadth					
	Reads			K-mers		
Dataset	Uncorrected	Best correction	Program	Uncorrected	Best correction	Program
M1	16.26	18.55	Blue	91.86	91.86	SGA
M2	9.24	14.28	QUESS	92.18	92.18	SGA
M3	5.48	11.73	QUESS	85.05	85.05	SGA
M4	10.84	14.69	QUESS	85.62	85.62	SGA
M5	7.87	12.56	Blue	86.14	86.14	SGA
M6	35.55	47.00	QUESS	96.51	96.51	SGA
M7	26.30	36.77	RACER	99.24	99.24	SGA
M8	20.47	30.07	QUESS	99.39	99.39	SGA
M9	10.21	12.18	Karect	99.72	99.72	SGA
M10	7.48	21.53	QUESS	91.09	91.09	SGA
AVERAGE	14.97	21.90	QUESS	92.68	92.68	SGA

Table 4.5: **MiSeq Performance Summary.** A collection of the best performers for each dataset for each measure; the depth measures are in orange and the breadth measures are in green.

Table 4.5 summarizes the programs that do the best at each dataset for each measure for MiSeq data. It can be seen that QUESS dominates most categories except for KMER-BREADTH. It can be noted that the changes in MiSeq datasets are due to changes in READDEPTH, which generally have a low percentage of correct reads beforehand, but gain a lot from correction. KMERDEPTH appears to be relatively high before and after correction, and READBREADTH appears to be relatively low before and after correction. KMERBREADTH is essentially not changed due to the best correction program SGA not yielding much result.



	Coverage Depth					
HiSeq	Reads			K-mers		
Dataset	Uncorrected	Best correction	Program	Uncorrected	Best correction	Program
H1	69.66	94.86	QUESS	90.79	98.42	QUESS
H2	70.09	90.21	QUESS	86.65	93.28	QUESS
H3	42.49	48.61	QUESS	76.14	78.37	ACE
H4	88.91	99.81	Blue	97.55	99.90	Blue
H5	52.69	74.35	QUESS	74.51	87.09	ACE
H6	75.41	86.92	Karect	91.66	95.32	Karect
H7	68.41	92.50	Karect	93.59	96.95	Karect
H8	31.77	53.67	Blue	75.05	85.41	ACE
H9	60.17	74.32	BFC	90.25	93.67	QUESS
H10	66.49	83.94	BFC	90.86	95.17	QUESS
AVERAGE	62.61	82.46	QUESS	86.70	92.14	QUESS

	Coverage Breadth					
	Reads			K-mers		
Dataset	Uncorrected	Best correction	Program	Uncorrected	Best correction	Program
H1	16.37	20.41	QUESS	95.25	95.25	SGA
H2	27.70	32.38	QUESS	98.49	98.46	SGA
H3	35.74	38.21	BLESS 2	83.88	83.37	SGA
H4	62.47	66.49	Karect	99.98	99.97	BFC
H5	66.01	70.76	QUESS	93.12	91.89	SGA
H6	37.76	42.10	Karect	99.95	99.95	SGA
H7	55.36	65.74	Karect	99.96	99.95	SGA
H8	37.96	49.74	BLESS 2	99.70	99.59	SGA
H9	13.61	16.20	BLESS 2	98.81	98.74	SGA
H10	19.60	23.89	BFC	99.81	99.80	SGA
AVERAGE	37.26	47.87	QUESS	96.90	96.70	SGA

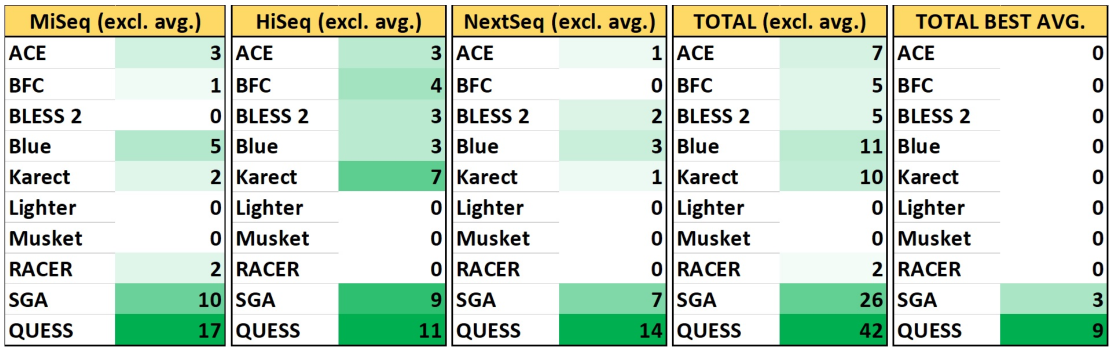
Table 4.6: **HiSeq Performance Summary.** A collection of the best performers for each dataset for each measure; the depth measures are in orange and the breadth measures are in green. Note that programs that fail to run for specific datasets are not accounted for in the average.

Table 4.6 summarizes the programs that do the best at each dataset for each measure for HiSeq data. It can once again be seen that QUESS performs the best in most categories except for SGA for KmerBreadth. Similar to MiSeq data, the greatest change in HiSeq data occurs in ReadDepth, albeit at a smaller amount, as these datasets have a greater initial measure as opposed to the MiSeq datasets. Also similar to MiSeq datasets is the high uncorrected and corrected KmerDepth and KmerBreadth values. However, initial ReadBreadth is a moderate level as compared to MiSeq datasets.

	Coverage Depth					
Nextseq	Reads			K-mers		
Dataset	Uncorrected	Best correction	Program	Uncorrected	Best correction	Program
H1	14.12	21.43	Blue	58.40	64.67	Karect
H2	18.07	37.25	QUESS	57.38	70.74	QUESS
H3	24.11	61.21	Blue	66.19	85.56	QUESS
H4	83.38	93.73	QUESS	93.30	97.03	ACE
H5	20.99	37.85	QUESS	67.05	77.86	QUESS
H6	26.90	38.76	QUESS	61.35	66.88	QUESS
H7	48.95	68.04	QUESS	84.62	91.00	QUESS
AVERAGE	33.79	51.03	QUESS	69.76	79.08	QUESS
	Coverage Breadth					
	Reads			K-mers		
Dataset	Uncorrected	Best correction	Program	Uncorrected	Best correction	Program
H1	4.90	6.28	Blue	68.08	67.47	SGA
H2	7.23	11.83	QUESS	78.47	77.82	SGA
H3	13.41	23.70	QUESS	84.58	83.95	SGA
H4	32.96	35.14	QUESS	99.95	99.94	SGA
H5	13.98	18.82	QUESS	79.18	78.29	SGA
H6	20.12	24.08	BLESS 2	79.44	78.72	SGA
H7	18.08	24.27	BLESS 2	99.88	99.87	SGA
AVERAGE	15.81	20.55	QUESS	84.23	83.72	SGA

Table 4.7: **NextSeq Performance Summary.** A collection of the best performers for each dataset for each measure; the depth measures are in orange and the breadth measures are in green.

Table 4.7 summarizes the programs that do the best at each dataset for each measure for NextSeq data. Compared to MiSeq and HiSeq datasets, NextSeq datasets have lesser values in all measures before and after correction, but follow the same trend as MiSeq data, in where there is most correction in ReadDepth and KmerBreadth, moderate correction in KmerDepth, and a small drop in KmerBreadth. QUESS is the best performer for all measures except for KmerBreadth.



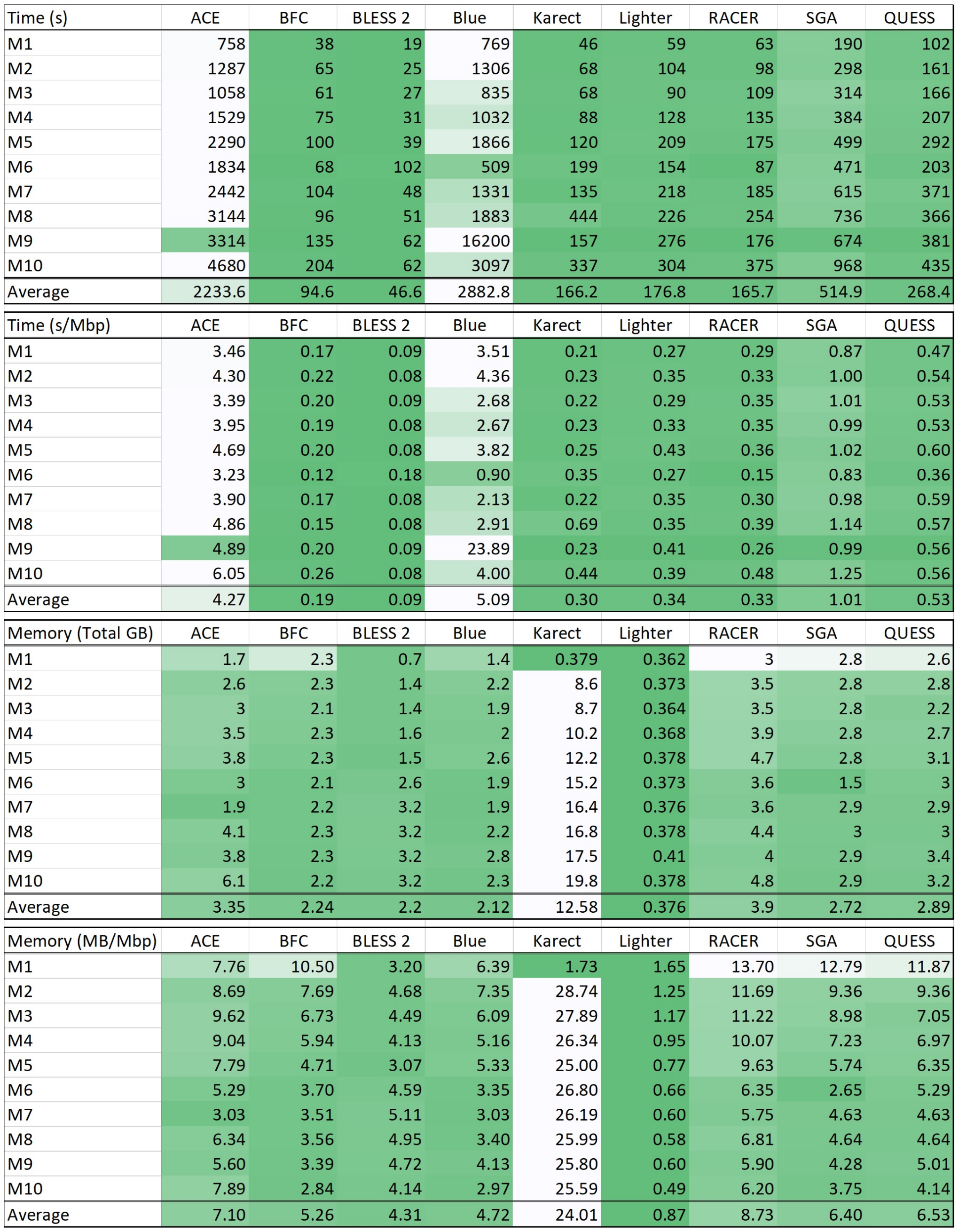
	MiSeq (excl. avg.)	HiSeq (excl. avg.)	NextSeq (excl. avg.)	TOTAL (excl. avg.)	TOTAL BEST AVG.
ACE	3	3	1	7	0
BFC	1	4	0	5	0
BLESS 2	0	3	2	5	0
Blue	5	3	3	11	0
Karect	2	7	1	10	0
Lighter	0	0	0	0	0
Musket	0	0	0	0	0
RACER	2	0	0	2	0
SGA	10	9	7	26	3
QUESS	17	11	14	42	9

Table 4.8: **Aggregation of Best Performers. (leftmost)** Tally of best measures for MiSeq datasets (M1 – M10). **(second from left)** Tally of best measures for HiSeq datasets (H1 – H10). **(centre)** Tally of best measures for NextSeq datasets (N1 – N7). **(second from right)** Tally of best measure for all datasets. **(rightmost)** A tally of top average performances for each four measures for each of the three types of datasets

We can observe that QUESS is the besr performer in the most datasets (Table 4.8). Overall, QUESS tops 42 out of 108 (38.9%) of all measures, or 42 out of 81 (51.9%) of measures that are not KmerBreadth. This contrasts the machine that does the second best, Blue, which performs best in 11 out of 81 (13.6%) measures. In total, QUESS performs the best, on average, in every single measure outside of KmerBreadth.

4.3.5 Time and Space

Time and space is provided in Tables 4.9 – 4.11. Each program is allotted 1TB of CPU memory, and has a maximum run time of 14 days. Each table is followed by a subsequent table denoting the normalized space and time measures that reflect the size of the input read bank size. Note that some datasets could not be run for specific programs, and the result is annotated at each relevant table.



Time (s)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	RACER	SGA	QUESS
M1	758	38	19	769	46	59	63	190	102
M2	1287	65	25	1306	68	104	98	298	161
M3	1058	61	27	835	68	90	109	314	166
M4	1529	75	31	1032	88	128	135	384	207
M5	2290	100	39	1866	120	209	175	499	292
M6	1834	68	102	509	199	154	87	471	203
M7	2442	104	48	1331	135	218	185	615	371
M8	3144	96	51	1883	444	226	254	736	366
M9	3314	135	62	16200	157	276	176	674	381
M10	4680	204	62	3097	337	304	375	968	435
Average	2233.6	94.6	46.6	2882.8	166.2	176.8	165.7	514.9	268.4

Time (s/Mbp)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	RACER	SGA	QUESS
M1	3.46	0.17	0.09	3.51	0.21	0.27	0.29	0.87	0.47
M2	4.30	0.22	0.08	4.36	0.23	0.35	0.33	1.00	0.54
M3	3.39	0.20	0.09	2.68	0.22	0.29	0.35	1.01	0.53
M4	3.95	0.19	0.08	2.67	0.23	0.33	0.35	0.99	0.53
M5	4.69	0.20	0.08	3.82	0.25	0.43	0.36	1.02	0.60
M6	3.23	0.12	0.18	0.90	0.35	0.27	0.15	0.83	0.36
M7	3.90	0.17	0.08	2.13	0.22	0.35	0.30	0.98	0.59
M8	4.86	0.15	0.08	2.91	0.69	0.35	0.39	1.14	0.57
M9	4.89	0.20	0.09	23.89	0.23	0.41	0.26	0.99	0.56
M10	6.05	0.26	0.08	4.00	0.44	0.39	0.48	1.25	0.56
Average	4.27	0.19	0.09	5.09	0.30	0.34	0.33	1.01	0.53

Memory (Total GB)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	RACER	SGA	QUESS
M1	1.7	2.3	0.7	1.4	0.379	0.362	3	2.8	2.6
M2	2.6	2.3	1.4	2.2	8.6	0.373	3.5	2.8	2.8
M3	3	2.1	1.4	1.9	8.7	0.364	3.5	2.8	2.2
M4	3.5	2.3	1.6	2	10.2	0.368	3.9	2.8	2.7
M5	3.8	2.3	1.5	2.6	12.2	0.378	4.7	2.8	3.1
M6	3	2.1	2.6	1.9	15.2	0.373	3.6	1.5	3
M7	1.9	2.2	3.2	1.9	16.4	0.376	3.6	2.9	2.9
M8	4.1	2.3	3.2	2.2	16.8	0.378	4.4	3	3
M9	3.8	2.3	3.2	2.8	17.5	0.41	4	2.9	3.4
M10	6.1	2.2	3.2	2.3	19.8	0.378	4.8	2.9	3.2
Average	3.35	2.24	2.2	2.12	12.58	0.376	3.9	2.72	2.89

Memory (MB/Mbp)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	RACER	SGA	QUESS
M1	7.76	10.50	3.20	6.39	1.73	1.65	13.70	12.79	11.87
M2	8.69	7.69	4.68	7.35	28.74	1.25	11.69	9.36	9.36
M3	9.62	6.73	4.49	6.09	27.89	1.17	11.22	8.98	7.05
M4	9.04	5.94	4.13	5.16	26.34	0.95	10.07	7.23	6.97
M5	7.79	4.71	3.07	5.33	25.00	0.77	9.63	5.74	6.35
M6	5.29	3.70	4.59	3.35	26.80	0.66	6.35	2.65	5.29
M7	3.03	3.51	5.11	3.03	26.19	0.60	5.75	4.63	4.63
M8	6.34	3.56	4.95	3.40	25.99	0.58	6.81	4.64	4.64
M9	5.60	3.39	4.72	4.13	25.80	0.60	5.90	4.28	5.01
M10	7.89	2.84	4.14	2.97	25.59	0.49	6.20	3.75	4.14
Average	7.10	5.26	4.31	4.72	24.01	0.87	8.73	6.40	6.53

Table 4.9: **MiSeq Run Time and Space Usage.** Runtimes and space usages of each program on each MiSeq dataset (M1 – M10). **(Top)** Raw Elapsed time. **(Second from Top)** Per mega base pair (10^6). **(Second from Bottom)** Raw memory usage in GB. **(Bottom)** Per mega base pair (10^6). Musket results are excluded due to resulting in outlier results. Darker colours indicate better results.

With regards to MiSeq datasets, ACE and Blue take magnitudes more time than the rest of the programs and Karect takes the most memory of all the programs by a substantial margin. In contrast, BLESS 2 and Lighter take the least time and memory among all the programs. QUESS is in the middle for both measures, and consistently utilizes approximately the same memory through all the MiSeq datasets.

Time (s)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
H1	576	22	11	1848	43	36	14	34	103	80
H2	1437	64	27	1278	124	113	55	92	369	179
H3	1092	35	27	283	71	88	43	42	348	138
H4	2356	94	45	990	132	179	56	151	562	292
H5	2718	118	43	1701	281	157	169	216	712	256
H6	67320	2963	1345	106200	6840	10800	2388	4320	17280	8280
H7	50400	1373	[b]	17640	6120	3600	2063	1996	15480	6840
H8	401760	16560	12600	301680	[c]	45360	70200	17280	78840	35640
H9	[a]	30960	59400	[a]	[c]	78120	119160	82080	186840	332640
H10	[a]	54000	[b]	[a]	[c]	75600	109080	63720	311400	327600
Average	65957.38	10618.90	9187.25	53952.50	1944.43	21405.30	30322.80	16993.10	61193.40	71194.50

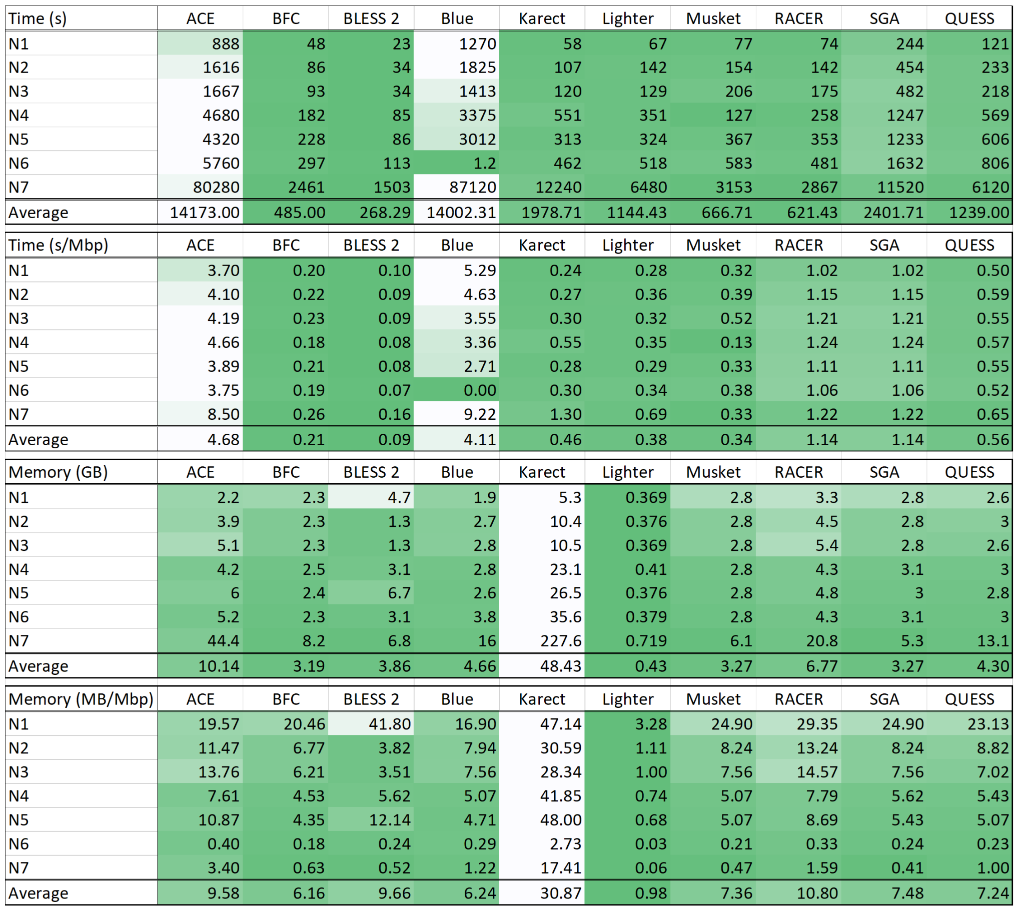
Time (s/Mbp)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
H1	5.12	0.20	0.10	16.44	0.38	0.32	0.12	0.30	0.92	0.71
H2	4.23	0.19	0.08	3.76	0.36	0.33	0.16	0.27	1.09	0.53
H3	2.95	0.09	0.07	0.76	0.19	0.24	0.12	0.11	0.94	0.37
H4	4.27	0.17	0.08	1.79	0.24	0.32	0.10	0.27	1.02	0.53
H5	4.92	0.21	0.08	3.08	0.51	0.28	0.31	0.39	1.29	0.46
H6	5.15	0.23	0.10	8.13	0.52	0.83	0.18	0.33	1.32	0.63
H7	3.86	0.11	[b]	1.35	0.47	0.28	0.16	0.15	1.18	0.52
H8	7.88	0.32	0.25	5.92	[c]	0.89	1.38	0.34	1.55	0.70
H9	[a]	0.22	0.41	[a]	[c]	0.54	0.83	0.57	1.30	2.31
H10	[a]	0.38	[b]	[a]	[c]	0.53	0.76	0.44	2.16	2.28
Average	4.80	0.21	0.15	5.15	0.38	0.46	0.41	0.32	1.28	0.90

Memory (GB)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
H1	1.5	2.1	5.6	1.3	3.4	0.368	2.6	0.48	2.7	2.5
H2	2.2	2.2	1.5	2	9.4	0.378	2.8	3.2	2.8	2.9
H3	1.6	2.1	1.5	1.4	9.9	0.365	0.53	1.4	2.8	2.7
H4	1.7	2.2	2.3	1.7	13	0.376	2.8	3.3	2.8	3
H5	5.7	2.2	4.4	2.4	13.3	0.363	2.8	3.5	2.9	2.7
H6	47.4	6.2	6.8	12.9	303.6	0.809	6.1	27.7	6.7	14
H7	48.3	6.1	[b]	12.4	308.8	0.856	4.4	18.6	6.5	15.9
H8	261.5	25.5	6.9	31.4	[c]	1	48.3	85.3	17.3	25.1
H9	[a]	73.7	7.8	[a]	[c]	12.3	70.6	306.8	38.2	338.5
H10	[a]	66.4	[b]	[a]	[c]	9.4	90.5	243	40.4	313.6
Average	46.24	18.87	4.60	8.19	94.49	2.62	23.14	69.33	12.31	72.09

Memory (MB/Mbp	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
H1	13.34	18.68	49.81	11.56	30.24	3.27	23.13	4.27	24.01	22.24
H2	6.47	6.47	4.41	5.88	27.65	1.11	8.24	9.41	8.24	8.53
H3	4.32	5.67	4.05	3.78	26.72	0.99	1.43	3.78	7.56	7.29
H4	3.08	3.99	4.17	3.08	23.55	0.68	5.07	5.98	5.07	5.43
H5	10.32	3.99	7.97	4.35	24.09	0.66	5.07	6.34	5.25	4.89
H6	3.63	0.47	0.52	0.99	23.24	0.06	0.47	2.12	0.51	1.07
H7	3.70	0.47	[b]	0.95	23.63	0.07	0.34	1.42	0.50	1.22
H8	5.13	0.50	0.14	0.62	[c]	0.02	0.95	1.67	0.34	0.49
H9	[a]	0.51	0.05	[a]	[c]	0.09	0.49	2.13	0.27	2.35
H10	[a]	0.46	[b]	[a]	[c]	0.07	0.63	1.69	0.28	2.18
Average	6.25	4.12	8.89	3.90	25.59	0.70	4.58	3.88	5.20	5.57

Table 4.10: **HiSeq Run Time and Space Usage.** Runtimes and space usages of each program on each HiSeq dataset (H1 – H10). **(Top)** Raw Elapsed time. **(Second from Top)** Per mega base pair (10^6). **(Second from Bottom)** Raw memory usage in GB. **(Bottom)** Per mega base pair (10^6). [a] Maximum runtime exceeded (14 d). [b] "ERROR: Irregular quality score range". [c] Out of memory (>1TB). Darker colours indicate better results.

With regards to HiSeq datasets, as with MiSeq datasets, ACE and Blue take magnitudes more time than the rest of the programs and Karect takes the most memory of all the programs by a substantial margin. In addition, ACE and Blue were unable to finish within the maximum allotted time of 14 days for 2 datasets H9 and H10. Karect utilized more than the allotted CPU memory of 1TB for three datasets H8-H10. Due to these incomplete jobs, the relative ranking of other programs is affected. Of the programs that completed all datasets, BFC is the fastest and QUESS is the slowest (ACE is estimated to take the longest), and Lighter uses the least memory, whereas QUESS uses the most (Karect, ACE and Blue are expected to take more memory).



Time (s)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
N1	888	48	23	1270	58	67	77	74	244	121
N2	1616	86	34	1825	107	142	154	142	454	233
N3	1667	93	34	1413	120	129	206	175	482	218
N4	4680	182	85	3375	551	351	127	258	1247	569
N5	4320	228	86	3012	313	324	367	353	1233	606
N6	5760	297	113	1.2	462	518	583	481	1632	806
N7	80280	2461	1503	87120	12240	6480	3153	2867	11520	6120
Average	14173.00	485.00	268.29	14002.31	1978.71	1144.43	666.71	621.43	2401.71	1239.00

Time (s/Mbp)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
N1	3.70	0.20	0.10	5.29	0.24	0.28	0.32	1.02	1.02	0.50
N2	4.10	0.22	0.09	4.63	0.27	0.36	0.39	1.15	1.15	0.59
N3	4.19	0.23	0.09	3.55	0.30	0.32	0.52	1.21	1.21	0.55
N4	4.66	0.18	0.08	3.36	0.55	0.35	0.13	1.24	1.24	0.57
N5	3.89	0.21	0.08	2.71	0.28	0.29	0.33	1.11	1.11	0.55
N6	3.75	0.19	0.07	0.00	0.30	0.34	0.38	1.06	1.06	0.52
N7	8.50	0.26	0.16	9.22	1.30	0.69	0.33	1.22	1.22	0.65
Average	4.68	0.21	0.09	4.11	0.46	0.38	0.34	1.14	1.14	0.56

Memory (GB)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
N1	2.2	2.3	4.7	1.9	5.3	0.369	2.8	3.3	2.8	2.6
N2	3.9	2.3	1.3	2.7	10.4	0.376	2.8	4.5	2.8	3
N3	5.1	2.3	1.3	2.8	10.5	0.369	2.8	5.4	2.8	2.6
N4	4.2	2.5	3.1	2.8	23.1	0.41	2.8	4.3	3.1	3
N5	6	2.4	6.7	2.6	26.5	0.376	2.8	4.8	3	2.8
N6	5.2	2.3	3.1	3.8	35.6	0.379	2.8	4.3	3.1	3
N7	44.4	8.2	6.8	16	227.6	0.719	6.1	20.8	5.3	13.1
Average	10.14	3.19	3.86	4.66	48.43	0.43	3.27	6.77	3.27	4.30

Memory (MB/Mbp)	ACE	BFC	BLESS 2	Blue	Karect	Lighter	Musket	RACER	SGA	QUESS
N1	19.57	20.46	41.80	16.90	47.14	3.28	24.90	29.35	24.90	23.13
N2	11.47	6.77	3.82	7.94	30.59	1.11	8.24	13.24	8.24	8.82
N3	13.76	6.21	3.51	7.56	28.34	1.00	7.56	14.57	7.56	7.02
N4	7.61	4.53	5.62	5.07	41.85	0.74	5.07	7.79	5.62	5.43
N5	10.87	4.35	12.14	4.71	48.00	0.68	5.07	8.69	5.43	5.07
N6	0.40	0.18	0.24	0.29	2.73	0.03	0.21	0.33	0.24	0.23
N7	3.40	0.63	0.52	1.22	17.41	0.06	0.47	1.59	0.41	1.00
Average	9.58	6.16	9.66	6.24	30.87	0.98	7.36	10.80	7.48	7.24

Table 4.11: **NextSeq Run Time and Space Usage.** Runtimes and space usages of each program on each NextSeq dataset (N1 – H7). **(Top)** Raw Elapsed time. **(Second from Top)** Per mega base pair (10^6). **(Second from Bottom)** Raw memory usage in GB. **(Bottom)** Per mega base pair (10^6).

With regards to NextSeq datasets, similarly to MiSeq and HiSeq data, ACE and Blue are the slowest programs, and BLESS 2 and BFC are the fastest. Karect takes the most memory, whereas Lighter takes the least memory. QUESS is on the slower end of programs, but uses a relatively lesser amount of memory.

Chapter 5

Conclusions

5.1 Discussion

It can be concluded that programs having higher ReadDepthGain and ReadBreadthGain result in having fully correct reads, and this aids in saving incorrect reads from the input read bank. Similarly, programs with high ReadBreadthGain illustrate a stronger ability to make corrections that will allow for better genome assembly, due to their increased coverage of the reference genome. Programs with high KmerBreadthGain correspond to programs that are conservative in nature, and make little changes, relative to programs with lower scores in this measure.

It should be noted that the negative values represented by KmerBreadthGain could be a result of programs that miscategorize unique k-mers, and miscorrect them into a more frequent and similar k-mer. Inherently, correction of programs tends to bring datasets of high diversity into a more ordered set of reads. This is a problem that arises from lack of coverage or uneven coverage, as this miscategorizes less frequent k-mers within the dataset to be unique, and unique reads with low contextual support is a prime target for correction.

From observation of the performance from each measure, we can characterize each program for its strengths and weaknesses. ACE runs an aggressive algorithm, leading to the greatest loss in KmerBreadth among all programs constantly, but produces good results in the Depth measures as a result. BFC runs a non-greedy algorithm, and this results in having decent but not leading results in a lot of measures; it also runs faster and with less memory than its leading competitors. BLESS 2, Lighter and Musket, which are all Bloom filter based algorithms, are faster and use less memory, but result in worse measurements as compared to other algorithms; as a result, they rarely lead in any measures. Karect runs a memory-intensive algorithm, resulting in strong contention for leading position in a lot of members, but it also uses magnitudes more memory than every other program, unable to run medium sized programs, even with 1 TB of memory. RACER can be seen as a less aggressive algorithm compared to ACE, achieving decent scores across all measures, beating the Bloom filter algorithms, but not enough to lead. SGA is a slow and conservative algorithm, leading to small corrections with minimal loss in KmerBreadth.

5.2 Final Thoughts

The strongest performer in the majority of measures is QUESS, which tops in 42 out of 108 (38.9%) total measures, or 42 out of 81 (51.9%) non-KmerBreadthGain measures. As aforementioned, QUESS is unequivocally the best program to use on an entirely new dataset with no reference genome assembled as of yet, as it consistently has the greatest scores in most measures, rarely having outlier results. In addition, while there are programs that run faster than it, and use less memory than it, QUESS outperforms those programs in all measures. In addition, it runs faster and on less memory than other leading competitors, who either run out of time or memory on the largest datasets.

Bibliography

[1] A. Allam, P. Kalnis, and V. Solovyev. Karect: accurate correction of substitution, insertion and deletion errors for next-generation sequencing data. *Bioinformatics*, 31(21):3421–3428, Nov 2015. [DOI:10.1093/bioinformatics/btv415] [PubMed:26177965].

[2] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commu. ACM*, 13:422–426, 1970.

[3] A. Califano and I. Rigoutsos. FLASH: a fast look-up algorithm for string homology. *Proc Int Conf Intell Syst Mol Biol*, 1:56–64, 1993. [PubMed:7584371].

[4] L. Demetrius and M. Ziehe. Darwinian fitness. *Theor Popul Biol*, 72(3):323–345, Nov 2007. [DOI:10.1016/j.tpb.2007.05.004] [PubMed:17804030].

[5] P. Greenfield, K. Duesing, A. Papanicolaou, and D. C. Bauer. Blue: correcting sequencing errors using consensus and context. *Bioinformatics*, 30(19):2723–2732, Oct 2014. [DOI:10.1093/bioinformatics/btu368] [PubMed:24919879].

[6] Y. Heo, A. Ramachandran, W. M. Hwu, J. Ma, and D. Chen. BLESS 2: accurate, memory-efficient and fast error correction method. *Bioinformatics*, 32(15):2369–2371, Aug 2016. [DOI:10.1093/bioinformatics/btw146] [PubMed:27153708].

[7] L. Ilie, S. Ilie, and A. M. Bigvand. SpEED: fast computation of sensitive spaced seeds. *Bioinformatics*, 27(17):2433–2434, Sep 2011. [DOI:10.1093/bioinformatics/btr368] [PubMed:21690104].

[8] L. Ilie and M. Molnar. RACER: Rapid and accurate correction of errors in reads. *Bioinformatics*, 29(19):2490–2493, Oct 2013. [DOI:10.1093/bioinformatics/btt407] [PubMed:23853064].

[9] Illumina. Illumina sequencing platforms, 2017. Retreived from: https://www.illumina.com/systems/sequencing-platforms.html.

[10] H. Li. BFC: correcting Illumina sequencing errors. *Bioinformatics*, 31(17):2885–2887, Sep 2015. [PubMed Central:PMC4635656] [DOI:10.1093/bioinformatics/btv290] [PubMed:25953801].

[11] M. Li, B. Ma, D. Kisman, and J. Tromp. Patternhunter II: highly sensitive and fast homology search. *J Bioinform Comput Biol*, 2(3):417–439, Sep 2004. [PubMed:15359419].

[12] Y. Liu, J. Schroder, and B. Schmidt. Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics*, 29(3):308–315, Feb 2013. [DOI:10.1093/bioinformatics/bts690] [PubMed:23202746].

[13] B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, Mar 2002. [PubMed:11934743].

[14] A. E. Minoche, J. C. Dohm, and H. Himmelbauer. Evaluation of genomic high-throughput sequencing data generated on Illumina HiSeq and genome analyzer systems. *Genome Biol.*, 12(11):R112, Nov 2011. [PubMed Central:PMC3334598] [DOI:10.1186/gb-2011-12-11-r112] [PubMed:22067484].

[15] M. Molnar and L. Ilie. Correcting Illumina data. *Brief. Bioinformatics*, 16(4):588–599, Jul 2015. [DOI:10.1093/bib/bbu029] [PubMed:25183248].

[16] F. Sanger, S. Nicklen, A. R. Coulson, F. Sanger, S. Nicklen, and A. R. Coulsen. DNA sequencing with chain-terminating inhibitors. 1977. *Biotechnology*, 24:104–108, 1992. [PubMed:1422003].

[17] S. C. Schuster. Next-generation sequencing transforms today's biology. *Nat. Methods*, 5(1):16–18, Jan 2008. [DOI:10.1038/nmeth1156] [PubMed:18165802].

[18] S. Sheikhizadeh and D. de Ridder. ACE: accurate correction of errors using K-mer tries. *Bioinformatics*, 31(19):3216–3218, Oct 2015. [DOI:10.1093/bioinformatics/btv332] [PubMed:26026137].

[19] J. T. Simpson and R. Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.*, 22(3):549–556, Mar 2012. [PubMed Central:PMC3290790] [DOI:10.1101/gr.126953.111] [PubMed:22156294].

[20] L. Song, L. Florea, and B. Langmead. Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biol.*, 15(11):509, 2014. [PubMed Central:PMC4248469] [DOI:10.1186/s13059-014-0509-9] [PubMed:25398208].

[21] Sponk. Difference DNA RNA-EN, 2010. [Edited] CC BY-SA 3.0.

[22] K.A. Wetterstrand. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program, 2017.

[23] X. Yang, S. P. Chockalingam, and S. Aluru. A survey of error-correction methods for next-generation sequencing. *Brief. Bioinformatics*, 14(1):56–66, Jan 2013. [DOI:10.1093/bib/bbs015] [PubMed:22492192].

Curriculum Vitae

Name:	Stephen Lu
Post-Secondary Education and Degrees:	University of Western Ontario London, Ontario, Canada 2016 - present M.Sc. candidate
	Western University London, Ontario 2011 - 2015 B.MSc.
Honours and Awards:	Dean's Honor List (2016, 2012) The Western Scholarship of Distinction (2011)
Related Work Experience:	Teaching Assistant The University of Western Ontario 2016 - present
	Research Assistant – Ilie Lab The University of Western Ontario 2016 - present