Comparing the Performance of Sequence Alignment Software:

Bandage, SPAligner, GraphAligner

By
Yusreen Shah

A Thesis Submitted to
Saint Mary's University, Halifax, Nova Scotia
in Partial Fulfillment of the Requirements for
the Degree of Honours.

August, 2023, Halifax, Nova Scotia

Approved: Dr. Somayeh Kafaie

Approved: Dr. Yasushi Akiyama

Date: August 28, 2023

Comparing the Performance of Sequence Alignment Software:

Bandage, SPAligner, GraphAligner

by Yusreen Shah

Abstract

In this thesis, we investigate the performance of available methods and tools for sequence alignment in assembly and de Bruijn graphs. Sequence alignment tools are employed to detect antimicrobial (AMR) gene sequences within the assembly graphs. Utilizing precise and efficient tools for identifying these genes enables us to locate their neighboring genes and evidence of horizontal gene transfer (HGT) more accurately. To this end, we have considered three sequence alignment tools namely Bandage, SPAligner and GraphAligner. The tools have similar input and output types. The outputs are analyzed qualitatively and quantitatively using Panda, Numpy and GFA libraries in Python. The paths returned by each pair of tools for each query are compared to measure the similarity between them. Furthermore, the output sequences from each software are compared to the target sequence using a modified version of edit distance. It was seen that Bandage was the most efficient and precise tool, followed by GraphAligner and then SPAligner for the datasets tested.

August 28, 2023

Comparing the performances of Sequence Alignment Software: Bandage, SPAligner and GraphAligner

Saint Mary's University

Yusreen Shah

August 2023

Abstract

In this thesis, we investigate the performance of available methods and tools for sequence alignment in assembly and de Bruijn graphs. Sequence alignment tools are employed to detect antimicrobial (AMR) gene sequences within the assembly graphs. Utilizing precise and efficient tools for identifying these genes enables us to locate their neighboring genes and evidence of horizontal gene transfer (HGT) more accurately. To this end, we have considered three sequence alignment tools namely Bandage, SPAligner and GraphAligner. The tools have similar input and output types. The outputs are analyzed qualitatively and quantitatively using Panda, Numpy and GFA libraries in Python. The paths returned by each pair of tools for each query are compared to measure the similarity between them. Furthermore, the output sequences from each software are compared to the target sequence using a modified version of edit distance. It was seen that Bandage was the most efficient and precise tool, followed by GraphAligner and then SPAligner for the datasets tested.

Acknowledgement

I would like to thank my supervisor Dr. Somayeh Kaffaie for her patience,encouragement and support throughout this paper. I am grateful for my family in Mauritius, community in Halifax, Windsor ON and Waterloo ON for the continuous love and care.

Mischief managed!

Contents

Chapter 1

Introduction

The genome sequencing technology has developed and improved rapidly. As a result, research in microbiome has had some significant breakthroughs making reference genomes more available as well as improving the ability to sequence entire microbial communities using high-throughput sequencing. These technological advancements have introduced more questions such as how the data, in the form of short-read sequences, are managed, processed and analyzed. Given the short length of the reads, meaningful DNA information such as genes cannot be extracted right away. Assembly methods are hence used to construct longer sequences. These methods in turn create large assembly graphs. In order to process the graphs and extract information from them, special software are available for sequence alignment in such graphs. The research presented in this thesis investigates the performance of the software readily available for use in this area.

1.1 Motivation

In this thesis, we apply sequence alignment tools to identify the sequence of antimicrobial (AMR) genes in the assembly graphs. The motivation behind this project is finding the pros and cons of available tools and choosing the best one based on requirements and limitations. Having accurate and efficient tools for identifying genes such as AMR genes can help us to find their neighborhood genes and evidences of horizontal gene transfer (HGT) precisely.

1.2 Objectives

Our main objective is the comparison of the state-of-the-art sequence alignment tools available in terms of efficiency and accuracy.The results from this research can be used to find a replacement for Bandage in software that are currently using it given that the code behind Bandage is no longer being updated. Our research focuses on the application of sequence alignment in large genome assembly and De Bruijn graphs. The tools we are considering for this purpose are Bandage [1], SPAligner [2], and GraphAligner [3]. Although, there exists some other tools for sequence alignment on graphs as well (e.g.Astarix [4] and GraphChainer [5]), the chosen ones are the most popular ones which are available online and have similar type of input graphs for sequence alignment.

1.3 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 provides an overview of the background materials such as de Novo assembly, antimicrobial resistance and the algorithm behind the tools discussed. In Chapter 3, the experiment undertaken to compare the tools as well as a description of the datasets are documented. After describing the methodology, Chapter 4 presents an analysis

of the time and memory consumption for every datasets in each tool, comparison of the output paths for each pair of tools as well as a measurement of similarity of the output sequences. Finally in Chapter 5, we conclude and summarize the contributions presented in this thesis, and discuss several potential extensions to our research.

Chapter 2

Background

2.1 Introduction

Every living organism is made up of cells. A cell consists of 3 components namely the cell membrane, the nucleus and the cytoplasm [6]. The molecule inside the cell contains information on the function and development of the cell. Deoxyribonucleic acid *(DNA)* resides in the nucleus of the cell. During cellular reproduction, DNA allows the cellular information to be passed from one generation to the other. *Genes* are associated to DNA. In fact, genes are composed of specific DNA segments which control the hereditary traits of organisms. On the other hand, the *Genome* is the entire hereditary information of a living thing. It is the total DNA content in a cell [7].

All living things have a unique genome (or genetic code). The genetic code is composed of nucleotide bases (A, T, C and G). If the sequence of the bases in an organism is determined, then we say that its unique DNA pattern has been identified. Sequencing is the process by which the order of bases is determined. Genome Sequencing is conducted through four main steps [8]:

1. DNA Shearing: the DNA is cut into pieces so that they can be read by the sequencing machine.

2. DNA Barcoding: barcodes are added in order to identify from which bacteria is the sheared DNA from.

3. DNA Sequencing: the DNA from multiple bacteria is combined and put in a DNA sequence. The sequencer identifies the bases in each bacterial sequence.

4. Data Analysis: computer analysis tools are used to compare sequences from multiple bacteria.

Sequencing has been used to identify microbial communities in many environments as well as organisms including humans and animals. High throughput sequencing, which is a technology to sequence DNA and RNA in a rapid and cost-effective manner, has paved the way to *metagenomics*. Metagenomics is the analysis of the combined genomes of organisms co-existing in a community. Metagenomics assembly is an important step in such analyses. It is the stitching of short DNA sequences, called **reads** into genes or organisms. Genome assembly algorithms are used to identify the genomes of single organisms.

2.2 De Novo Assembly

De Novo assembly is an algorithm for Genome Assembly. In *de Novo*, genomes are reconstructed without prior knowledge of the source DNA sequence length, layout or composition. Since *de Novo* is NP-Hard, heuristic-based methods have been devised to perform it. The strategies are greedy, overlap-layout-consensus, and de Bruijn graph [9].

De Bruijn graph considers the relationship between substrings of fixed length k (also known as *k-mers*) derived from *reads* (i.e., sequence of base pairs corresponding to all or part of a single DNA). The nodes are $k-1$ prefixes and suffices of k-mers, while the edges are k-mers. Instead of aligning the reads next to each other, it is inferred that an overlap exists if they share the k-mers. The goal is eventually to find an *Eulerian Path* (that is a path visiting each edge once). Error correction affects de Bruijn graphs. The errors are eliminated prior to identifying an Eulerian path using heuristic strategies. SPADES [10] is one such assembler that uses the De Bruijn graph paradigm.

As discussed previously, De Bruijn Graph (DBG) and Overlap-Layout-Consensus-Approach (OLC) are both different paradigms of the de Novo assembly algorithm. When considering the time and memory consumption, OLC is more effective for low-coverage long reads. On the other hand, DBG is more suitable for high-coverage short reads. It is also worth noting that DBG is particularly more suited for large genome assembly.[11]

2.3 Antimicrobial Resistance

One area that has piqued the interest of many bioinformaticians is Antimicrobial Resistance (AMR). The AMR phenomenon can be attributed to the extensive use of antibiotics in the medical as well as in the agricultural sector. AMR is an important health problem globally. In fact, it is predicted that AMR-related deaths will be around 10 million lives annually [12]. Active Horizontal Gene Transfer (HGT) within the gut microbiota, antibiotic resistance genes and the usage of antibiotics in world populations increase the chances that a pathogenic microbes will obtain genetic resistance from commensal microbes inhabiting the human body [13]. It is therefore important to look at how resistance is affected during and after antibiotic intake and the mechanisms of AMR transmission.

HGT is the non-sexual way of transferring genetic information between genomes. HGT enables microbes to share genetic materials with others to get beneficial traits. This allows them to gain better adaptions, as well as to acquire genes from distant species. HGT therefore increases the genetic diversity of the recipients. It is also a key factor in the microbial evolution [14]. We focus on using sequence alignment tools to identify AMR genes in assembly graphs. Accurate and efficient identification of these genes aids in locating their neighboring genes and provides evidence for HGT with precision.

2.4 Basic Local Alignment Search Tool (BLAST)

One of the most popular sequence analysis tools in the public domain is the Basic Local Alignment Search Tool (BLAST) [15]. It is a computer algorithm that can be accessed online at the National Center for Biotechnology Information (NCBI) website. BLAST compares pairs of sequences and searches for regions of local similarity. BLAST is heuristic in nature; compared to the alignment programs (Needleman-Wunsch and Smith-Waterman algorithms), it does not compare each residue against each other [16].

2.4.1 Heuristics of BLAST

The motivation of BLAST is that comparable sequences are almost certain to contain a short high scoring similarity region, *Hit*. A seed is produced from each hit. These seeds are extended by BLAST on both sides.
BLAST uses two parameters: W, length of a hit and T, minimum score of a hit. It follows the following steps:

1. Compile a list of possible words which, if paired with those in Q, the query sequence creates high scoring pairs.

2. The database is then scanned to find exact matches with the list compiled in step 1. This can be done efficiently using hash tables.

3. The hits from 2 are then extended.

4. The significance of the extended hits from 3 is then assessed.

2.5 Description of the Tools

While BLAST is one of the most popular sequence alignment tools, it only works with plain sequences. In recent years, new tools have been introduced that can identify sequences in the assembly graphs. In this thesis, our focus is on such tools, namely Bandage, GraphALigner and SPAligner.

2.5.1 Bandage

Repeated sequences can be challenging during genome assembly. These sequences cause fragmented assemblies which can be challenging as we try to construct the DNA [17]. These distinct structures in the assembly graph can be visualized within Bandage. While Bandage is primarily used to interact and visualize assembly graphs, we are going to focus on the BLAST searches that can be performed within the Bandage GUI [1].

Bandage constructs a BLAST search by building a BLAST database utilizing all of the graph nodes. It then attempts to find a path through the graph which covers the maximal amount of query for each BLAST query. Bandage uses the following pseudo code to find a path for each query, Querypaths.

1. Search for all conceivable path beginnings which are within a satisfactory distance from the query.

2. For each possible beginning, search the paths to each possible ending.

3. Assuming that a path exists from start hit to end hit, the ideal length is calculated. The ideal length is the query length minus parts of the query that are not covered.

4. Find the minimum and maximum length for the path.

5. Discard any paths that are sub-paths of other longer paths

6. Sort the paths from best to worst.[1]

2.5.2 SPAligner

SPAligner (Saint Petersburg Aligner) tool is used to align nucleotide and amino acid sequences, referred to as a query sequence S, against assembly graphs referred to as graph, G. The regions of high nucleotide identity between the query and the graph sequences are identified. The regions are then extended to semi-global alignments. In semi-global alignment, gaps are allowed at the beginning and/or the end of the sequence. If we consider two sequences, s and t, semi-global alignment is used if the sequences are related along the entire length of the region where they meet.

The way SPAligner works is as follows (further highlighted in Figure 2.1):

1. Anchor search: The query and edge label regions with a high degree of similarity are identified.

2. Anchor filtering: Secures the promising anchors, and discard the questionable ones.

3. Anchor chaining: The anchors are given weights. The assigned weight is proportional to the sequence's length. Secures a and b are viable if the

[1]Extracted from https://github.com/rrwick/Bandage/blob/main/blast/blastquery.cpp

negligible distance between them in G is less than or equivalent to their positions in s [2]. A skeleton of the last arrangement is then made by looking for the heaviest chain of viable anchors.

4. Reconstructing the filling paths: The skeleton's successive anchors can be identified by their paths. This reduces the alignment cost to a minimum. It uses the *Edlib*, a library for sequence alignment for a faster rendition.

[2]

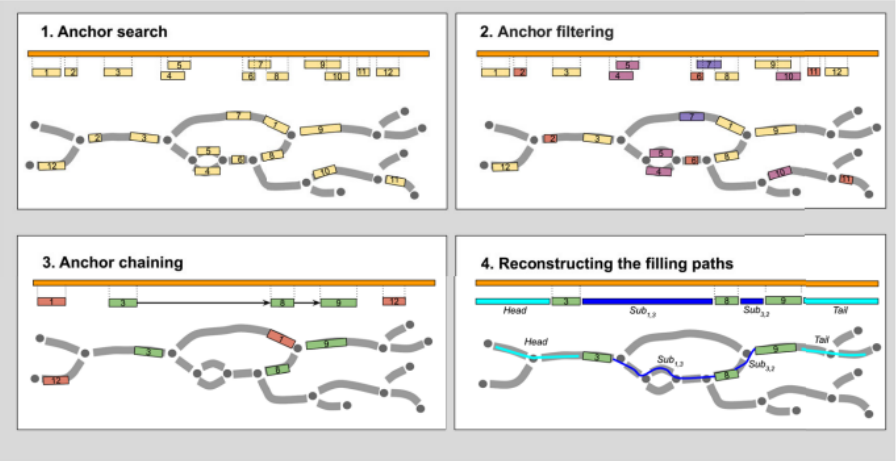

Figure 2.1: *Steps used in SPAligner for sequence alignment [2].*

2.5.3 GraphAligner

GraphAligner is a tool for aligning long reads to sequence graphs. According to [3], it is 13x faster and uses 3x less memory compared to other state-of-the-art tools namely minimap2 and VG toolkit. It can work with different types of graphs.

[2]Extracted from https://github.com/ablab/spades/tree/spaligner-paper

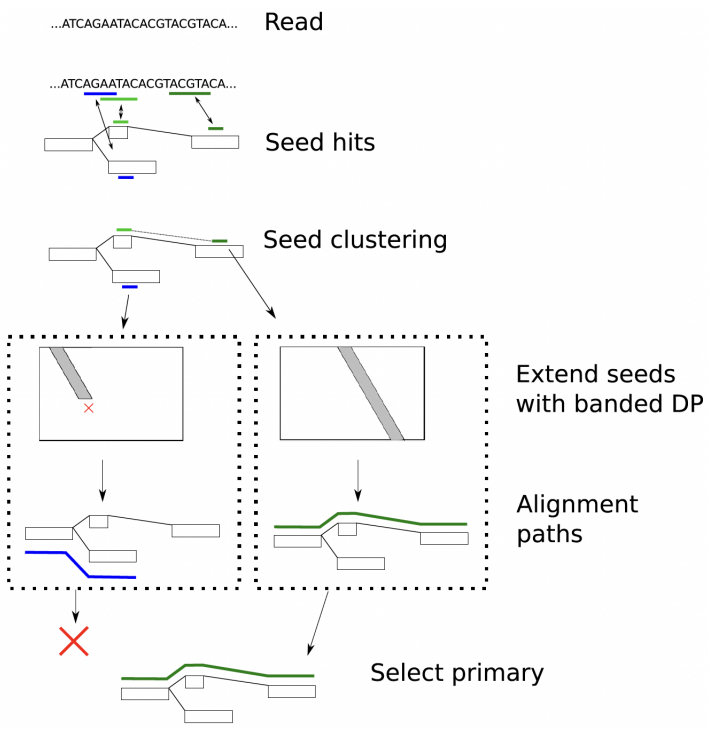

Figure 2.2: *How GraphAligner works [3].*

As shown in Figure 2.2, the algorithm uses a seed-and-extend approach and works as follows:

1. The reads are aligned independently from each other.
2. The sequence of the read is matched to the sequences inside the nodes (blue and green bars)
3. The seed hits are then grouped in locally acyclic parts of the graph and scored.
4. Seed hits are then extended (small dotted boxes) with a banded dynamic programming algorithm, using Viterbi's algorithm to decide when to cut the alignment (red X).

5. Each seed hit can bring about an alignment (blue and green paths).

6. Alignments that cross-over with another longer alignment in the query sequence are classified as secondary.

7. Secondary alignments are usually discarded (red X), yet can be incorporated in the result with a discretionary boundary.

8. The output is then saved to a file either as alignments or corrected reads.

2.5.4 Comparing Bandage, SPALigner and GraphAligner Regarding Sequence Alignment

Table 2.1 summarizes similarities and differences among these tools in terms of main usage and the algorithm behind them. As presented in the table, Bandage provides a way to visualize the graph as well as sequence alignment. While SPAligner and GraphAligner have similar usage in terms of sequence alignment, they use different algorithms under the hood for this purpose.

Table 2.1: This table describes the differences among the 3 tools in terms of usage and algorithms.

Tool	Main uses	Algorithms used to find path
Bandage	Used primarily to visualize and interact with Assembly Graphs.	Uses BLAST to find paths within the assembly graph based on the query sequence.
SPAligner	Used to align long diverged molecular sequences against assembly graphs.	Uses BWA to detect longer anchor alignments. It also uses the Edlib library to calculate the optimal alignment.
GraphAligner	Used to align long reads to sequence graphs.	Uses BWA to detect longer anchor. It also uses the Bit vector alignment extension algorithm.

Chapter 3

Methodology

3.1 Introduction

Since the genetic code has been discovered, scientists have accumulated DNA and protein sequences rapidly. In order to make sense of this vast amount of data, scientists attempt to answer the following questions [18]:

1. Can the functions of newly cloned genes be identified?
2. Can the evolutionary relationship between genes or proteins be estimated just by examining their amino acids?

In order to answer these questions, it is important to identify the relationships between different species. The sequence similarity is used to deduce the function and evolutionary relationships [18]. One of the methods applied to check similarity is sequence alignment on assembly graphs, in which sequence alignment tools identify the path representing a query sequence in the graph. For example in Figure 3.1, the query sequence is identified as a path starting from the 90^{th} nucleotide at node $n2$ and will end at the 80^{th} nucleotide at node $n3$.

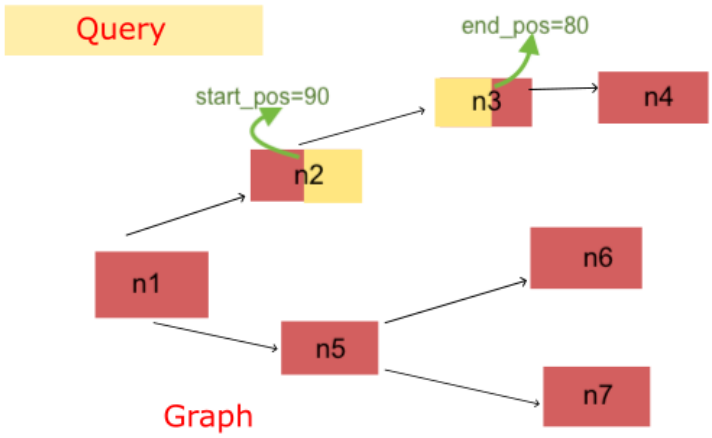

Figure 3.1: *Sequence alignment in the assembly graph for a query sequence.*

3.2 Steps to Compare the Output of the Tools

Here, we analyze the result from the selected tools, Bandage, SPAligner and GraphAligner, available for sequence alignment in assembly graphs in terms of time and memory usage. We also measure the accuracy of the output sequences identified by each tool for different samples as shown by the flowchart in Figure 3.2.

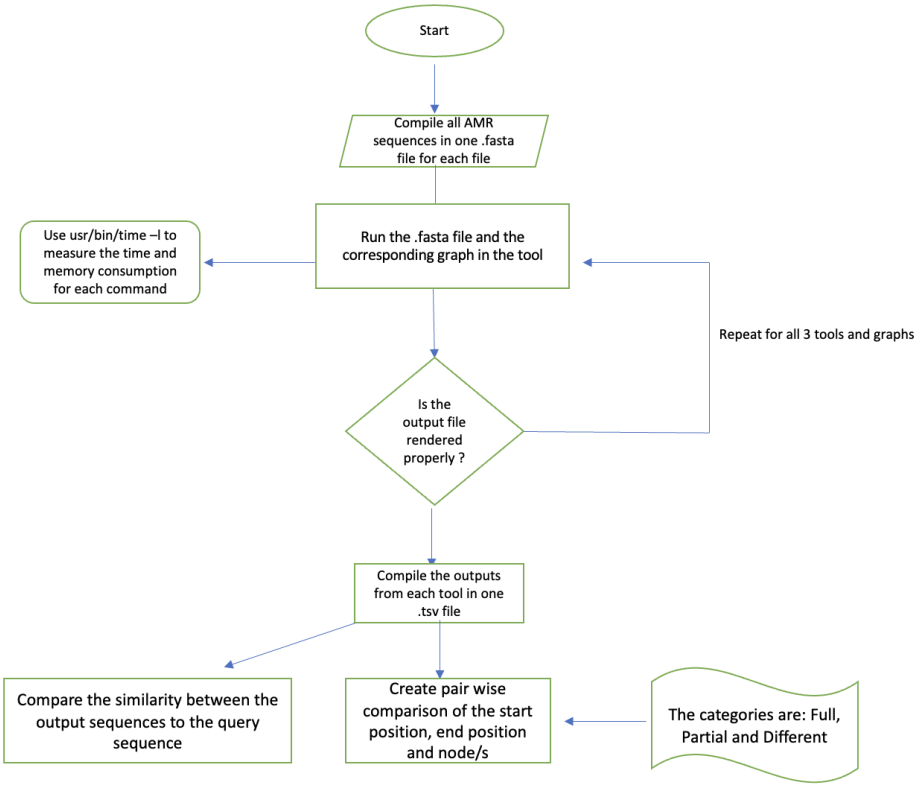

Figure 3.2: *This flowchart shows the steps used to compare the results from the output file for each graph.*

It is important to note that there is another tool called Astarix [4] that was considered initially; however, it was excluded from our research due to significant differences in its input and output file format compared to the other tools.It would not be worth the time to write separate scripts in order to process the input and output.

Upon analyzing the output generated by Bandage, SPAligner, and GraphAligner, we have identified several common elements that are suitable for further analysis such as using the graphs in GFA format. These include the length of the query

sequence, the starting and ending positions on the graph, the specific nodes considered on the graph, and the resulting output sequence. These shared characteristics provide a basis for comparison and evaluation of the performance and effectiveness of these tools in genome assembly.

3.3 Experiments

3.3.1 Datasets

In order to compare Bandage, SPAligner and GraphAligner, the data sets in Table 3.1 were used. We conducted the experiments on assembly graphs with a variety of sizes from small (1_1_1) to medium (CAMI_M_2) to large (CAMI_H_1 and ERR1713331) graphs.

Table 3.1: Description of the datasets.

Name	Description	Number of AMR genes	Number of nodes/edges
1_1_1	Simulated from E.coli SMS-3-5, k pneumoniae MGH and S. aureus Mu50	378	2529 nodes and 2406 edges
CAMI_M_2	CAMI Challenge with 132 genomes	54	396,319 nodes and 101,235 edges
CAMI_H_1	CAMI Challenge with 596 genomes	698	939,234 nodes and 127,706 edges
ERR1713331	Real metagenomic samples from the Global Urban Sewage AMR Monitoring Project	355	3,852,226 nodes and 1,256,367 edges

As explained in Table 3.1, 1_1_1 is a simulated metagenomic dataset generated using one strain from each of Escherichia coli, Staphylococcus aureus, and Klebsiella pneumoniae retrieved from RefSeq. The reads were simulated using ART V2.5.8 on the HiSeq 2500 platform, with read length set to 150bp, insert size at 500bp, and fold coverage at 20 [19]. CAMI_M_2 and CAMI_H_1 are obtained from the Critical Assessment of Metagenome Interpretation (CAMI) study. These datasets offer varying levels of complexity and real-world challenges for metagenome analysis [20]. ERR1713331 is from published metagenome samples derived from urban sewage, which were sequenced using the Illumina HiSeq platform [21].

During this validation process, we thoroughly assessed the performance and capabilities of the three tools using this diverse range of datasets. For each datatset, we used identified AMR sequences in each dataset as the queries and ran the three tools to find them in the corresponding assembly graph. All experiments were run on an iMac, with an Apple M1 chip and 8 GB memory.

Command to Run Bandage

In order to run Bandage, the following command was used:

```
./bandage querypaths <gfa_file> <sequence_file> <output_file> --minmeanid 0.66
```

Given that the default identity threshold(given by *precise-clipping*) for GraphAligner was 0.66 and for the sake of fair comparison, we set *minmeanid* for Bandage to 0.66. It was easier to change the default values in Bandage than in GraphAligner.

Command to Run SPAligner

In order to run SPAligner, the following command was used:

```
./spaligner -d pacbio -g <gfa_file> -s <sequence_file> > <output_file>,
```

where the default values were used.

Command to Run GraphAligner

In order to run GraphAligner, the following command was run:

```
GraphAligner -g <gfa_file> -f <sequence_file> -a <output_file.gfa> -x dbg,
```

where the default values were used.

3.4 Comparing the Output Files

In order to compare the results, we chose two approaches.

1. We used:

   ```
   usr/bin/time -l
   ```

   to measure the time and memory usage for each command.

2. We also compared the accuracy of the results by comparing the paths returned by different tools using the Numpy, GFA and Panda libraries in Python. For example, the start position, end position and the node returned by Bandage and GraphAligner for a given query were compared. Edit distance was used for comparison between the query sequence and the sequences returned by tools.

3.4.1 Analyzing the Output File from Bandage

As shown in Figure 3.3, the output file of Bandage, which is in a *.tsv format, has the following columns and each row represents one output path[1]:

[1]Extracted from https://github.com/rrwick/Bandage/wiki/BLAST-searches

Path	Length (bp)	Query covered by path	Query covered by hits	Mean hit identity	Total hit mis-matches	Total hit gap opens	Length discre-pancy	E-value product
(128274) 2-, 15+, 97+ (565)	615	100.00%	99.02%	96.05%	8	0	0.00%	1.6e-122
(5) 96+, 342+, 341-, 334+ (126)	576	93.65%	93.17%	100.00%	0	0	0.00%	2e-114
(5) 96+, 342+, 95-, 334+ (126)	576	93.65%	93.17%	100.00%	0	0	0.00%	2e-114
(128274) 2-, 15+, 16+, 342+, 341-, 334+ (126)	615	100.00%	98.53%	100.00%	0	0	0.00%	1.6e-113
(128274) 2-, 15+, 16+, 342+, 95-, 334+ (126)	615	100.00%	98.53%	100.00%	0	0	0.00%	1.6e-113
(63946) 17+ (64554)	609	98.53%	98.53%	59.11%	82	1	+0.49%	7e-83
(84690) 9- (85265)	576	92.68%	92.68%	54.69%	85	2	+1.05%	7e-64

Figure 3.3: *A sample of output file from Bandage.*

3.4.2 Analyzing the Output File from SPAligner

Compared to Bandage, the output file from SPAligner can be in *.tsv, *.fasta and *.gpa format. In our experiment, we used *.tsv files to facilitate the comparison. Figure 3.4 shows the output format.

```
name    0       2491  536  1142  2491    44+,24+,22+,1+,38-      909,4,115,1,1142        AAACTTTTATTGTGCATACGGCGATTA
```

Figure 3.4: *A sample of the output file from SPAligner.*

Columns of this file represent the following items[2]:

1. name — sequence name
2. 0 — start position of alignment on sequence
3. 2491 — end position of alignment on sequence
4. 536 — start position of alignment on the first edge of the Path (here on edge with id=44)

[2]Extracted from https://github.com/ablab/spades/tree/spaligner-paper/assembler/src/projects/spaligner

5. 1142 — end position of alignment on the last edge of the Path (here on conjugate edge to edge with id=38)

6. 2491 — sequence length

7. 44+,24+,22+,1+,38- — Path of the alignment

8. 909,4,115,1,1142 — lengths of the alignment on each edge of the Path respectively (44+,24+,22+,1+,38-)

9. AGGTTGTTTTTTGTTTCTTCCGC... — sequence of alignment Path

3.4.3 Analyzing the Output File from GraphAligner

```
read   71    0     71    +     >1>2>4  87    3     73    67    72    60
NM:i:5  AS:f:56.3     dv:f:0.0694444  id:f:0.930556   cg:Z:4=1X2=1I38=1D5=1I5=1X13=
```

Figure 3.5: *A sample of the output file from GraphAligner.*

The output file from GraphAligner is in *.gaf format. Figure 3.5 shows how the output looks like in GAF format[3], and Figure 3.6 provides the description of all columns.

[3]Extracted from https://github.com/maickrau/GraphAligner

Col	Type	Description
1	string	Query sequence name
2	int	Query sequence length
3	int	Query start (0-based; closed)
4	int	Query end (0-based; open)
5	char	Strand relative to the path: "+" or "-"
6	string	Path matching `/([><][^\s><]+(:\d+-\d+)?)+|([^\s><]+)/`
7	int	Path length
8	int	Start position on the path (0-based)
9	int	End position on the path (0-based)
10	int	Number of residue matches
11	int	Alignment block length
12	int	Mapping quality (0-255; 255 for missing)

Figure 3.6: *This figure shows how the GAF format looks like.*

[4]

[4]Extracted from https://github.com/lh3/gfatools/blob/master/doc/rGFA.md#the-graph-alignment-format-gaf

Chapter 4

Results And Findings

In this chapter, we analyzed the results from Bandage, SPAligner and GraphAligner. The analysis is divided into three sections:

1. Analyzing and comparing the path returned by the three tools, including start position, end position and nodes in the path, for each dataset.
2. Comparing time and memory consumption for each dataset in each software.
3. Comparing the similarity between the query sequence and the output sequence from each software for each dataset to check the accuracy of the sequences.

4.1 Comparing the Output Files

For each dataset, the query sequences were combined into one file. The file, along with the corresponding graph, were then used as inputs to run each software. The results were analyzed for each pair of tools by comparing start position(s), end position(s) and node(s) involved in the output path(s). For example, in

Figure 4.1, the Path column represents a path consisted of a single node with ID 605844, where the start position of the query sequence in the node is 130 and its end position is 990.

Query	Path
gb|X13543.1|+|185-1046|ARO:3002533|AAC(3)-IIa	(130) 605844+ (990)

Figure 4.1: *A sample path returned by Bandage representing a query sequence in a graph.*

For these experiments, Numpy, Pandas and GFA libraries in Python were used extensively.

4.1.1 Categories

The categories used to compare the paths returned by each pair of tools are as follows:

1. Full: The start position, end position and node list are identical in paths returned by both tools.
2. Partial: The path(s) returned by the pair of software have some meaningful similarities, i.e for the same query, Bandage and GraphAligner return the same but with different start positions and/or end positions.
3. Different: The node list, start position and end position are different in the two tools.

The motivation behind these categories was to find out which tool between SPAligner and GraphAligner returns the results closest to that of Bandage. This could be helpful in the eventual quest for the replacement of Bandage.

Tables, 4.1-4.4, show the percentage of different categories regarding the pair comparison of the tools for each dataset.

As shown by the Bandage VS GraphAligner row in each table, the paths returned by GraphAligner are closest to those in Bandage as compared to SPAligner.

Table 4.1: The percentage of each category for 1_1_1 in pair comparison of the tools.

Name	Full	Partial	Different
Bandage VS SPAligner	57.9	7.2	34.9
Bandage VS GraphAligner	93.9	2.1	4.0
SPAligner VS GraphAligner	57.7	4.8	37.6

Table 4.2: The percentage of each category for CAMI_M_2 in pair comparison of the tools.

Name	Full	Partial	Different
Bandage VS SPAligner	83.3	0	16.7
Bandage VS GraphAligner	63	20.3	16.7
SPAligner VS GraphAligner	0	85.2	14.8

Table 4.3: The percentage of each category for CAMI_H_1 in pair comparison of the tools.

Name	Full	Partial	Different
Bandage VS SPAligner	0	94.5	5.5
Bandage VS GraphAligner	71.0	24.0	5.0
SPAligner VS GraphAligner	0	95.3	4.7

Table 4.4: The percentage of each category for real sample in pair comparison of the tools.

Name	Full	Partial	Different
Bandage VS SPAligner	0	63.8	36.2
Bandage VS GraphAligner	22.1	23.5	54.4
SPAligner VS GraphAligner	0	63.1	36.9

4.2 Comparing the Time Consumption

As shown by the plots in Figure 4.2, for all of the scenarios Bandage took the shortest time to align all the sequences. With the exception of 1_1_1 dataset, SPAligner took the longest time.

The following plots show the time taken to run the tools for sequence alignment.

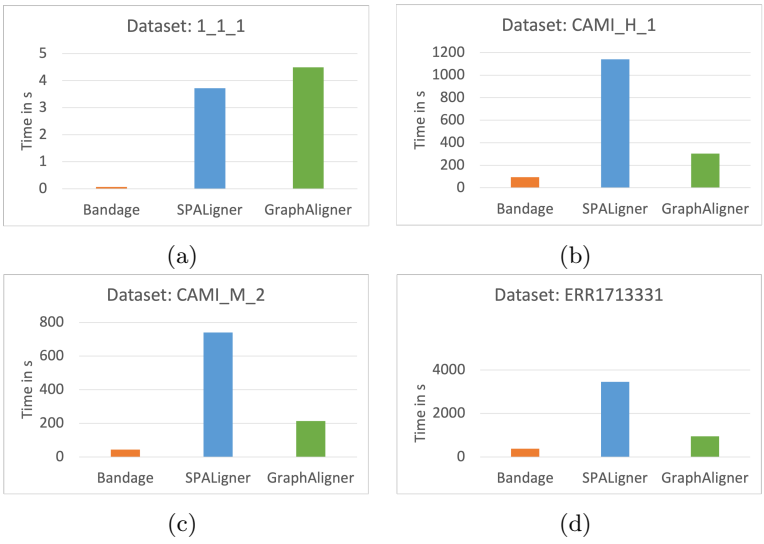

Figure 4.2: Time consumption for each datasets.

4.3 Comparing the Memory Consumption

As shown by the plots in Figure 4.3, for all of the scenarios Bandage used the smallest amount of memory. With the exception of 1_1_1 dataset, GraphAligner consumed the largest amount of memory. The following plots show the memory consumption to run the tools for sequence alignment.

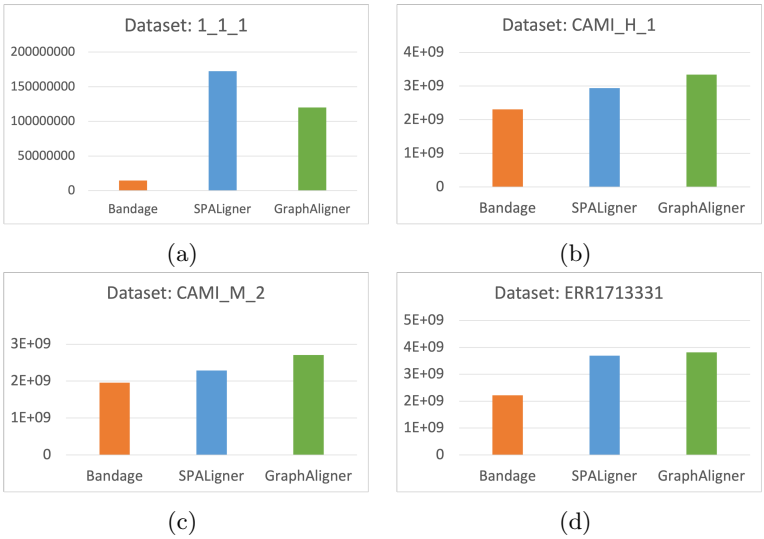

Figure 4.3: Memory consumption of the tools for each datasets.

4.4 Measuring Match_rate

In order to test the quality of the sequences rendered by each software, we measured the similarity between any target query sequence and its corresponding output sequences from the three tools. To do so, we measured the edit distance between the target sequence and the output sequence. The edit distance metric measures the minimum number of character changes that is insertions, deletions, or substitutions required to change one sequence into the other [22]. In an attempt to facilitate the comparison, the edit distance was normalized to make it between 0 and 1. For the new metric, called match_rate, the closer

is the value to 1, the more similar the output sequence is to the target query sequence. The following formula was used to calculate match_rate:

```
match_rate= 1-[edit_Distance/
          max(length(targetSequence),length(output))]
```

For some queries, Bandage returned multiple paths and sequences. For the sake of consistency, we just chose the path with the highest confidence.

4.4.1 Results for 1_1_1 Dataset

For *Bandage*, match_rate for most of the output sequences was closer to 1. *GraphAligner* was the second software that rendered the most accurate sequences. The output sequences from *SPAligner* were the least accurate as shown in Figure 4.4.

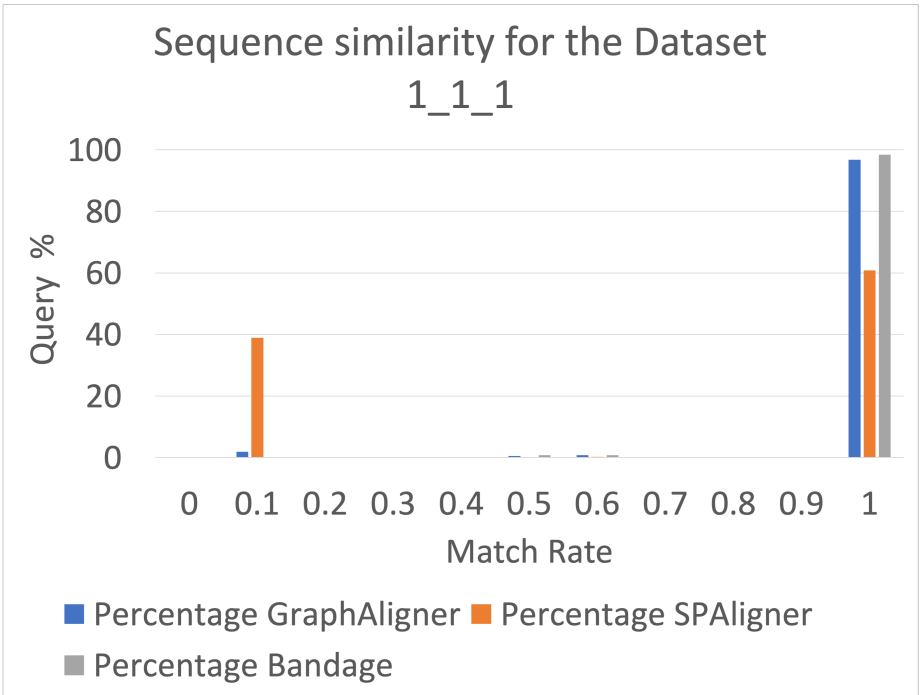

Figure 4.4: *The percentage of sequences with different values of match_rate for each software for 1_1_1 dataset.*

4.4.2 Results for CAMI_M_2 Dataset

As shown in Figure 4.5, *Bandage* created sequences that were identical to the query sequence. Similar to the *real* dataset, *SPAligner* outperfomed *GraphAligner* and some sequences were not found by the GraphAligner.

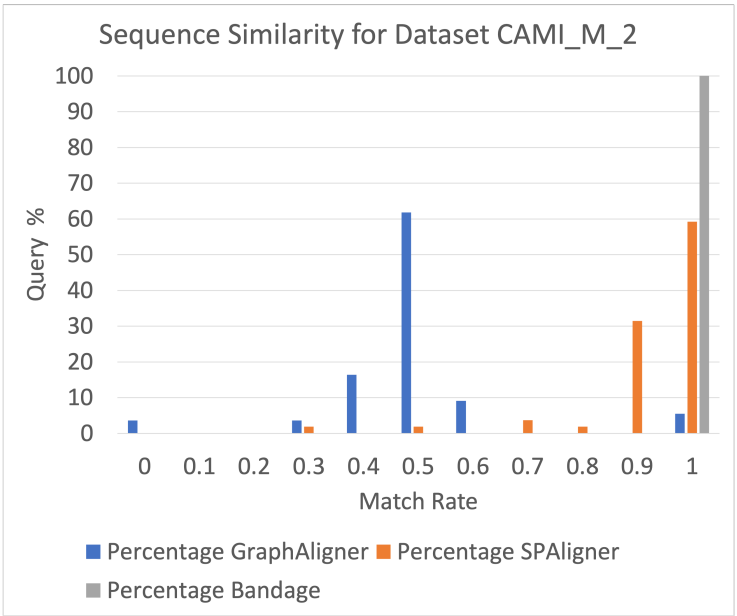

Figure 4.5: *The percentage of sequences with different values of match_rate for each software for CAMI_M_2 datatset.*

4.4.3 Results for CAMI_H_1 Dataset

For *Bandage* and *GraphAligner*, the output sequences hover mostly around 0.5. On the other hand, for *SPAligner* there was a consequent number of sequences that did not match the target sequence at all as shown in Figure 4.6.

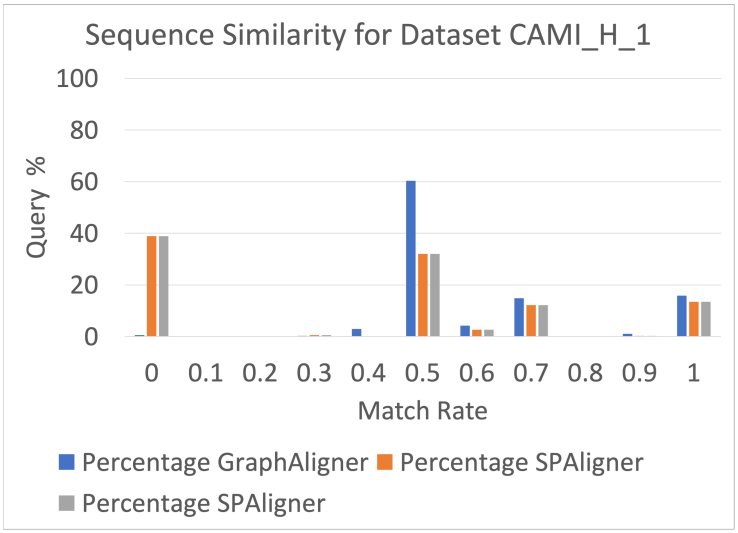

Figure 4.6: *The percentage of sequences with different values of match_rate for each software for CAMI_H_1 dataset.*

4.4.4 Results for the real Dataset

For this dataset, *Bandage* created sequences that were identical to the query sequence. For the graph in particular, *SPAligner* outperfomed *GraphAligner* as shown in Figure 4.7.

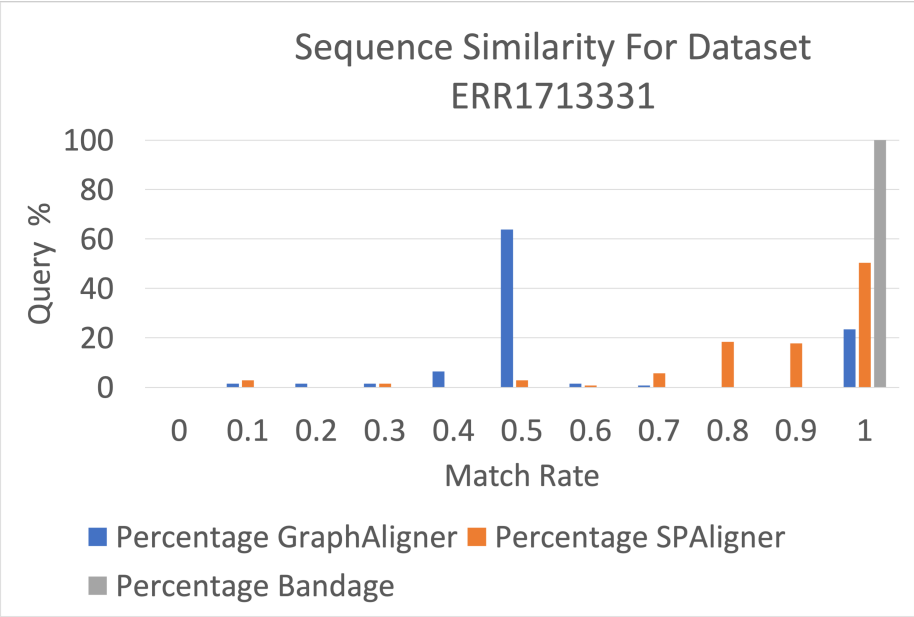

Figure 4.7: *The percentage of sequences with different values of match_rate for each software for real dataset.*

4.5 Average Match Rate

This section shows the average match rate for each dataset by each tool.

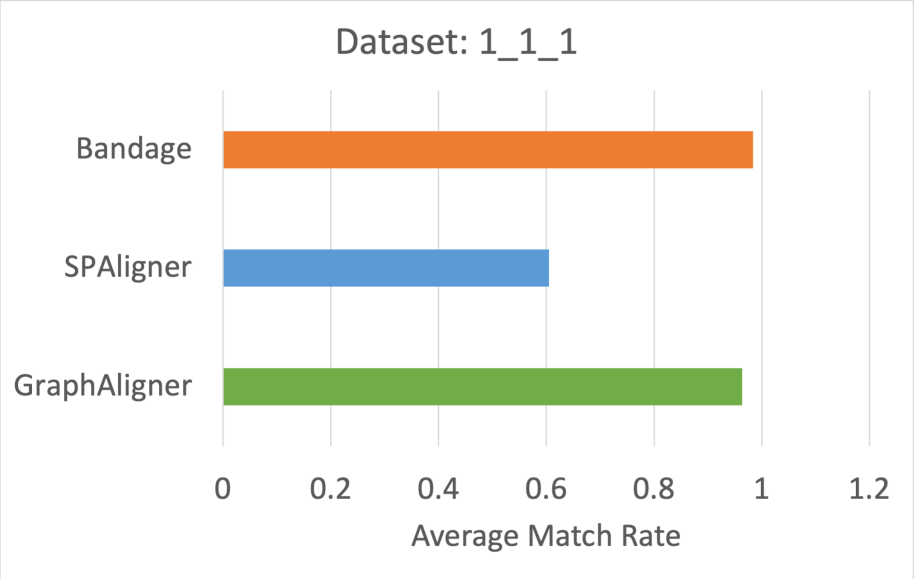

Figure 4.8: *The average match rate for 1_1_1dataset.*

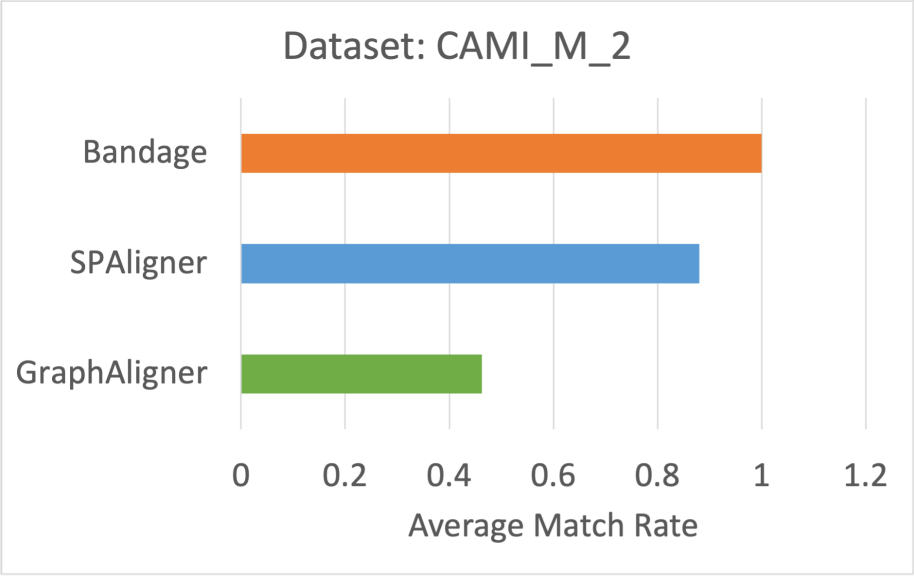

Figure 4.9: *The average match rate for CAMI_M_2 dataset.*

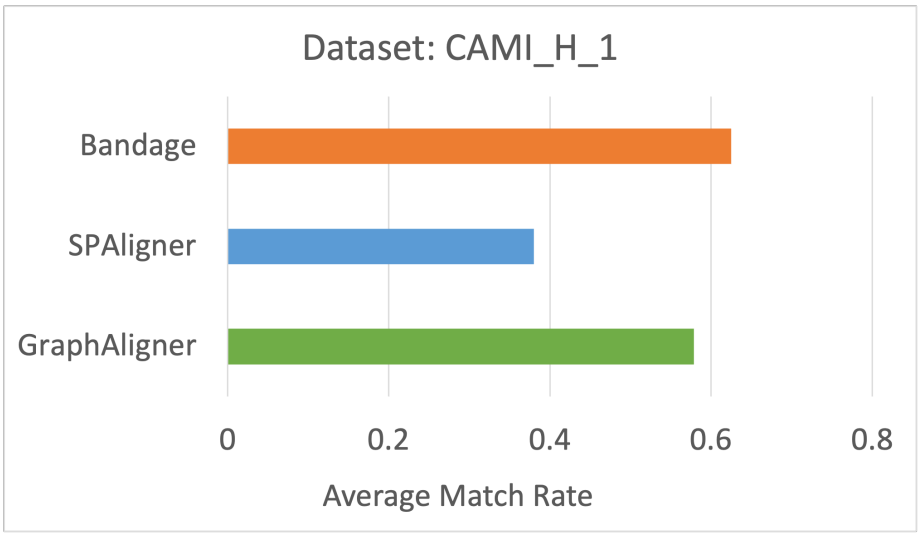

Figure 4.10: *The average match rate for CAMI_H_1 dataset.*

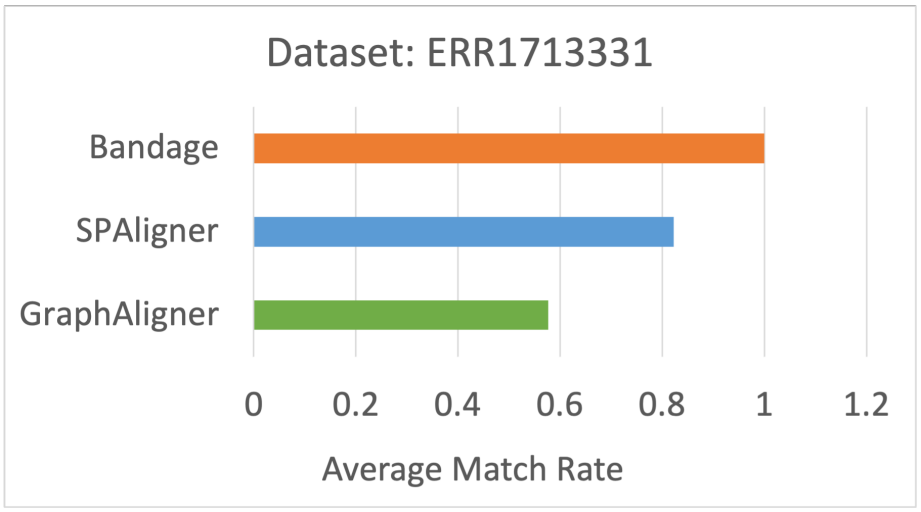

Figure 4.11: *The average match rate for ERR1713331 dataset.*

Chapter 5

Conclusion and Future Work

5.1 Summary and Conclusion

In this thesis, we analyzed and compared the performance of three popular sequence alignment tools, namely Bandage, GraphAligner and SPAligner with respect to accuracy, run-time and memory consumption. We conducted experiments with datasets and graphs of different sizes from small (1_1_1) to medium (CAMI_M_2) to large (CAMI_H_1 and real sample). The results show that Bandage produces the most accurate outputs. Also, as shown by the results, Bandage is the most efficient tool when time and memory are considered. This is followed by GraphAligner and then SPAligner in most cases.

Regarding generated outputs, Bandage and SPAligner construct the sequences as part of the output results. However, GraphAligner does not create the sequences, and additional scripts had to be written to construct the sequences. Table 5.1 summarizes the results.

Table 5.1: Summary of the results

Experiment	Best Performing tool
Run-time and memory usage	Bandage
Accuracy of output sequences	Bandage
Potential replacement for Bandage	GraphAligner

5.2 Challenges

The study observed a prevailing issue concerning the source codes available online, indicating a lack of compatibility with the Apple Silicon M1 chip. The development communities faced delays in providing versions tailored for this specific architecture. Furthermore, it was noted that certain software offerings were deficient in well-structured documentation, impeding users' understanding of their functionalities. Consequently, a substantial number of trial and error attempts were required to gain a comprehensive grasp of the software's capabilities. These findings highlight the importance of addressing compatibility challenges and improving documentation to enhance the user experience and efficiency of software utilization on the Apple Silicon M1 chip.

5.3 Future Works

Given that Bandage seems to be discontinued, both SPAligner and GraphAligner are promising replacements. In our experiments, GraphAligner outperfomed SPAligner in terms of memory and time for most of the datasets. As far as sequence accuracy, GraphAligner outperformed SPAligner for two datasets. We also noted that extra coding had to be done to be able to get the output sequences for GraphAligner. On the other hand, although SPAligner missed some queries, the software does construct the sequences. For future works, it will be beneficial to explore additional tools available that are gaining momentum for

sequence alignment and might provide more comparable outputs to Bandage than GraphAligner and SPAligner.

Bibliography

[1] Ryan R Wick, Mark B Schultz, Justin Zobel, and Kathryn E Holt. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, 31(20):3350–3352, 2015.

[2] Tatiana Dvorkina, Dmitry Antipov, Anton Korobeynikov, and Sergey Nurk. Spaligner: alignment of long diverged molecular sequences to assembly graphs. *BMC Bioinformatics*, 21, 07 2020.

[3] Mikko Rautiainen and Tobias Marschall. Graphaligner: rapid and versatile sequence-to-graph alignment. *Genome biology*, 21(1):253, 2020.

[4] Pesho Ivanov, Benjamin Bichsel, Harun Mustafa, André Kahles, Gunnar Rätsch, and Martin Vechev. Astarix: Fast and optimal sequence-to-graph alignment. In *International Conference on Research in Computational Molecular Biology*, pages 104–119. Springer, 2020.

[5] Jun Ma, Manuel Cáceres, Leena Salmela, Veli Mäkinen, and Alexandru Tomescu. Graphchainer: Co-linear chaining for accurate alignment of long reads to variation graphs, 05 2023.

[6] Yusuf S Khan and Aisha Farhana. Histology, cell. *StatPearls [Internet]*, 2022.

[7] Nigel Chaffey. Alberts, b., johnson, a., lewis, j., raff, m., roberts, k. and walter, p. molecular biology of the cell. 4th edn. *Annals of Botany*, 91(3):401–401, 02 2003.

[8] Mansi Verma, Samarth Kulshrestha, and Ayush Puri. Genome sequencing. *Bioinformatics: Volume I: Data, Sequence Analysis, and Evolution*, pages 3–33, 2017.

[9] Jay S Ghurye, Victoria Cepeda-Espinoza, and Mihai Pop. Focus: microbiome: metagenomic assembly: overview, challenges and applications. *The Yale journal of biology and medicine*, 89(3):353, 2016.

[10] Sergey Nurk, Dmitry Meleshko, Anton Korobeynikov, and Pavel A Pevzner. Metaspades: a new versatile metagenomic assembler. *Genome research*, 27(5):824–834, May 2017.

[11] Zhenyu Li, Yanxiang Chen, Desheng Mu, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Xuesong Hu, Binghang Liu, et al. Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph. *Briefings in functional genomics*, 11(1):25–37, 2012.

[12] J. O'Neill, Review on Antimicrobial Resistance, and England) Wellcome Trust (London. *Tackling Drug-resistant Infections Globally: Final Report and Recommendations*. Review on Antimicrobial Resistance, 2016.

[13] Evgenii I Olekhnovich, Artem T Vasilyev, Vladimir I Ulyantsev, Elena S Kostryukova, and Alexander V Tyakht. MetaCherchant: analyzing genomic context of antibiotic resistance genes in gut microbiota. *Bioinformatics*, 34(3):434–444, 10 2017.

[14] Chen Li, Jiaxing Chen, and Shuai Cheng Li. Understanding horizontal gene transfer network in human gut microbiota. *Gut Pathogens*, 12(1):1–20, 2020.

[15] Scott McGinnis and Thomas L Madden. Blast: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic acids research*, 32(suppl_2):W20–W25, 2004.

[16] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[17] Huilong Du and Chengzhi Liang. Assembly of chromosome-scale contigs by efficiently resolving repetitive sequences with long reads. *Nature Communications*, 10(1):5360, 2019.

[18] Ingrid Lobo. Basic local alignment search tool (blast). *Nature Education*, 1(1), 2008.

[19] Weichun Huang, Leping Li, Jason R Myers, and Gabor T Marth. Art: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012.

[20] Alexander Sczyrba, Peter Hofmann, Peter Belmann, David Koslicki, Stefan Janssen, Johannes Dröge, Ivan Gregor, Stephan Majda, Jessika Fiedler, Eik Dahms, et al. Critical assessment of metagenome interpretation—a benchmark of metagenomics software. *Nature methods*, 14(11):1063–1071, 2017.

[21] Rene S Hendriksen, Patrick Munk, Patrick Njage, Bram Van Bunnik, Luke McNally, Oksana Lukjancenko, Timo Röder, David Nieuwenhuijse, Susanne Karlsmose Pedersen, Jette Kjeldgaard, et al. Global monitoring of

antimicrobial resistance based on metagenomics analyses of urban sewage. *Nature communications*, 10(1):1124, 2019.

[22] Gueddah Hicham. Introduction of the weight edition errors in the levenshtein distance. *arXiv preprint arXiv:1208.4503*, 2012.