

# DISTRIBUTED STREAM-PROCESSING FRAMEWORK FOR GRAPH-BASED SEQUENCE ALIGNMENT

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE  
OF BILKENT UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR  
THE DEGREE OF  
MASTER OF SCIENCE  
IN  
COMPUTER ENGINEERING

By  
Alim Şükrücan Gökkaya  
January 2020

Distributed Stream-Processing Framework for Graph-based Sequence  
Alignment

By Alim Şükrücan Gökkaya

January 2020

We certify that we have read this thesis and that in our opinion it is fully adequate,  
in scope and in quality, as a thesis for the degree of Master of Science.

---

Can Alkan(Advisor)

---

Eray Tüzün

---

Tolga Can

Approved for the Graduate School of Engineering and Science:

---

Ezhan Kardeşan  
Director of the Graduate School

# ABSTRACT

## DISTRIBUTED STREAM-PROCESSING FRAMEWORK FOR GRAPH-BASED SEQUENCE ALIGNMENT

Alim Şükrücan Gökkaya

M.S. in Computer Engineering

Advisor: Can Alkan

January 2020

Optimized the sequence alignment pipelines are needed to minimize the time required to complete processing the short-read genomic data. Today there are many sequence alignment tools exist, yet few of them are capable of directly ingesting the streaming base-call data. The sequencing has to be entirely completed before the mainstream aligners can begin mapping the reads to the reference. The sequencing process can take days to complete. The output is then needs to be demultiplexed into individual reads and aligned to the reference, which can take several more hours. Overall time of a genomic analysis can be shortened significantly by progressively computing the alignments at the time when the reads are still being generated. It is important to have genomic analysis done as quickly as possible, especially in life critical situations.

Here we introduce a distributed stream processing framework for aligning short-reads into a graph representation of the genome. The massively parallel nature of the genomic sequencing data requires a massively parallel computation architecture. Thus we have designed our pipeline called R2G<sup>2</sup>Flow to align many reads to a de Bruijn graph in parallel. Our aligning method is specialized for the sequencing technologies that are based on base-call cycles, such as produced by Illumina. The results are made available soon after the final bases from the sequencing devices has been emitted.

R2G<sup>2</sup>Flow is available at <https://github.com/BilkentCompGen/r2g2>

*Keywords:* read mapping, de Bruijn graphs, stream processing.

## ÖZET

# ÇİZGE TABANLI OKUMA HIZALANDIRMASI İÇİN DAĞITIK AKINTI İŞLEME SİSTEMİ

Alim Şükrücan Gökkaya  
Bilgisayar Mühendisliği, Yüksek Lisans  
Tez Danışmanı: Can Alkan  
Ocak 2020

Kısa okuma genom verilerinin işleme süresini en aza indirmek için optimize edilmiş okuntu hizalama sistemleri gerekmektedir. Günümüzde birçok dizilim hizalama aracı mevcut, fakat bunlardan sadece birkaçı akıntı halindeki baz-çağrışımlarını doğrudan işleyebilme yeteneğine sahiptir. Anaakım hizalayıcıların okuntuları referansa hizalamaya başlayabilmesinden önce okuma işleminin bütünüyle tamamlanması gerekir. Okuma işleminin tamamlanması günler sürebilir. Çıktılar daha sonra, çoğullama çözme işlemiyle, tekil okumalara dönüştürülür, bu işlem fazladan bir kaç saat daha sürebilir. Uçtan uca genom analiz süresi, yeni okumalar henüz üretilmekte iken hizalandırılmanın aşamalı olarak hesaplanması halinde, önemli miktarda kısaltılabilir. Özellikle hayati durumlarda genom analizinin mümkün olduğunca çabuk yapılması önem taşımaktadır.

Bu tez, kısa okumaların genom çizge yapılarına hizalandırılması için dağıtık akıntı işleme sistemi sunar. Genom okuma verilerinin yüksek miktarda paralel veri sunan doğasına karşılık yüksek miktarda paralel veri işleyebilen bilgisayar mimarisi gerekir. Bu nedenle R2G<sup>2</sup>Flow adlı sistemimizi bir çok okumayı aynı anda de Bruijn çizgesine hizalayabilecek şekilde tasarladık. Yöntemimiz Illumina gibi baz-çağrışım tabanlı okuma teknolojileri için özelleştirilmiştir. Sonuçlar okuma aygıtından son bazlar üretildikten kısa bir süre sonra çıkarılır.

*Anahtar sözcükler:* okuntu haritalandırması, de Bruijn çizgeleri, akıntı işleme.

## Acknowledgement

First and foremost I would like to express my sincerest gratitude to my supervisor, Asst. Prof. Can Alkan, for his invaluable guidance and support that he have invested to me through my graduate studies.

I would like to express my gratitude to my thesis comitee, Asst. Prof. Eray Tüzün and Prof. Tolga Can for their valuable reviews that are crucial for the completion of this thesis.

I would like to thank current and past members of the Bilkent Bioinformatics and Computational Genomics Group, Halil İbrahim Özercan, Balanur İçen, Fatma Kahveci, Mohammed Alser, Fatih Karaođlanođlu, Ezgi Ebren, Zülal Bingöl and Can Fırtına.

I would also express my thanks to my coworkers Alperen Yeşil, Sabit Gökberk Karaca, Burak Sefa Sert, Ceyhun Özgün and Mert Çalışkan at Atlassian Opsgenie team that I have joined in the last year. Their support allowed me to follow my graduate studies unhindered.

Finally, I would like to thank my family members Lütfi, Ayşe, Alpaslan, Ömer Gökkaya for their permanent love, patience and support through my education and entire life. None of this would be possibe without them.

# Contents

- 1 Introduction** **1**
- 1.1 Background . . . . . 2
- 1.1.1 Sequencing . . . . . 2
- 1.1.2 The Human Genome Project and the Linear Reference  
            Genome . . . . . 5
- 1.1.3 Mapping to a Linear Reference . . . . . 5
- 1.1.4 Graph Genomes . . . . . 7
- 1.1.5 de Bruijn Graphs . . . . . 8
- 1.1.6 Stream Processing . . . . . 10
- 1.1.7 Our Contribution . . . . . 11
- 
- 2 Methods** **13**
- 2.1 Indexing . . . . . 15
- 2.1.1 Graph Construction . . . . . 15

2.1.2	K-mer Index . . . . .	15
2.1.3	Unitig Index . . . . .	16
2.2	Alignment Pipeline . . . . .	16
2.2.1	Read Demultiplexer . . . . .	16
2.2.2	$K$ -mer Lookup . . . . .	18
2.2.3	Processing Candidate Seeds . . . . .	19
2.2.4	Graph Alignment . . . . .	20
2.2.5	Output . . . . .	20
2.2.6	Read Termination Events . . . . .	21
2.2.7	Backpressure and Out-of-Order Events . . . . .	21
2.3	Graph Server . . . . .	21
2.4	BCL Broker . . . . .	22
<b>3</b>	<b>Results</b>	<b>23</b>
3.1	Simulation Details . . . . .	23
3.1.1	Fastq2BCL Tool . . . . .	24
3.2	Experiments with Bacterial Genome . . . . .	24
3.2.1	Simulated Read Alignment . . . . .	24
3.2.2	$K$ -mer Cache Improvements . . . . .	25
3.3	Experiments with Human Genome . . . . .	26

3.3.1	Simulated Variants . . . . .	28
3.4	Reference Location Resolution . . . . .	31
<b>4</b>	<b>Discussion and Future Work</b>	<b>33</b>
4.1	Mission Critical Streaming Alignments . . . . .	33
4.2	Distributed Execution Environment . . . . .	34
4.3	Future Work . . . . .	35
4.3.1	Improved de Bruijn Graph Alignment Heuristics . . . . .	35
4.3.2	Integration of Other Types of Graph Aligners . . . . .	35
4.3.3	Post-alignment Analysis Flows . . . . .	35
<b>A</b>	<b>Glossary</b>	<b>44</b>
<b>B</b>	<b>Code</b>	<b>45</b>



# List of Figures

2.1	Architecture of the aligner . . . . .	14
2.2	Streaming graph alignment data flow . . . . .	17
3.1	Histogram of read counts by matched length for E. Coli experiment	25
3.2	Timing diagram for practical applications . . . . .	28
3.3	Histogram of read counts by matched length for human genome experiment . . . . .	29
3.4	Alignment coordinate accuracy in E. Coli dataset . . . . .	31

# List of Tables

3.1	Results for mapping 1M simulated short-reads on E. Coli UMN026 compared to the other tools. All tasks are executed on a single host with 112 cores. Remarkd run times are shown without the demultiplexer time. . . . .	24
3.2	Comparison of k-mer lookup performance with respect to the $k$ -mer LRU cache and batch query settings using 50K reads and 8 CPU cores . . . . .	26
3.3	Results for mapping 9.5M simulated short-reads on GRCh38.Chr19 assembly compared to the other tools on single host with 112 cores. Remarkd run times shown without added time cost of demultiplexing, which might take up to an hour in practical applications.	27
3.4	Results for mapping 1M simulated short-reads on GRCh38.Chr19 on single host with 112 cores. . . . .	27
3.5	Results for mapping 9.4M simulated short-reads from the sample generated by VarSim from the reference GRCh38.Chr19 assembly. All alignments are performed on single host with 112 cores. Remarkd run times shown without added time cost of demultiplexing, which might take up to an hour in practical applications.	30

# Chapter 1

## Introduction

Every cellular organism stores its genetic material in DNA molecules that are made of long chains of nucleotide base pairs. Human genome consists of about 3 billion bases. Every individual organism carries a copy of the genome with small deviations, which makes their DNA a unique sequence. After the Human Genome Project was completed in 2003, many new instruments have been developed to extract the sequences from the DNA molecules, and the advent of High Throughput Sequencing (HTS) technologies made it cheaper and faster to do so. Due to these advancements, we are now able to perform widespread genome sequencing for both research and diagnostic purposes [1]. Today, it is possible to extract readings from an entire genome in a few days at a very little cost. However, the immense amount of data generated by the sequencing instruments prove both computational and engineering challenges to process and annotate the data in a useful way for the researchers or physicians. One of these challenges is to ensure the results from the analysis are available as early as possible by beginning the analysis of the genetic data while it is still being generated by the sequencing instrument. This is especially important for the time critical tasks such as the clinical use of the genome sequencing that is becoming no longer a novelty, but an ordinary diagnostic method.

In most applications, the analysis of the sequenced genomic data depends on

mapping the reads to a known reference genome. Traditionally, the reference genome is represented as a single linear continuous sequence that can be obtained from a certain individual. In contrast to the linear genomes, graph representations of genomes can provide much more powerful analyses as they combine the genetic information from a population where many known genetic variations were annotated. Today, there are many studies such as the 1000 Genomes Project [2] that highlight those genetic variations in the human populations. Researchers continually analyze the functional aspects of genomic variants to identify their roles in specific phenotypes or diseases.

In this thesis, we propose a framework to perform read alignments to a genome graph in near real-time and using a distributed and parallel computational infrastructure. In the following section, we give the necessary background information about the concepts related to genomics and stream-processing.

## 1.1 Background

### 1.1.1 Sequencing

Genome sequencing is defined as the process of determining the exact order of bases in the DNA molecule chains obtained from a biological sample. From a computational view, DNA can be seen as a sequence of characters from a small alphabet of 4 letters, each corresponding to a type of base ( $\Sigma = \{A, C, G, T\}$ ). Today, there are a variety of HTS instruments that have been made commercially available, each providing trade offs between the read length, accuracy and cost-per-megabase characteristics [3]. Read length, corresponds to the number of sequential bases that the sequencing device is able to provide for a single read from a DNA chain. Accuracy defines the rate of mistakes made during the sequencing, such as mixing-up between the bases (substitutions), skipping some bases (deletions) or addition of extra bases (insertions). The characteristics of the sequencing instruments may have significant influence over the computational

strategies to be used for alignment. Below, we give an overview of some relevant genome sequencing technologies.

#### **1.1.1.1 Sanger Sequencing**

Sanger sequencing [4] is the ancestor to many of the later generations of sequencing technologies. It has been regarded as the first successful method for sequencing DNA fragments above the length of 500 base pairs with very little inaccuracy below 0.01% error rate [1]. Sanger's chain termination method involves addition of modified nucleotides that terminate the natural synthesis of the opposite strand by the DNA polymerase enzyme. This allows stochastically generating different sized copies originating from the same DNA fragment with a radiologically or fluorescently labeled nucleotide at the end. The terminated DNA fragments are then sorted by their length using electrophoresis and photographed to extract the sequence. Despite the good accuracy and read length characteristics, Sanger sequencing remains cost and time prohibitive for many applications. Sanger sequencing was the predominant method for over 30 years since its invention and the Human Genome Project was completed largely using this method. Today this method is still being used for benchmarking the novel HTS technologies or small scale studies.

#### **1.1.1.2 High Throughput Sequencing Technologies**

The advancements in the HTS technologies made it possible to sequence an entire human genome within a few days with high fidelity and at a fraction of the cost with the Sanger sequencing. Some HTS technologies can achieve massive throughput by relying on having the DNA into cut into fragments even smaller than it is in the Sanger sequencing. Other methods exist to produce reads at much larger lengths, but they are in general much more expensive and error prone than the short-read methods.

**1.1.1.2.1 Short Read Sequencing** Illumina sequencing platform was released in 2008 and became the dominant sequencing platform in the market due to its ability to cheaply generate high quality short reads [5]. Illumina sequencer simultaneously reads the base-pairs starting from millions of different locations throughout the genome. The underlying technology works by attaching the small cuts from one strand of the DNA into a slide of glass and iteratively synthesizing the other strand while observing the light emitted when a new base is attached to the DNA. Akin to Sanger sequencing, the nucleotides used in this step are modified to pause the synthesis until the next cycle. In each cycle one nucleotide is attached to every DNA chains that were attached to the slide and the light emitted during the event is captured by a camera. At the end of each cycle, the slide is prepared for the next cycle by washing off the excess molecules and cleaving off the part of the modified nucleotide that pauses the synthesis. Furthermore, Illumina platform is also capable of sequencing paired end reads, in other words, the read from the opposite strand of the DNA fragment. Read length of the Illumina sequencers averages at 150 base pairs with 0.1% error rate.

**1.1.1.2.2 Long Read Sequencing** Long read technologies are segmented into two categories, single-molecule methods and synthetic long reads. Synthetic long read methods are based on augmenting the short read sequencing technologies to preserve more information on whereabouts of the DNA fragment on the originating DNA chain [6]. Single-molecule methods are based on rapidly sampling the signals emitted during the chemical processing of the DNA molecule. PacBio platform adopts sequence by synthesis method. Reads are observed by a camera in real-time while the DNA is being synthesized by a modified DNA polymerase enzyme fixed inside a nanoscale pit. Oxford Nanopore technology threads the DNA molecule from a nanoscale hole and samples the electrical current in the process.

### **1.1.2 The Human Genome Project and the Linear Reference Genome**

A fully sequenced human genome was published for the first time in 2003 [7] as a product of a 13 year long effort committed to the Human Genome Project (HGP). It is estimated that in total \$3 billion was spent for the project by various institutions across the globe [8]. The scope of the HGP was more than only sequencing the human DNA and it included many important achievements such as identification of all protein coding genes, mapping the genome for regions of interest and mapping the genetic variations. Only Sanger-based sequencing methods were used. The reads obtained from sequencing was assembled into continuous strings corresponding to each chromosome. The reference genome assembly was built from the consensus obtained by sequencing the genome of a few individuals. In this way the reference genome represents a model human genome as a single continuous string. However, a linear reference is not suitable to maintain a comprehensive representation for entire species.

Most of the HGP reference assembly donors were of European ancestry. Due to the lack of diversity in the donor group, the reference assembly suffers from gaps and biases in some regions [9, 10]. It is difficult to overcome of these problems using the linear genome representations since the genetic variations among the different human populations cannot be expressed without building multiple assemblies. Using the traditional mapping methods, one can only align the reads to a single linear reference at a time. Thus the genomic variations cannot be easily discovered through linear reference methods.

### **1.1.3 Mapping to a Linear Reference**

A key step to the analysis of the genomic data is to align the sequenced DNA strings to previously build genome assembly, such as the reference genome. This process is often called read alignment or read mapping. Aligning the reads to a reference genome shows the similarities and differences between the two genomes

so that further processing can be done such as eliminating the sequencing errors, identifying genetic variations or discovering a mutation that may lead to a disease. Optimal sequence alignment problem was formally defined by Levenshtein as the minimum number of edits required to obtain the target string from the initial one, where each edit is either addition, subtraction or substitution of one letter [11]. A dynamic programming solution was described by Needleman-Wunsch [12]. Later Smith-Waterman described local alignment algorithm by extending the previous work [13]. Local alignment determines the longest substrings from the source and target sequences that have the largest score based on the given similarity metric. Utilization of a similarity metric is especially useful alignment of biological sequences because each kind of substitution, subtraction or addition of a base-pair can be assigned a predetermined cost value that is proportional to their natural occurrence in the input data.

The vast amount of data provided by the HTS instruments requires faster alignment algorithms to be implemented. HTS devices are fast, but they also produce less reliable output. To overcome the drawbacks of the increased error rates with the HTS data, the genome is sequenced with higher coverage. By super-sampling the genetic data, aligned reads can be later used for obtaining a consensus sequence to statistically eliminate the sequencing errors. The sufficient amount of sequencing coverage rate depends largely on the error rate and read-depth capabilities of the sequencing platform used. For Illumina platform, it is advised to have at least 30x to 50x coverage in human genome studies [14]. Due to the quadratic time and space requirements of the optimal alignment algorithm, most of the short-read alignment strategies are evolved around seed-and-extend-based heuristics [15]. Earlier aligners developed for Sanger reads such as BLAST [16] became successful by eliminating large amounts of regions from the reference from dynamic programming comparison by selecting short segments of a few consecutive characters from the query that shows high similarity to the segments of the same length from reference. Gaps between the matched segments are later filled using Smith-Waterman algorithm to finalize the alignment of the read.



### 1.1.3.1 Seed Selection Strategies

Common approaches for selecting seeds for short-read alignment can be categorized into hash-based and suffix-array-based indexing approaches to identify exact matches. Hash-based approaches use efficient hash strategies such as FastHASH [17] and minimal perfect hash function (MPHF) [18] to index the fixed size segments ( $k$ -mers) from the genome for exact matching. Suffix-array methods often use Burrows-Wheeler Transform (BWT) of the reference sequence with the support of FM-index for sparsely indexing the matching locations for memory efficiency with little extra computational cost [19, 20]. Compressed suffix-trees were also used for indexing genome graphs [21, 22, 23]. Limasset et al. [24] described a read mapping strategy which used MPHF index to identify seed nodes in a de Bruijn graph [25]. Finding good seeds are also proven to be difficult especially for the regions of the genome with high number of repetitions. In order to improve the seed quality, some modern aligners use minimizers to eliminate low quality matches [26, 27, 28]. A minimizer is a filtering function that selects the least frequently occurring  $k$ -mer within all candidate matches from the same proximity. This method is usually preferred for long read mapping tasks due to the increased number of false positives generated by the higher read error characteristics that may increase the computation cost significantly. GraphAligner [28] uses de Bruijn graph generated from short reads for error correcting the long reads before finding seeds. For the de Bruijn graph alignment, the seeds were chosen from near the start and end of the read only. In both [24] and [28],  $k$ -mers were indexed for seed selection using an implementation of the minimal perfect hash function.

### 1.1.4 Graph Genomes

Many of the existing read aligners target a reference genome that is represented as a single continuous string. Due to the shortcomings of the linear references, pan-genome studies prefer to use graph genomes to fully utilize the diverse features obtained from large set of genomic data at hand [29]. Graph aligners can

map the reads to the non-linear representations of the genome, which may be useful when studying population genomes or complex regions of the genome with high number of repetitions. String graphs and sequence graphs are commonly used for read alignment tasks [30, 28, 23]. The bit-parallel alignment approach used in [30] was later expanded by the authors using Navaro’s [31] adaptation of dynamic-programming alignment to graphs in the GraphAligner [28]. Another data structure that efficiently can represent a genome is the de Bruijn graph (DBG) [32]. De Bruijn graphs are useful for many tasks such as read assembly, error correction, variant discovery and read mapping. In the next section, the details of the DBG data structure and the alignment strategies are covered.

### 1.1.5 de Bruijn Graphs

De Bruijn graphs are constructed from the set of all substrings of length  $k$  of the given input set of sequences. Such substrings are commonly referred as  $k$ -mers. For a given set of strings  $S_{1..n}$  in an alphabet  $\Sigma$ ,  $DBG_k(S) = (V, E)$  is a de Bruijn graph of order  $k$  where:

$$\begin{aligned}
 V &= \{p \in \Sigma^k \mid \exists i \in \{1..n\}, \text{ s.t. } p \text{ is a } k\text{-mer} \in S_i\} \\
 E &= \{(p, p') \mid \exists t \in \Sigma^{k+1}, \text{ s.t. } p \text{ is the prefix of } t \text{ and} \\
 &\quad p' \text{ is the suffix of } t\}
 \end{aligned}$$

A notable property of de Bruijn graph is that all occurrences of a  $k$ -mer are collapsed into a single node. This property makes the de Bruijn graphs especially useful for genome assembly applications; once the graph constructed from the short read data where the  $k$  is equal to the read depth, any traversal of the graph leads to a possible assembly. For the read mapping tasks, the graph can be efficiently constructed from multiple pre-assembled genomes. In this case, each of the input genomes exist as a path in that can be obtained from the resulting de Bruijn graph. However, for the highly repetitive parts of the genome, the resulting path may contain cycles due to the collapsing property. Overlapping cycles can further complicate the graph and lead to many false-positive paths

exists in the graph. Therefore not all of paths found in the graph has to be present as a substring of an input sequence.

Due to the information loss occurred during de Bruijn graph construction, read mapping tasks can be challenging. There are a few aligners have shown that sequence alignment to de Bruijn graph is feasible [33, 24]. A recent study focusing on read assembly [34] shows that the de Bruijn graphs that are augmented with long-range information can also increase read mapping accuracy for the highly repetitive parts of the genome.

#### 1.1.5.1 Read Alignment on de Bruijn Graphs

The first practical application of HTS read alignment on de Bruijn graphs was conducted by deBGA [33] with the motivation of mapping the reads to multiple genomes at once instead of aligning to a singular linear genome at a time as the mainstream aligners typically do. In contrast to linear genomes, the problem of optimally aligning reads to de Bruijn graphs has been shown to be NP-complete [24]. Different variation of the seed-and-extend based alignment heuristics include local alignment over unipath position sets (as used by deBGA [33]) and greedy extension of seeds (as used by BGREAT [24] and McCortex [34]). In deBGA, seeds are first merged to find the maximal exact matches and the location information regarding the unipaths in the input are collected. After seed processing, the gaps between the seeds from the adjacent locations in the same genome is filled using dynamic programming. On the other hand, greedy extension methods used by BGREAT and McCortex does not extend the seeds on the original input sequences based on an optimal alignment. Instead, the reads are aligned to the graph directly using a greedy depth-first search algorithm. This type of alignment provide more ambiguous results because typical read alignment annotations such as the loci and edit distance are not computed. The output is still suitable for error correction of reads and identification of potential genomic variations. The BrownieAligner [35] demonstrated that the seed extension performance can be significantly improved by using a cost function that is derived from the read data and a Markov model that is derived from the reference genome sequences.

## 1.1.6 Stream Processing

Stream processing systems allow immediate processing of the data produced by continuous data sources with minimum amount of delay. In recent years, many powerful platforms have been published with distributed processing capabilities and fault-tolerance. Core principle to the distributed processing systems is to divide the work into self-standing operators with definitive inputs and outputs. With this mindset, the applications can achieve very large throughput by adding more hosts to the system so that more instances of the operators can be run. One of such platforms is the Apache Flink [36]. Flink is an open-source platform with enterprise grade capabilities, supporting many execution environments such as Hadoop YARN and Kubernetes.

### 1.1.6.1 Apache Flink

**1.1.6.1.1 Architecture and Operations** Flink has an distributed stream processing architecture that provides an abstraction with the application logic and the execution environment. Each Flink program defines an execution plan that layouts an order of atomic transformations to be performed on the underlying data stream. The instructions are defined using Java or Scala APIs are then serialized and published to the task execution cluster. Flink Applications can define operations such as low level process functions, map/reduce functions and window functions for batch processing. In addition, a *key* can be defined on a stream to efficiently perform operations on a certain aspect of the data without changing the event order. Such streams are named *keyed streams* and the functions defined on them become *keyed functions*. Using keyed stream, one can compute complex aggregation that are grouped by a specific aspect. Each Flink application typically define a source function that produce events to down stream operators and a sink function to digest the final events.

**1.1.6.1.2 State Management** In the ideal case, an event processing function should be idempotent. Such functions can be distributed to any host without

any side effects since the function is not maintaining any internal or global state. However in many applications, pure functions cannot be efficiently implemented. Flink provides a state management infrastructure such that the functions can register their state in a safe way. At any time, a host can be removed from the cluster. In that case the sub-tasks can be redistributed to another host without data loss if the state was managed by Flink storage engine. Keyed data streams have separate state management for each key in the data stream. Depending on the application needs, in-memory and file based state management back ends can be chosen by the application developer.

#### **1.1.6.2 Streaming Alignment**

Real-time alignment methods are required to be designed with respect to the low level output characteristics of the specific HTS equipments. For Illumina short reads, this means that there has to be a simultaneous alignment procedure corresponding to each of the individual DNA fragments attached to the flowcell from the start of the sequencing run. Depending on the setup, there might up to a few billion read alignments in parallel. HiLive [37] implements a BWT based aligner that consumes raw Illumina BCL data from the file system. Alignment state of each read is kept in the file system until the next cycle in order to avoid overloading the memory. HiLive does not support distributed execution out-of-the-box, yet it can finish the read alignment as soon as the sequencing run is ended. There is also a streaming pipeline [38] made for BCL data which feeds reads to the BWA tool [39], but read alignment is blocked until the final cycle of the sequencing run is completed.

#### **1.1.7 Our Contribution**

We have created a sequence-to-graph alignment pipeline in a distributed stream processing environment specialized for Illumina short reads. Our computation model is a streaming implementation of seed-and-extend based graph alignment

heuristics that avoids unnecessarily delaying the read alignment until sequencing run to be completed, in contrast to many existing alignment pipelines do. The results of analysis are made soon after the last sequencing cycle is completed. Distributed nature of our aligner enables taking advantage of horizontally scaled computation for genomic analysis tasks.

# Chapter 2

## Methods

In a streaming short-read alignment task, all reads has to be processed in parallel as the next base in each of the reads are gradually emitted by the processing cycles of the sequencing device. Thus, we designed an alignment pipeline called R2G<sup>2</sup>Flow, that can scale up with the space and computational power required by the large number of reads can be provided by the HTS equipment (Figure 2.1). Our system has two main components that execute the main alignment logic, a Flink alignment job and a graph alignment service. This separation allows us to define the alignment jobs without depending the graph data directly. Both alignment job and the graph service can be scaled up individually by adding more instances. In addition, we implement an indexing tool to correlate the nodes of the graph with the positions of the input genome and a file broker service that connects sequencing equipment to the aligner.

Main motivation of the project is to adapt the sequence-to-graph aligners to a stream processing system. Aligning reads to a graph genome is an key step for the further analysis of genomic variants and population related studies. For that reason, we chose to implement the greedy seed-and-extend based heuristics similar to state-of-the-art de Bruijn graph aligners [24, 35]. Since finding the optimal alignment of a read to a de Bruijn graph is proven to be NP-complete [24] we output the first feasible mapping with the limited contextual information

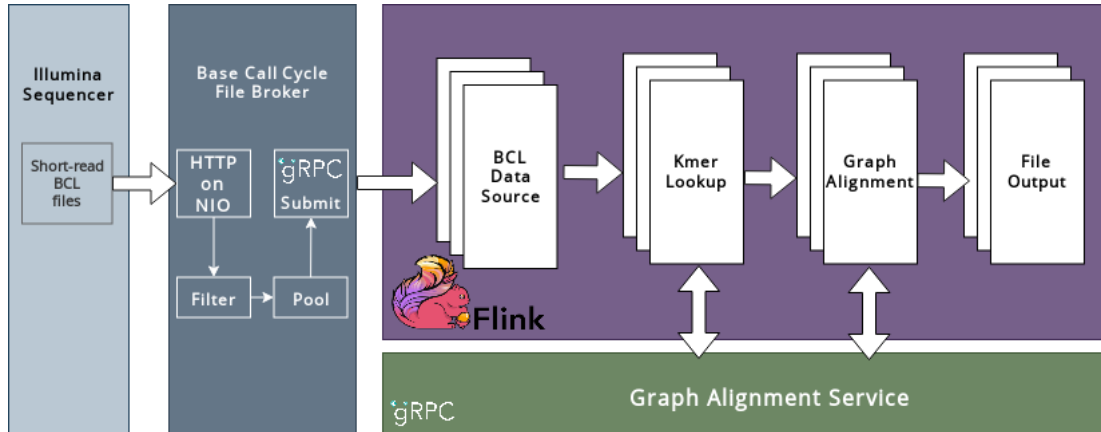


Figure 2.1: Overview of the R2G<sup>2</sup>Flow pipeline. Raw sequencing files are fed to the system through the broker service in BCL format. Alignment job is deployed to the Flink cluster to consume many files in parallel. State of each parallel read alignment is kept within the Flink alignment job only. Thus each call to the graph alignment service is an idempotent call, allowing graph service to be scaled behind a load balancing proxy.

around the seed rather than performing a semi-global alignment.

Additionally we explore further innovations in greedy sequence-to-graph alignment methods to support outputting the possible mapping locations on the original reference sequences. The greedy aligners we know [24, 34, 35] do not necessarily provide positional information for the alignment. We include a novel processing step in our pipeline determine the possible locations in the original reference sequences so that the more detailed mapping information can be extracted.

**Input format** We take BCL frames as input where in each cycle  $k$  of the sequencing run, a frame is emitted to contain the base at the  $k$ th position of the reads contained in a specific cluster in the flowcell. We identify each read by their cluster id and the index of the read within the cluster.

**Output format** As a result from the alignment operation, we find a path in the reference de Bruijn graph for each read in the input. When unitig-to-reference



location index is available, possible mapping locations in the original reference is also provided in the output.

## 2.1 Indexing

We enhance the de Bruijn graph with an additional  $k$ -mer index to determine the unitig identifier in order to locate the match positions on the original input sequences. By having a bidirectional index on the  $k$ -mer set, we can efficiently map the reads on the de Bruijn graph then extract the possible alignment locations.

### 2.1.1 Graph Construction

From given set of reference sequences we build a reference de Bruijn graph. This graph is used for identifying the seeds and extending them to come up with a sequence to graph mapping. We used the GATB tool [40] build the reference graph. Since the graph creation tool is usually targets the read data, the  $k$ -mers are filtered by occurrence frequency to eliminate sequencing errors. We disabled the  $k$ -mer abundance filter to ensure that all input  $k$ -mers included in the graph because the reference strings are assumed to contain no errors.

### 2.1.2 K-mer Index

$K$ -mers are stored in a minimal perfect hash index available as part of the GATB tool [41]. Minimal hash indexes are efficient data structures to keep large number of entries with minimal amount of space requirements.

### 2.1.3 Unitig Index

A unitig is a maximal non-branching path in the de Bruijn graph. To keep the minimal memory footprint, a hash index is used for keeping the reference positions per unitig and a suffix array [42] to keep  $k$ -mer position within each unitig. We use BCALM2 [43] to extract unitigs from the graph, Then we populate a lookup table for the reference positions each unitig. Once a  $k$ -mer is mapped to a unitig, the lookup table is used for extracting locations in the reference strings.

## 2.2 Alignment Pipeline

The graph alignment seed-and-extend heuristic algorithm is divided into the a series of functional operations. Each operation tackles one responsibility by reacting to a certain event in the data stream. Due to the streaming nature of the data, we employ aspects of functional programming while structuring our alignment method. Thus, we designed an interface between the streaming part of the system and static part of the system to be a set of idempotent operations. The streaming part of the system is responsible to keep track of each on-going alignment operation, such as demultiplexing the base calls into  $k$ -mers and keeping track of the seed and gaps in a read alignment. However the static part of the system encapsulate the operations on the reference graph only, such as  $k$ -mer lookup and searching for a path between given two nodes of the graph.

### 2.2.1 Read Demultiplexer

In non-streaming alignment tasks, short-reads are demultiplexed usually into FASTQ formatted files in advance so that the aligner can iterate over each read and perform alignment. We have integrated the read demultiplexing step into our alignment pipeline minimize the processing delay. Since the seeding step of graph alignment algorithm relies on determining exact matches to the nodes of

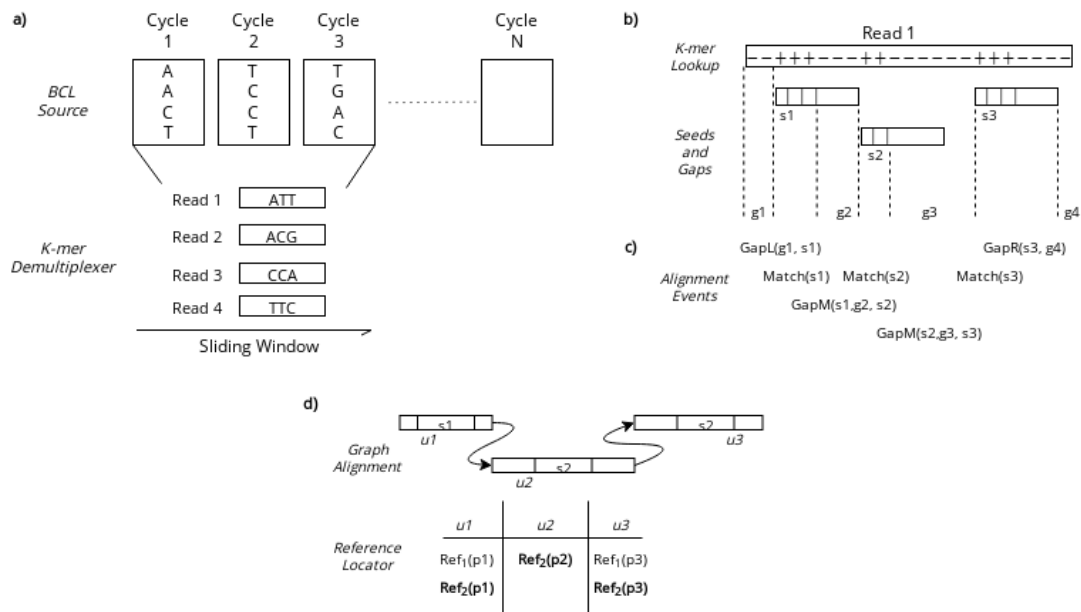


Figure 2.2: Data flow in the streaming read aligner. On the left-hand side of each step the functional operation name is given. (a) Raw BCL data is demultiplexed into a  $k$ -mer stream using a sliding window of size  $k$ . (b) Positions in the read is marked (+) or (-) to signify whether the  $k$ -mer is found in the graph, consecutive  $k$ -mers are merged to create windows of gaps or seeds. (c) A stream of alignment events are formed. (d) Alignment stream is processed to form a path in the graph where the read is aligned. Unitigs are a highlighted for querying the actual genomic positions to output the most likely positions in the reference.

the de Bruijn graph, we are only interested in the fixed size window of sequences from each read. Read demultiplexer function produces a stream of such  $k$ -mers in the same order as the read. Computation of the  $k$ -kmer stream is performed efficiently using bitwise shift operations. Thus the output of the demultiplexer is  $KmerEvent(r_i, p_c)$  where  $r_i$  is a unique identifier of the read,  $p_c$  is a pattern of  $k$  bases at as of cycle  $c$ .

## 2.2.2 $K$ -mer Lookup

The  $k$ -mer index is queried during the streaming alignment flow to identify exact matches to the graph. The queries are made to the  $k$ -mer index are batched and done through asynchronous I/O methods to reduce the cost of the each query. We select a sufficiently large batch size to minimize the networking overhead and use non-blocking calls to avoid blocking the processing threads unnecessarily.

### 2.2.2.1 $K$ -mer Sharding and Lookup Cache

Additionally we have implemented a caching method further reduce the computational cost of the  $k$ -mer lookup operation by allocating a least-recently-used cache (LRU-cache) to each processing thread. To do so, the  $k$ -mer stream is sharded by the value of  $p \bmod T$  where  $p$  is the numerically encoded value of a  $k$ -mer and  $T$  is the number of processing threads allocated for the operation. This ensures that a specific  $k$ -mer sequence is always assigned to the same physical instance of the demultiplexer in the Flink cluster. With this shortcut we keep the query results for the most frequently occurring  $k$ -mers in the cache so that redundant queries are avoided. The  $k$ -mer lookup cache is kept in the thread local memory to keep all cache operations lock-free.

## 2.2.3 Processing Candidate Seeds

### 2.2.3.1 Maximizing Seeds

Consecutive  $k$ -mer hits are grouped together to find maximum spanning exact matches. The seed maximization is implemented using a finite state machine that takes a *KmerEvent* as an input and keeps track of the beginning and ending intervals of the immediate context before the current  $k$ -mer. The state of the machine is defined as *AlignmentContext*( $p_{branching}, p_{solid}, s_{extension}$ ) where  $p_{branching}$  is the last solid  $k$ -mer before a discordant block of read,  $p_{solid}$  is the first  $k$ -mer in a concordant block of read and  $s_{extension}$  is the additional sequences in the current block. Upon a transition between a discordant block and a concordant block, a *PartialAlignment* event is emitted to the stream. The partial alignment event contains the corresponding begin and end *KmerEvents*.

### 2.2.3.2 Greedy Chaining

Partial alignments are chained into graph alignment events where each event correspond to an operation on the graph. The possible output events are gap and match events within a read (Figure 2.2.c). The chaining method is greedy because only one exact maximized matches are before the current event is considered for extension. Ideally seeds should be chained globally and all possible extensions should be explored to identify the path that minimizes the alignment cost. However this not possible during stream processing since it requires the read to be fully sequenced. As a result, events corresponding to gaps and matches from the read are stream to the output to be processed by the graph alignment operation.

## 2.2.4 Graph Alignment

Graph alignment function invokes the graph server to execute the given path search problem on the graph. The gaps between the seeds are filled using depth-first search algorithm and the context from the graph. Additionally left-most and right-most seeds can also be extended to the read boundaries. The contextual information is always provided by the streaming aligner to the graph server for the gap alignment queries. The gaps are filled using a branch-and-bound algorithm [44, 24]. The gap events contain the context from read around one or two anchor points where the exact matches occur. Similar to the [24], we assign cost function for each base that did not match the sequence from the read. All path search request calls are made in an asynchronous stream to avoid blocking the event processing threads.

### 2.2.4.1 Extracting Positions in the Reference

Reference position lookup takes place after a path in the de Bruijn graph obtained. During the graph alignment process, we return the unitig identifier that the seed was mapped if the unitig information was available. We extract the set of reference positions from the reference table that we have precomputed. Finally the most frequently mapped references are selected to present the approximate locations that the read is mapped in the reference genomes. Note that a read may be aligned to multiple reference sequences due to common variations and repeats.

## 2.2.5 Output

The graph alignment results are written to the file system in text format where the name, original sequence and results from the mapping events are recorded in each line. For unsuccessful mappings, only the read name and the sequence is written. Otherwise the path from the graph is highlighted. If the reference positions were extracted from the reference genome, a separate file is produced to

include the sequence of the genomic positions and read positions for each read.

## 2.2.6 Read Termination Events

Since the streaming read alignment consists of reactive event processing functions that we have explained in 2.2, the down stream processing functions should be notified when a read is terminated, i.e., when the HTS equipment signals that the last sequencing cycle is completed. To handle this, we implement additional input cases for each operation in the pipeline to flush the batched data and finalize the read alignment.

## 2.2.7 Backpressure and Out-of-Order Events

In some cases, the slowest operation in the stream processing pipeline causes the input events to back-up. This may introduce performance problems as the waiting queue takes more and more memory. Thanks to the Flink data stream architecture, we can handle this problem at the source by introducing the backpressure to the upstream processors. We have ensured that our implementation can reflect the backpressure up to the BCL file uploading step. Thus, if the cluster is overloaded, the further load will not be added to the system until the system catches up with the existing work load.

## 2.3 Graph Server

The graph server is an essential part of the streaming alignment pipeline where the graph related operations are abstracted from the stream processing operations. We have established an graph service interface that provides the functional endpoints to perform the seed discovery and seed extension tasks required by graph-based seed-and-extend algorithms. Our graph server implementation

performs heuristics similar to the BGREAT [24]. However more robust heuristics can also be applied without making significant changes to the service interface.

## 2.4 BCL Broker

We implement a HTTP service utility to publish raw to the stream alignment pipeline. This design allows running the pipeline independently of the HTS equipment. The broker service is a high-performance application where the files are uploaded to the alignment pipeline using non-blocking calls. A file upload operation ensures that the BCL data frame is actually takes its place in the stream processing steps of the graph alignment application.



# Chapter 3

## Results

We have evaluated our method using simulated read data to compare performance and accuracy against other short-read alignment tools; BWA [39], BGREAT2 (successor to BGREAT [24]) and BrownieAligner [35]. We select BWA as the baseline to compare the performance characteristics of the R2G<sup>2</sup>Flow to the traditional short-read alignment methods. BGREAT2 and BrownieAligner are both greedy aligners for de Bruijn graphs with state-of-the-art seed-and-extend heuristics. Since our tool is the only greedy de Bruijn graph aligner to extract genomic positions from the original input references, we compare our alignment results with BWA only for evaluating the accuracy of the aligner. For these experiments we build the reference graph using a single reference genome in order to make reasonable comparison with the BWA tool.

### 3.1 Simulation Details

We used ART [45] to simulate read data from Illumina HiSeq 2500 equipments. We used predefined settings that are provided by ART in order to approximate the error profile of the physical equipment. The read length is chosen to be 150 base pairs.

### 3.1.1 Fastq2BCL Tool

The simulated read data needs to be converted to the base call cycle based format for the streaming alignment. We developed an utility for extracting multiplexed BCL frames from the given FASTQ file and uploading the frames to the broker service. Thus we can simulate end-to-end streaming alignment without having to rely on actual sequencing device.

## 3.2 Experiments with Bacterial Genome

### 3.2.1 Simulated Read Alignment

We have generated 1 million reads from the UMN026 strain of the E. Coli genome [46]. Approximately  $30\times$  read-depth is achieved in this data set due to small size of the bacterial genome. We have compared the performance and read alignment quality of R2G<sup>2</sup>Flow with the other aligners (Table 3.2.1). As the baseline BWA run shows, it is possible to have all reads mapped back to the reference successfully. Furthermore R2G<sup>2</sup>Flow demonstrated similar read alignment quality in terms of alignment coverage to the state-of-the-art de Bruijn graph aligners (Figure 3.1).

Table 3.1: Results for mapping 1M simulated short-reads on E. Coli UMN026 compared to the other tools. All tasks are executed on a single host with 112 cores. Remarkd run times are shown without the demultiplexer time.

Tool Name	#Aligned	Aligned%	Coverage%	Run Time
BWA	1,000,000	100%	100%	9s*
BGREAT	963,153	96.3%	96.3%	46s*
BrownieAligner	999,993	100%	100%	27s*
R2G <sup>2</sup> Flow	1,000,000	100%	99.4%	2m20s (~50s demux.)

Compared to the other tools, R2G<sup>2</sup>Flow run times are noticeably longer for the

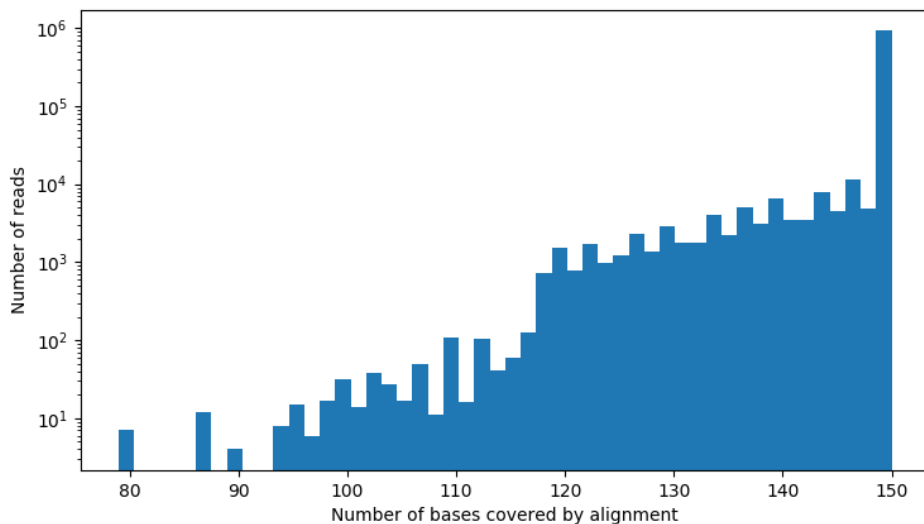


Figure 3.1: Read alignment coverage histogram of 1 million simulated HTS reads from E. Coli genome. More than 96% of the reads were aligned to graph with at least 140 base pairs.

same number of reads alignments. However, our stream alignment pipeline carries out other important tasks for demultiplexing the raw BCL data into reads. Considering that computationally intensive read pre-processing and cluster coordination tasks are not being performed by the traditional alignment tools, additional overhead is expected. There are at least two places in the pipeline where the alignment events are transferred between different processes over the network, namely in between BCL broker, Flink data stream and the graph service. Additional network calls may also occur in case the streaming alignment flow is distributed to more than one hosts. Although we have included all services on the same physical server during the experiment, data serialization and deserialization events still occur during the remote procedure calls (RPC).

### 3.2.2 *K*-mer Cache Improvements

We have also evaluated the performance gain achieved by using thread-local *k*-mer cache optimization as described in section 2.2.2.1. The LRU cache is shown

to be effective in decreasing the impact of RPC I/O overhead for the remainder of the  $k$ -mer lookup process by at least 25%. However batching the consecutive  $k$ -mer lookup calls in to single network call makes a greater impact for reducing I/O overhead.

Table 3.2: Comparison of  $k$ -mer lookup performance with respect to the  $k$ -mer LRU cache and batch query settings using 50K reads and 8 CPU cores

		<b>Run Time</b>
With batch queries	LRU-cache enabled	34s
	LRU-cache disabled	45s
Without batch queries	LRU-cache enabled	1m49s
	LRU-cache disabled	2m42s

### 3.3 Experiments with Human Genome

We compared the alignment rate, base coverage and speed of R2G<sup>2</sup>Flow to other tools using a single chromosome from human genome as reference (Table 3.3). The GRCh38 assembly published by the Genome Reference Consortium [47]. We used ART to generate about 9.5 million reads to achieve 28× read coverage in the simulated data. Although the alignment performance of R2G<sup>2</sup>Flow is similar to the state-of-the-art methods, the alignment took much longer in comparison to the other aligners. We measured that about 40% of the CPU time accounts for the seed extension phase of the pipeline. In comparison to the bacterial genome experiment, time spent during  $k$ -mer lookup is also slightly increased due to larger number of unique  $k$ -mers found in the human genome. To evaluate the run time impact of using larger reference graphs, performed another experiment using 1 million reads only (Table 3.4). In comparison to the bacterial genome, mapping the same number of reads in parallel was nearly took 2 times as long where as the reference genome is 10 times longer. In human genome, increasing the number of parallel reads aligned from 1M to 10M resulted in less than 1.2 times increase in the runtime.

It is worth noting that the experimental setup for the other tools does not cover the demultiplexing time performed after the sequencing is completed. Although

Table 3.3: Results for mapping 9.5M simulated short-reads on GRCh38.Chr19 assembly compared to the other tools on single host with 112 cores. Remarkd run times shown without added time cost of demultiplexing, which might take up to an hour in practical applications.

Tool Name	#Aligned	Aligned%	Coverage%	Run Time
BWA	9,513,560	100%	100%	1m49s*
BGREAT	5,532,418	58.1%	58.1%	52s*
BrownieAligner	9,498,837	99.6%	99.6%	8m2s*
R2G <sup>2</sup> Flow	9,513,560	100%	99.5%	1h6m (~18m demux.)

Table 3.4: Results for mapping 1M simulated short-reads on GRCh38.Chr19 on single host with 112 cores.

	Mapping Coverage%	Run Time (real)	RunTime (user)
R2G <sup>2</sup> Flow	99.4%	5m34s	6h5m

R2G<sup>2</sup>Flow performs slower in these experiments than the baseline, our pipeline has the demultiplexer built-in (Figure 3.2). As a result, the end to end time between preparing the genetic material and obtaining read mapping data would be shorter.

In terms of mapping quality, R2G<sup>2</sup>Flow performed similar to the state-of-the-art de Bruijn aligners. However BGREAT2 was not able to map 41.9% of the reads with the default alignment parameters. Both R2G<sup>2</sup>Flow and the BrownieAligner were able to map with the accuracy above 99.5%. We calculated the number of bases that are covered per each read aligned by R2G<sup>2</sup>Flow (Figure 3.3). 97% of the reads were mapped with at least 140 base pairs. Small number of reads mapped to a larger length in the reference than the read read itself due to gaps. This problem can be resolved with better alignment scoring heuristics such as used by the BrownieAligner.

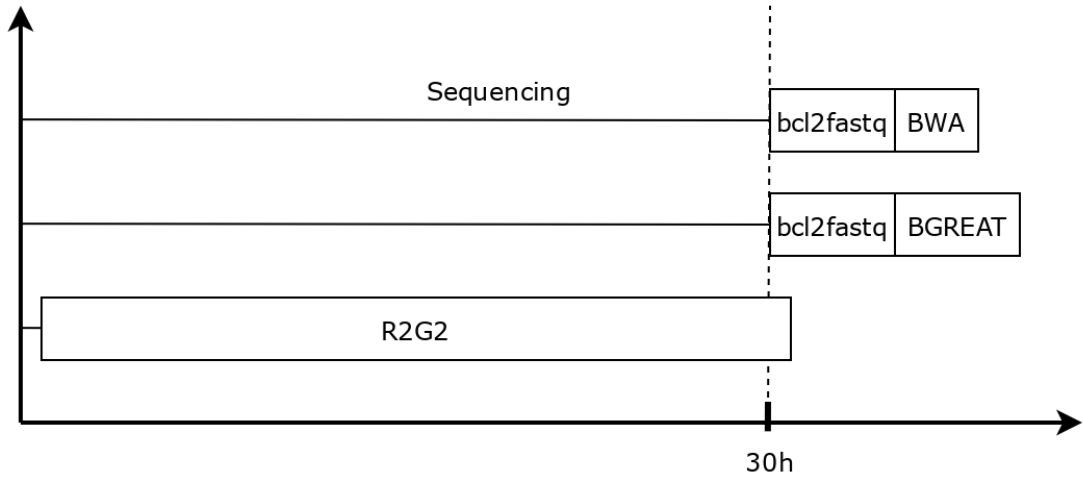


Figure 3.2: Approximate timing diagram demonstrates the difference of R2G<sup>2</sup>Flow run time characteristics in comparison to the other aligners. The traditional read aligners can only be used after the sequencing run is completed. For Illumina sequencing platform this process may take 30 hours.

### 3.3.1 Simulated Variants

In the practical applications, the source of the sequenced data carries a number of variations from the reference genome. In order to simulate the effect of the variations, we have generated a new genome from the actual reference genome using VarSim [48]. Then, we have simulated reads using ART [45] from the new genome we have obtained and repeated the experiment we have performed for single chromosome (Table 3.3.1). Due to the larger divergence between the sample and reference, a small drop in the alignment rate occurred as expected. R2G<sup>2</sup>Flow has performed similarly to the baseline graph aligners in terms of the alignment rate.

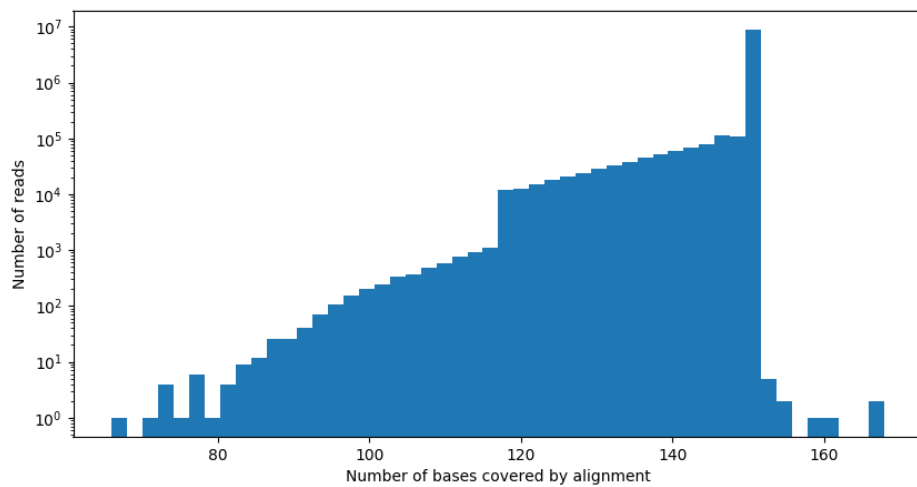


Figure 3.3: Read alignment coverage histogram of 9.5 million simulated HTS reads from E. Coli genome. There were 15 reads that were mapped to region greater than the read length ( $> 150\text{bp}$ ).

Table 3.5: Results for mapping 9.4M simulated short-reads from the sample generated by VarSim from the reference GRCh38.Chr19 assembly. All alignments are performed on single host with 112 cores. Remarkd run times shown without added time cost of demultiplexing, which might take up to an hour in practical applications.

<b>Tool Name</b>	<b>#Aligned</b>	<b>Aligned%</b>	<b>Coverage%</b>	<b>Run Time</b>
BWA	9,447,932	100%	100%	2m19s*
BGREAT	5,308,131	56.5%	56.6%	52s*
BrownieAligner	9,186,650	97.2%	97.2%	13m30s*
R2G <sup>2</sup> Flow	9,447,070	99.99%	98.6%	1h19m



### 3.4 Reference Location Resolution

As a post-processing step, R2G<sup>2</sup>Flow enriches the de Bruijn graph alignment results with the possible locations that the read could be mapped from the original reference strings. We indexed the unitigs from the de Bruijn graph we used in section 3.2 with the help of BCALM [43] and BWA [39] programs. Then we performed a post-processing step to calculate offsets in the original reference genome from the unitig alignment offsets and converted the results to SAM format. We compared the genomic alignment positions from R2G<sup>2</sup>Flow to the results of a BWA run (Figure 3.4). Although the alignment coordinates mostly overlap with the baseline, there are a few locations where the reads are aligned different locations than the baseline. The most of the discordant alignments occur at the locations of the genome where there are repetitive  $k$ -mer patterns.

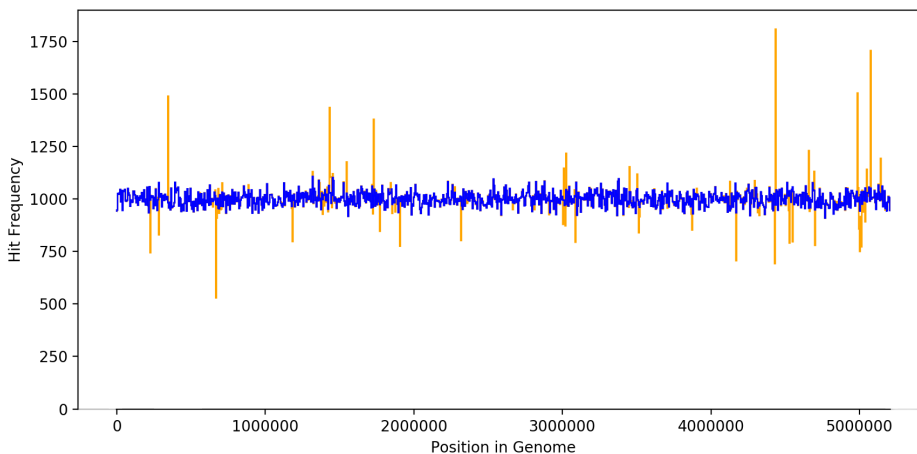


Figure 3.4: Accuracy of alignment visualized by number of hits per coordinates of *E. Coli* genome. Blue line represents the baseline alignment, orange lines corresponds to the actual mapped coordinates of R2G<sup>2</sup>Flow.

Since the alignments are performed against the de Bruijn graph, the output produced by R2G<sup>2</sup>Flow may favor a specific copy of a repeat in the genome when calculating locations in the original references. This is because we currently take the first alternative to output during the reference location resolution. The mapping accuracy may be improved if multiple outputs are taken into accounts by

evaluating the set of reference positions across the mapped unitigs and extracting possible chains based on the continuity of the mapped positions within the reference. Further analysis of the mapping path we produce may also be used for extracting novel variation events in case the interspecies genome references were used.

# Chapter 4

## Discussion and Future Work

We demonstrated that our novel aligner R2G<sup>2</sup>Flow can perform genome graph alignments on a distributed streaming infrastructure. Our assessment shows that it is feasible to adapt greedy de Bruijn graph alignment methods for streaming read data. Evidently, sufficiently high throughput can be achieved with the stream processing adaptations of the state-of-the-art graph aligners that are relying on seed-and-extend paradigm.

### 4.1 Mission Critical Streaming Alignments

There is a growing need for streaming analysis for genome sequencing workflows in mission critical tasks. Streamlined analysis pipelines are being used to reduce time needed for accurate diagnosis such as determining antibiotic resistance by genome sequencing [49]. The overall diagnosis time can be very important for responding to life-threatening situations and major outbreaks such as COVID-19. Streaming aligners can be used for monitoring the mutations and determining the particular strains of pathogens affecting many patients at the same time. Although the overall computational requirements of the R2G<sup>2</sup>Flow is greater than the non-streaming alternatives, overall run time is still projected to take

much less than a full run of the underlying sequencing session. R2G<sup>2</sup>Flow can be deployed to as many hosts as required to meet the throughput requirements to provide real-time response.

## 4.2 Distributed Execution Environment

The distributed nature of R2G<sup>2</sup>Flow processing architecture makes it possible to distribute the work load into multiple hosts with ease. As both streaming flow and the graph alignment service of R2G<sup>2</sup>Flow supports distributed execution, the system can be scaled up horizontally. With horizontal scaling, the total computational capacity of the system can exceed the capacity of what can be achieved using a single host. We also believe that operational costs of sequence alignment work flows can be reduced if many but smaller instances are allocated to the cluster as opposed to allocating single large instance. This feature may be especially useful for the institutions where the cloud computing is on demand.

The streaming flow of R2G<sup>2</sup>Flow is using the Scala API of Apache Flink [36] and runs on Java Virtual Machine. K-mer index and graph search algorithms are implemented natively in C++ using de Bruijn graph API provided by GATB [40]. The two parts of the system communicate through asynchronous calls through a platform agnostic gRPC [50] protocol. We have taken measures to reduce the impact of communication overhead between different parts of the system so that the overall runtime is not affected significantly by the separation of processes. The design of the  $k$ -mer cache takes advantage of CPU affinity of the individual processes. As the  $k$ -mer lookup tasks are routed to the worker threads based on hash value of the seed, the  $k$ -mer index is being implicitly redistributed across the cluster based on the access pattern and availability of resources. The access locality is positively impacted from this process. We believe this may benefit significantly for the cluster configurations where large number of CPU cores are available.

## 4.3 Future Work

### 4.3.1 Improved de Bruijn Graph Alignment Heuristics

The alignment accuracy profile of R2G<sup>2</sup>Flow can be improved by incorporating more advanced alignment heuristics as exemplified by BrownieAligner [35] and McCortex [34]. The BrownieAligner introduces major alignment heuristics such as applying an edit distance-based cost function and a Markov model derived from the original reference strings. These improvements help the greedy aligner to choose better paths at the branching nodes of the de Bruijn graph. Markov model may be especially beneficial for the regions of the genome where there are a large number of repetitions since the context from the read before the branching node is important for path decision. Utilization of a Markov model helps this decision by using a statistical method. In McCortex [34], the long range information from the respective reference strings are stored in the de Bruijn graph itself. We believe our alignment procedure can be extended in the future by using similar heuristics for the seed-and-extend algorithm.

### 4.3.2 Integration of Other Types of Graph Aligners

There are a number of graph aligners [23, 28] that are not based on de Bruijn graphs. Instead the derivations of string graphs and sequence graphs are used for representing the set of genomes. We believe that with small set of modifications to our streaming pipeline, these alignment algorithms can be used for continuous analysis of the HTS data in the future.

### 4.3.3 Post-alignment Analysis Flows

Continuous processing of the sequencing data for read mapping is an essential step for building the systems perform higher level analyses such as structural variant

discovery in an automated manner. Thus, R2G<sup>2</sup>Flow paves the way to perform streamlined genome analyses on the streaming HTS data by having partial alignment events exposed as soon as a the related sequencing cycle is completed. In a future application, the partial alignment streams can be used for extracting preliminary results with increasingly better confidence as the sequencing cycles progress forward. Thus, it may be proven that some preliminary results from the sequencing experiments can be made available even before the final sequencing cycles.

Streaming read aligners allow performing additional analysis even before the sequencing cycles are fully completed. Since R2G<sup>2</sup>Flow incrementally outputs the partial read alignments, preliminary results can be obtained towards the later cycles with incrementally higher fidelity. When combined with other bioinformatics pipelines such as variant discovery and detecting mutations based on the read depth statistics, a streaming alignment workflow expose crucial findings hours before it would normally take with non-streaming pipelines.

# Bibliography

- [1] Y. Cao, S. Fanning, S. Proos, K. Jordan, and S. Srikumar, “A review on the applications of next generation sequencing technologies as applied to food-related microbiome studies,” vol. 8, p. 1829.
- [2] The 1000 Genomes Project Consortium, “A global reference for human genetic variation,” *Nature*, vol. 526, pp. 68–74, Sep 2015.
- [3] S. Goodwin, J. D. McPherson, and W. R. McCombie, “Coming of age: ten years of next-generation sequencing technologies,” vol. 17, no. 6, pp. 333–351.
- [4] F. Sanger, S. Nicklen, and A. R. Coulson, “DNA sequencing with chain-terminating inhibitors,” vol. 74, no. 12, pp. 5463–5467.
- [5] D. R. Bentley, S. Balasubramanian, H. P. Swerdlow, G. P. Smith, J. Milton, C. G. Brown, K. P. Hall, D. J. Evers, C. L. Barnes, H. R. Bignell, J. M. Boutell, J. Bryant, R. J. Carter, R. K. Cheetham, A. J. Cox, D. J. Ellis, M. R. Flatbush, N. A. Gormley, S. J. Humphray, L. J. Irving, M. S. Karbelashvili, S. M. Kirk, H. Li, X. Liu, K. S. Maisinger, L. J. Murray, B. Obradovic, T. Ost, M. L. Parkinson, M. R. Pratt, I. M. J. Rasolonjatovo, M. T. Reed, R. Rigatti, C. Rodighiero, M. T. Ross, A. Sabot, S. V. Sankar, A. Scally, G. P. Schroth, M. E. Smith, V. P. Smith, A. Spiridou, P. E. Torrance, S. S. Tzonev, E. H. Vermaas, K. Walter, X. Wu, L. Zhang, M. D. Alam, C. Anastasi, I. C. Aniebo, D. M. D. Bailey, I. R. Bancarz, S. Banerjee, S. G. Barbour, P. A. Baybayan, V. A. Benoit, K. F. Benson, C. Bevis, P. J. Black, A. Boodhun, J. S. Brennan, J. A. Bridgham, R. C. Brown, A. A. Brown, D. H. Buermann, A. A. Bundu, J. C. Burrows, N. P. Carter,

N. Castillo, M. C. E. Catenazzi, S. Chang, R. N. Cooley, N. R. Crake, O. O. Dada, K. D. Diakoumakos, B. Dominguez-Fernandez, D. J. Earnshaw, U. C. Egbujor, D. W. Elmore, S. S. Etchin, M. R. Ewan, M. Fedurco, L. J. Fraser, K. V. F. Fajardo, W. S. Furey, D. George, K. J. Gietzen, C. P. Goddard, G. S. Golda, P. A. Granieri, D. E. Green, D. L. Gustafson, N. F. Hansen, K. Harnish, C. D. Haudenschild, N. I. Heyer, M. M. Hims, J. T. Ho, A. M. Horgan, K. Hoschler, S. Hurwitz, D. V. Ivanov, M. Q. Johnson, T. James, T. A. H. Jones, G.-D. Kang, T. H. Kerelska, A. D. Kersey, I. Khrebtukova, A. P. Kindwall, Z. Kingsbury, P. I. Kokko-Gonzales, A. Kumar, M. A. Laurent, C. T. Lawley, S. E. Lee, X. Lee, A. K. Liao, J. A. Loch, M. Lok, S. Luo, R. M. Mammen, J. W. Martin, P. G. McCauley, P. McNitt, P. Mehta, K. W. Moon, J. W. Mullens, T. Newington, Z. Ning, B. L. Ng, S. M. Novo, M. J. O'Neill, M. A. Osborne, A. Osnowski, O. Ostadan, L. L. Paraschos, L. Pickering, A. C. Pike, A. C. Pike, D. C. Pinkard, D. P. Pliskin, J. Podhasky, V. J. Quijano, C. Raczy, V. H. Rae, S. R. Rawlings, A. C. Rodriguez, P. M. Roe, J. Rogers, M. C. R. Bacigalupo, N. Romanov, A. Romieu, R. K. Roth, N. J. Rourke, S. T. Ruediger, E. Rusman, R. M. Sanches-Kuiper, M. R. Schenker, J. M. Seoane, R. J. Shaw, M. K. Shiver, S. W. Short, N. L. Sizto, J. P. Sluis, M. A. Smith, J. E. S. Sohna, E. J. Spence, K. Stevens, N. Sutton, L. Szajkowski, C. L. Tregidgo, G. Turcatti, S. Vandevondele, Y. Verhovsky, S. M. Virk, S. Wakelin, G. C. Walcott, J. Wang, G. J. Worsley, J. Yan, L. Yau, M. Zuerlein, J. Rogers, J. C. Mullikin, M. E. Hurles, N. J. McCooke, J. S. West, F. L. Oaks, P. L. Lundberg, D. Klenerman, R. Durbin, and A. J. Smith, "Accurate whole human genome sequencing using reversible terminator chemistry," *Nature*, vol. 456, pp. 53–59, Nov 2008.

- [6] Y. Mostovoy, M. Levy-Sakin, J. Lam, E. T. Lam, A. R. Hastie, P. Marks, J. Lee, C. Chu, C. Lin, Ž. Džakula, H. Cao, S. A. Schlebusch, K. Giorda, M. Schnall-Levin, J. D. Wall, and P.-Y. Kwok, "A hybrid approach for de novo human genome sequence assembly and phasing," *Nature Methods*, vol. 13, no. 7, pp. 587–590, 2016.
- [7] "Initial sequencing and analysis of the human genome," vol. 409, no. 6822, pp. 860–921.



- [8] L. Gannett, “The human genome project,” in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, winter 2019 ed., 2019.
- [9] D. Y. C. Brandt, V. R. C. Aguiar, B. D. Bitarello, K. Nunes, J. Goudet, and D. Meyer, “Mapping Bias Overestimates Reference Allele Frequencies at the HLA Genes in the 1000 Genomes Project Phase I Data,” *G3 (Bethesda, Md.)*, vol. 5, pp. 931–941, Mar. 2015.
- [10] R. M. Sherman, J. Forman, V. Antonescu, D. Puiu, M. Daya, N. Rafaels, M. P. Boorgula, S. Chavan, C. Vergara, V. E. Ortega, A. M. Levin, C. Eng, M. Yazdanbakhsh, J. G. Wilson, J. Marrugo, L. A. Lange, L. K. Williams, H. Watson, L. B. Ware, C. O. Olopade, O. Olopade, R. R. Oliveira, C. Ober, D. L. Nicolae, D. A. Meyers, A. Mayorga, J. Knight-Madden, T. Hartert, N. N. Hansel, M. G. Foreman, J. G. Ford, M. U. Faruque, G. M. Dunston, L. Caraballo, E. G. Burchard, E. R. Bleecker, M. I. Araujo, E. F. Herrera-Paz, M. Campbell, C. Foster, M. A. Taub, T. H. Beaty, I. Ruczinski, R. A. Mathias, K. C. Barnes, and S. L. Salzberg, “Assembly of a pan-genome from deep sequencing of 910 humans of African descent,” *Nature Genetics*, vol. 51, pp. 30–35, Jan. 2019.
- [11] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals,” vol. 10, p. 707.
- [12] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” vol. 48, no. 3, pp. 443–453.
- [13] T. F. Smith and M. S. Waterman, “Identification of common molecular subsequences,” vol. 147, no. 1, pp. 195–197.
- [14] “Sequencing coverage for NGS experiments.” <https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/coverage.html>. Accessed 2020-01-04.
- [15] S. Canzar and S. L. Salzberg, “Short read mapping: An algorithmic tour,” vol. 105, no. 3, pp. 436–458.

- [16] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” vol. 215, no. 3, pp. 403–410.
- [17] H. Xin, D. Lee, F. Hormozdiari, S. Yedkar, O. Mutlu, and C. Alkan, “Accelerating read mapping with FastHASH.”
- [18] D. Belazzougui, P. Boldi, G. Ottaviano, R. Venturini, and S. Vigna, “Cache-oblivious peeling of random hypergraphs,”
- [19] M. Burrows and D. J. Wheeler, “A block sorting lossless data compression algorithm,” tech. rep., Digital Equipment Corporation, 1994.
- [20] P. Ferragina and G. Manzini, “Opportunistic data structures with applications,” vol. 2000, pp. 390–398.
- [21] J. Sirén, N. Välimäki, and V. Mäkinen, “Indexing graphs for path queries with applications in genome research,” vol. 11, no. 2, pp. 375–388.
- [22] J. Sirén, “Indexing variation graphs,” in *2017 Proceedings of the Nineteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pp. 13–27, Society for Industrial and Applied Mathematics.
- [23] E. Garrison, J. Sirén, A. M. Novak, G. Hickey, J. M. Eizenga, E. T. Dawson, W. Jones, S. Garg, C. Markello, M. F. Lin, B. Paten, and R. Durbin, “Variation graph toolkit improves read mapping by representing genetic variation in the reference,” vol. 36, no. 9, pp. 875–879.
- [24] A. Limasset, B. Cazaux, E. Rivals, and P. Peterlongo, “Read mapping on de bruijn graphs,” vol. 17, no. 1, p. 237.
- [25] N. G. DE BRUIJN, “A combinatorial problem,” *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, vol. 49, pp. 758–764, 1946.
- [26] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, “Reducing storage requirements for biological sequence comparison,” vol. 20, no. 18, pp. 3363–3369.
- [27] H. Li, “Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences,” vol. 32, no. 14, pp. 2103–2110.

- [28] M. Rautiainen and T. Marschall, “GraphAligner: Rapid and versatile sequence-to-graph alignment,” p. 810812.
- [29] T. Marschall, M. Marz, T. Abeel, L. Dijkstra, B. E. Dutilh, A. Ghaffari, P. Kersey, W. P. Kloosterman, V. Mäkinen, A. M. Novak, B. Paten, D. Porubsky, E. Rivals, C. Alkan, J. A. Baaijens, P. I. W. De Bakker, V. Boeva, R. J. P. Bonnal, F. Chiaromonte, R. Chikhi, F. D. Ciccarelli, R. Cijvat, E. Datema, C. M. Van Duijn, E. E. Eichler, C. Ernst, E. Eskin, E. Garrison, M. El-Kebir, G. W. Klau, J. O. Korb, E.-W. Lammeijer, B. Langmead, M. Martin, P. Medvedev, J. C. Mu, P. Neerincx, K. Ouwens, P. Peterlongo, N. Pisanti, S. Rahmann, B. Raphael, K. Reinert, D. de Ridder, J. de Ridder, M. Schlesner, O. Schulz-Trieglaff, A. D. Sanders, S. Sheikhzadeh, C. Schneider, S. Smit, D. Valenzuela, J. Wang, L. Wessels, Y. Zhang, V. Guryev, F. Vandin, K. Ye, and A. Schönhuth, “Computational pan-genomics: status, promises and challenges,” *Briefings in Bioinformatics*, vol. 19, pp. 118–135, Jan. 2018.
- [30] M. Rautiainen, V. Mäkinen, and T. Marschall, “Bit-parallel sequence-to-graph alignment,” p. 323063.
- [31] G. Navarro, “Improved approximate pattern matching on hypertext,” vol. 237, no. 1, pp. 455–463.
- [32] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean, “De novo assembly and genotyping of variants using colored de bruijn graphs,” vol. 44, no. 2, pp. 226–232.
- [33] B. Liu, H. Guo, M. Brudno, and Y. Wang, “deBGA: read alignment with de bruijn graph-based seed and extension,” vol. 32, no. 21, pp. 3224–3232.
- [34] I. Turner, K. V. Garimella, Z. Iqbal, and G. McVean, “Integrating long-range connectivity information into de Bruijn graphs,” *Bioinformatics*, vol. 34, pp. 2556–2565, Aug. 2018.
- [35] M. Heydari, G. Miclotte, Y. Van de Peer, and J. Fostier, “BrownieAligner: accurate alignment of illumina sequencing data to de bruijn graphs,” vol. 19, no. 1, p. 311.

- [36] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, “Apache flink: Stream and batch processing in a single engine,” *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.
- [37] M. S. Lindner, B. Strauch, J. M. Schulze, S. H. Tausch, P. W. Dabrowski, A. Nitsche, and B. Y. Renard, “HiLive: real-time mapping of illumina reads while sequencing,” vol. 33, no. 6, pp. 917–319.
- [38] F. Versaci, L. Pireddu, and G. Zanetti, “Scalable genomics: From raw data to aligned reads on Apache YARN,” in *2016 IEEE International Conference on Big Data (Big Data)*, pp. 1232–1241, Dec. 2016. ISSN: null.
- [39] H. Li and R. Durbin, “Fast and accurate short read alignment with burrows–wheeler transform,” vol. 25, no. 14, pp. 1754–1760.
- [40] E. Drezen, G. Rizk, R. Chikhi, C. Deltel, C. Lemaitre, P. Peterlongo, and D. Lavenier, “GATB: Genome assembly & analysis tool box,” vol. 30, no. 20, pp. 2959–2961.
- [41] A. Limasset, G. Rizk, R. Chikhi, and P. Peterlongo, “Fast and Scalable Minimal Perfect Hashing for Massive Key Sets,” in *16th International Symposium on Experimental Algorithms (SEA 2017)*, vol. 75 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 25:1–25:16, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.
- [42] S. Gog, T. Beller, A. Moffat, and M. Petri, “From theory to practice: Plug and play with succinct data structures,” in *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pp. 326–337, 2014.
- [43] R. Chikhi, A. Limasset, and P. Medvedev, “Compacting de Bruijn graphs from sequencing data quickly and in low memory,” *Bioinformatics*, vol. 32, pp. i201–i208, 06 2016.
- [44] J. Clausen, *Branch and Bound Algorithms – Principles And Examples*. 1999.
- [45] W. Huang, L. Li, J. R. Myers, and G. T. Marth, “ART: a next-generation sequencing read simulator,” *Bioinformatics*, vol. 28, pp. 593–594, Feb. 2012.

- [46] M. Touchon, C. Hoede, O. Tenaillon, V. Barbe, S. Baeriswyl, P. Bidet, E. Bingen, S. Bonacorsi, C. Bouchier, O. Bouvet, A. Calteau, H. Chiapello, O. Clermont, S. Cruveiller, A. Danchin, M. Diard, C. Dossat, M. E. Karoui, E. Frapy, L. Garry, J. M. Ghigo, A. M. Gilles, J. Johnson, C. Le Bouguéneq, M. Lescat, S. Mangenot, V. Martinez-Jéhanne, I. Matic, X. Nassif, S. Oztas, M. A. Petit, C. Pichon, Z. Rouy, C. S. Ruf, D. Schneider, J. Tourret, B. Vacherie, D. Vallenet, C. Médigue, E. P. C. Rocha, and E. Denamur, “Organised genome dynamics in the escherichia coli species results in highly diverse adaptive paths,” *PLOS Genetics*, vol. 5, pp. 1–25, 01 2009.
- [47] “Grch38.p13 - genome - assembly - ncbi.” [https://www.ncbi.nlm.nih.gov/assembly/GCF\\_000001405.39](https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.39). Accessed 2020-01-17.
- [48] J. C. Mu, M. Mohiyuddin, J. Li, N. Bani Asadi, M. B. Gerstein, A. Abyzov, W. H. Wong, and H. Y. K. Lam, “VarSim: a high-fidelity simulation and validation framework for high-throughput genome sequencing with cancer applications,” *Bioinformatics*, vol. 31, pp. 1469–1471, May 2015.
- [49] K. Břinda, A. Callendrello, K. C. Ma, D. R. MacFadden, T. Charalampous, R. S. Lee, L. Cowley, C. B. Wadsworth, Y. H. Grad, G. Kucherov, J. O’Grady, M. Baym, and W. P. Hanage, “Rapid heuristic inference of antibiotic resistance and susceptibility by genomic neighbor typing,” *bioRxiv*, p. 403204, Aug. 2019.
- [50] “A high performance, open-source universal rpc framework.” <https://www.grpc.io/>. Accessed 2020-01-17.

# Appendix A

## Glossary

**BCL:** Base calls cycle file generated by Illumina devices

**DNA:** Deoxyribonucleic Acid, the main regulatory molecule in the cell

**FASTQ:** A text file format to store reads with quality information

**HGP:** Human Genome Project

**HTS:** High Throughput Sequencing

**I/O:** Input/Output

**LRU cache:** A data structure to hold least recently used items

**RPC:** Remote Procedure Call

**SAM:** Sequence Alignment Mapping format

# Appendix B

## Code

Code is available at <https://github.com/BilkentCompGen/r2g2>.