Bioinformatics
doi: 10.1093/bioinformatics/xxxxx

Implementation of a computational pipeline for detection of AMR genes using Succinct de Bruijn Graph based seed and extend Algorithm

Succinct de Bruijn Graph for AMR detection.

Suvadeep Chaudhuri[1], Parth Rampal[1], Sarthak Chauhan[1] and Pankhuri Mehndiratta[1]

[1] Computer and Information Science & Engineering Department, University of Florida, Gainesville

Abstract
Antimicrobial Resistant genes are those elements that make bacteria resistant to antibiotics. AMR antibiotic resistance is an important issue that has a heavy impact on public health. There are several studies going on around the world which are faced with storing and analyzing very large data sets of genomes like the Human Genome. The goal of this project is to put together computational pipeline that takes as input a metagenomic dataset, an AMR database and finds reads that align to the AMR database uses them as seeds and uses partially aligned overlapping reads to extend these seeds in hopes of finding flanking regions. We present a succinct de Bruijn Graph implementation which takes the partially overlapping unaligned reads and builds a succinct de Bruijn Graph using BWT and auxiliary data structures to generate contigs which are then used to extend the aligned reads which we use as seeds.These regions could be annotated to identify flanking region which, from a research perspective, would give us interesting information about the genomic context of the AMR gene, such as host bacteria, plasmid or other nearby AMR genes that may be present.
Contact: parth.rampal@ufl.edu

1 Introduction

WHO defines Antimicrobial Resistance (AMR) as what happens when microorganisms, when exposed to antimicrobial drugs change and as a result, the medicines used to fight against them become ineffective, increasing the risk of infection and spread [17]. AMR has become an increasingly serious threat to public health across the globe. Globally, 480000 people develop drug resistant TB each year[17] and 700000 human deaths are related to AMR, with around 23,000 in the US alone[21]. This leads to higher healthcare costs and lower success rates for major surgeries and cancer treatments. This has led to actions plans formulated by the WHO, FAO and even the European Commission[2-8] and more than 80 other countries[18] to which affect policies are aimed at . There is even a $20 million prize competition looking for innovative diagnostic tests to combat AMR bacteria [19]. All of these efforts have predominantly used culture or polymerase chain reaction for characterization of AMR genes. While these efforts are important and provide new insights, they are limited in scope to study AMR genes over multiple organisms. This limits the ability to study the complex pathways, horizontal transfer, interactions and influences from the host environment [7,43].

With advances in High throughput Sequencing technologies, there has been a rapid increase in identification of AMR in metagenomic samples.

However there are very few publically available Bioinformatics tools that identify AMR sequences from data produced by current sequencing technologies. The methods that are available have limitations like limited reference sets and do not provide an end to end pipeline where the user can specify their own metagenomic inputs and AMR reference library and obtain AMR identifications in other organisms.

Two of the most popular alignment methods being currently used are Overlap Layout Consensus and de Bruijn Graph representation.[40] The Overlap Layout Consensus method involves finding overlaps between reads, layouts,, and then creates a final consensus based output genome. This strategy is difficult to use with large datasets, especially the data produced by Next Generation Sequencing (NGS) technologies.

The DBG method has been shown to perform much better with Next Generation Sequencing technologies which produce a much higher number of short reads (70-100bp). But this method poses the problem of extremely large graphs which tend to become very hard to work with especially for very large genomes like the Human Genome. Some work has been done on succinct representations for de Bruijn Graphs [32,39] but there hardly exists any implementation specifically for the study of AMR genes.

Here we present a new AMR pipeline which uses succinct De Bruijn Graphs on reads that are aligned with an AMR database and reads that do not align completely to the database and attempt to extend the alignments in order to gain information about the genomic context of the AMR gene such as the host bacteria. This pipeline will help identify bacteria or plasmids that are carrying these AMR genes.

2 Related Work

With the recent increase in studies based on AMR genes, and increase in the data available for study thanks to NGS technologies and High Throughput sequencing, there has been a rise in the number of AMR determinants available for study. Metagenomic HTS is also being developed for studies related to AMR genes in order to help public health surveillance efforts put in action by many international organizations.[10,43].

Although computational pipelines have been developed for the study of biomes like the Human Microbiome project [41] and even for the development and studies of specific databases, like AMRPlusPlus which was developed to work with the MEGARes database[1], there still exists a lack of tools designed to work with High throughput metagenomic analysis for AMR genes. Whatever methods exist use tools which were never designed for Metagenomic sequencing which leads to suboptimal results[44,45].

Several databases have also been developed for AMR genes like Resfinder[22], ARG-ANNOT[14], the Comprehensive Antibiotic Resistance Database (CARD)[15], the National Center for Biotechnology Information (NCBI) Lahey Clinic beta-lactamase and MEGARes[1] and they have their own respective tools or pipelines developed to work with their respective data.

With the exception of AMR Plus Plus developed to work with MEGARes[1], none of the other tools have been designed to specifically work with metagenomic data, increasing the need to for an end to end AMR analysis pipeline that works with metagenomic assembled data. Further, none of these tools employ the use of deBruijn Graph for assembly which proved to be far more efficient than traditional scaffolding methods for assembly of short reads.

	ALGORITHM	METHOD	SPACE USED
1	ABYSS	DISTRIBUTED HASH TABLE	336G
2	CONWAY & BROMAGE	SPARSE BIT VECTOR	32 Gb
3	MINIA	BLOOM FILTER	5.76 Gb
4	BOSS	BWT	2.56 Gb

De Bruijn Graphs for assembly were first introduced by Pevzner et al in 2001[38]. De Bruijn Graphs in Bioinformatics led to several popular assemblers like SPAdes[31] and IDBA[46] which within themselves showed a marked improvement in performance compared to previous implementations. However traditional de Bruijn graph based assemblers like these rely on constructing several de bruijn Graphs and have a large memory consumption and tend to be extremely slow especially when working with large genomes like the Human Genome.

This led to development of succinct de Bruijn Graph implementations for assembly which use much less memory and can efficiently work with those succinct representations for queries and assembly. Several succinct de Bruijn Graph representations have been proposed over the years, one of the earliest ones was by Simpson et al.[47] for their ABySS assembler which stores the graph as a distributed hash table.

Then the work of Conway and Bromage [36] presented a method that used a sparse bit vector- first introduced by Okanahora and Sadakane[48]- to represent the edges of the graph and used rank and select operations to traverse the graph.

Then comes Minia, by Chikhi and Rizk[35] which uses BLOOM filters to store the edges and traverse the graph by generating all possible outgoing edges at each node and checking their membership using the BLOOM filter.

At the same time, another succinct representation of de Bruijn Graphs was presented by Alexander Bowe, Onodera, Sadakane and Shibuya (also referred to as BOSS) [32] that makes use of the Burrows Wheeler transform [33] that used 2.5 GB to represent reads from the human genome (HapMap: NA18507). This representation is not affected by the size of k in the de Bruijn Graph.

Although there has been more work based on the work of Bowe, et al. like MEGAHIT [29] which implements a parallelizable de Bruijn Graph implementation which exploits the parallelizable architecture of Graphical Processing Units (GPUs). VARI [30] which implements variable ordered de Bruijn Graphs which can handle multiple values of k and Rainbowfish[39], which went a step further on VARI and implemented a coloured de Bruijn graph where each edge is assigned to a color set. Both of these methods use the BOSS method as their core. For the purposes of our implementation, we have used this BOSS implementation of succinct de Bruijn Graphs for our pipeline.

3 Methods

3.1 Description and Implementation of pipeline structure.

We present a computational pipeline that takes a metagenomic FASTA file as input along with an AMR Database reference file also in the FASTA format, generates seeds based on their alignment and outputs a FASTA file containing extensions of these AMR seeds using partially overlapping reads.

The first step uses Burrows Wheeler Aligner [13] an efficient short read aligner to align the Metagenomic reads with the AMR database. This step produces a SAM file [28] as an output.

To process this SAM file, we have written our own JAVA based utility which separates the aligned reads, unaligned reads and finds partially overlapping unaligned reads from the output.

The next step is the core of our pipeline. This step has 2 de-bruijn graph based methods. One is a regular de bruijn graph based seed and extend method and the other is a compact, succinct de Bruijn graph implementation based on the BOSS representation which uses Burrows Wheeler Transform put forward by bowe et. all[33]. This step does a de novo assembly by creating a de bruijn graph by building k-mers from the partially aligned reads and then forming contigs from them.

We then use the reads that aligned with the AMR database as seeds and attempt to extend them using the contigs generated by the de bruijn graph traversal.

The output of this step is a list of metagenomic reads that aligned to the AMR database and have been extended. These extensions should give us an insight on the genomic context of the AMR genes. This can allow us to understand which AMR genes have the potential to be horizontally transferred and is thus "higher risk" than the ones that are not in a given metagenomic sample.

Figure 1 gives the flow of our pipeline.

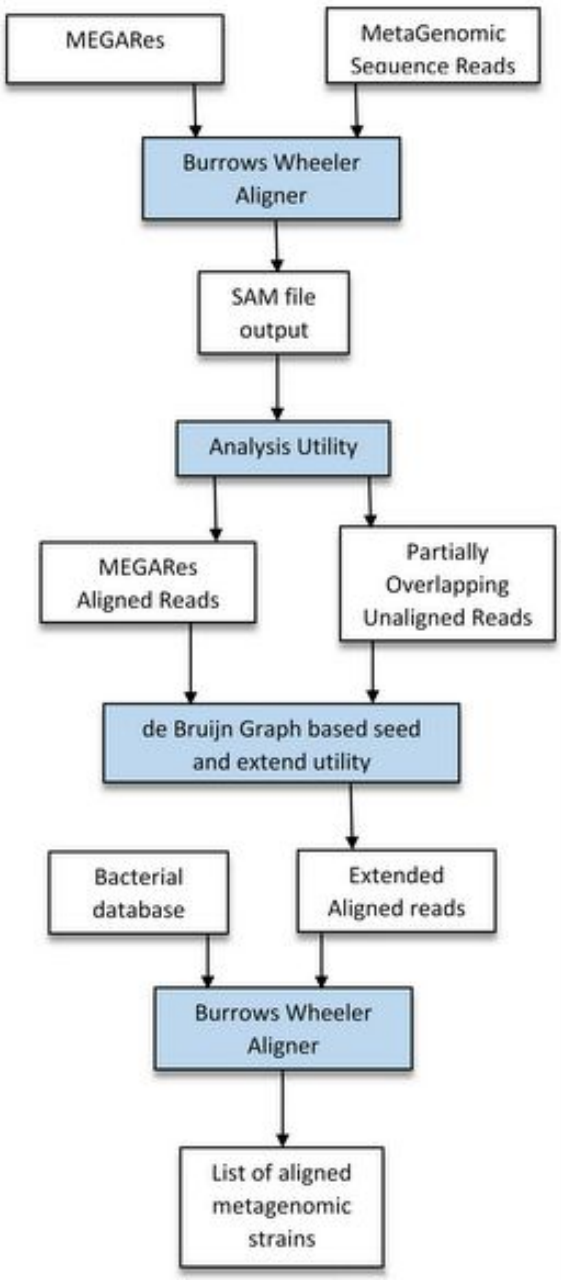

Figure 1: The pipeline representation

3.1.1 Alignment

An alignment can be defined as a tuple (X1, X2, A) where X1 and X2 are two strings and A is a sequence of mismatches like copying, substitution, insertion and deletion operations which transform X2 into X1. Gaps can be Insertions and deletions. These gaps and substitutions are differences. The edit distance of the alignment equals the total number of differences in A. A score can be calculated for an alignment given a scoring system. We say X1 matches X2 if X1 and X2 can be aligned with a positive score, and in this case, we also say (X1, X2) is a match.

A match (X1, X2) is said to be contained in (X1*, X2*) on the first sequence if X1 is a substring of X1*. Similarly, the contained relationship between alignments (a stronger condition) and between an alignment and a match can be defined. [13]
For the first step of our pipeline, we have adopted the Burrows Wheeler Aligner (BWA) tool to align the Metagenomic data with the AMR Database. BWA is an excellent tool for mapping short reads with a large reference which allows mismatches and gaps in the input.[13] Because of its error tolerant abilities, BWA is a good choice for aligning ILLUMINA data which is prone to errors like gaps and ambiguous bases.

BWA employs algorithms for exact and inexact matching each. It reduces space utilization by using Burrows Wheeler Transform, first introduced in 1994 [33] to represent the reference genome, creates an FM Index and then use a backward search on the index to find exact matches and a bounded traversal to look for inexact matches up to a difference of z mismatches. For our pipeline, we have used the BWA MEM algorithm which works best with 70bp-1Mbp reads and aligns them by seeding with Maximal Exact Matches (MEMs) and then extends them using the affine-gap Smith-Waterman algorithm[13].

3.1.2 SAM File Processing

The BWA alignment step produces an output in the Sequence Alignment Map (SAM) format. This is a text based format for storing biological sequences aligned to a reference sequence first developed by Heng Li[28]. The format consists of a header and an alignment section which has 11 mandatory and a number of other optional fields to describe the data. Although samtools [28] exists for parsing the SAM file, we have designed our own tool to parse the SAM file and provide us with fasta files of aligned, partially overlapping aligned and unaligned reads. As explained in section XX, we only use the first two outputs for the purposes of our pipeline.

The following figure(figure 2) depicts the components of the SAM file line according to the SAM format developed by Heng Li.[28]
We use field 3 from the table below in comparing the mappings that have an associated reference sequence name. We can also use the mapping quality that is field 5. However since we use the BWA MEM algorithm during the alignment phase, we are assured that only exact alignments to the individual records of the reference anti microbial database will be the output alignments in the SAM file report. This is why using the quality score is redundant for this BWA algorithm in our scenario for this computational pipeline. However for more flexible results, we can use the the mapping quality score field to drop the results that do not match the user's criteria, and can be more tailored according to the experimental requirements.

ColField	Type	Brief	Description
1	QNAME	String	Query template NAME
2	FLAG	Int	bitwise FLAG
3	RNAME	String	References sequence NAME
4	POS	Int	1- based leftmost mapping POSition
5	MAPQ	Int	MAPping Quality
6	CIGAR	String	CIGAR String
7	RNEXT	String	Ref. name of the mate/next read
8	PNEXT	Int	Position of the mate/next read
9	TLEN	Int	observed Template LENgth
10	SEQ	String	segment SEQuence
11	QUAL	String	ASCII of Phred-scaled base QUALity+33

Figure 2: The SAM file structure

3.1.2.1 De Bruijn Graph

A De Bruijn Graph(DBG) [49] mathematically, is a directed graph representing overlaps between sequences of symbols. It has m^n vertices consisting of m symbols and all possible length-n sequences of these symbols. This concept was first used to represent reads by Pevzner et al.[38] in 2001 and has since then become a widely used alternative to Overlap-layout-consensus assembly[40]. De Bruijn Graphs have been shown to be better than Overlap Consensus layout (OCL) method as genomes are not completely random sequences. OCL programs mask repetitive and low complexity regions and assemble what remains into contigs and scaffolds. Then a very expensive step to merge these scaffolds is carried out where the assembler needs to constantly guess whether variations are due to repeats or errors which greatly reduces the quality of the contigs that are output. This method further became a problem with the introduction of Next Generation Sequencing technologies like Illumina which produce millions of short reads rapidly and inexpensively.

OCL methods fail to perform well for such data as they scale quadratically over the number of reads. De Bruijn graphs overcome this problem by first breaking the short reads into k-mers and the graph is then constructed from these k-mers with the vertices holding k-1 mers that are the left and right subsequences in the k-mer of length k-1 [38]

3.1.2.2 Our Succinct Implementation

We present a new solution designed to extend the AMR aligned reads that were generated in the alignment step using a succinct de Bruijn Graph implementation based on the Burrows Wheeler Transform method presented by Bowes et. al.[32]

3.1.2.3 Burrows Wheeler Transform

The Burrows Wheeler Transform, also known as the block-sorting lossless data compression algorithm permutes letters in a string in reverse lexicographical order of its prefixes.[33]
The biggest advantage of this compression method is that it is lossless and hence reversible. The storage space required by this method is just the order of *n*.
In case of reads, first we generate k-mers and store them in a hashed table. Each node stores the outgoing vertices that are connected to it.
This will be essential for the discovery of the extension path that needs to be found when we find the edge kmer of an aligned read and try to extend it with de Bruijn graph traversal.

3.1.2.4 XBW Transform

The XBW Transform is a method for compressing and indexing labelled trees. It is an extension of the Burrows Wheeler Transform which manages to convert the tree representation into one of size $2n+n \log\sigma$ bits [51].
Keeping in line with succinct data structures, the size of the representation matches the theoretical lower bound.
However, XBW was designed to store a tree and not a graph therefore it is a good choice to use with the succinct de Bruijn Graph. For the traversal of the tree, auxiliary indices[32] are implemented.

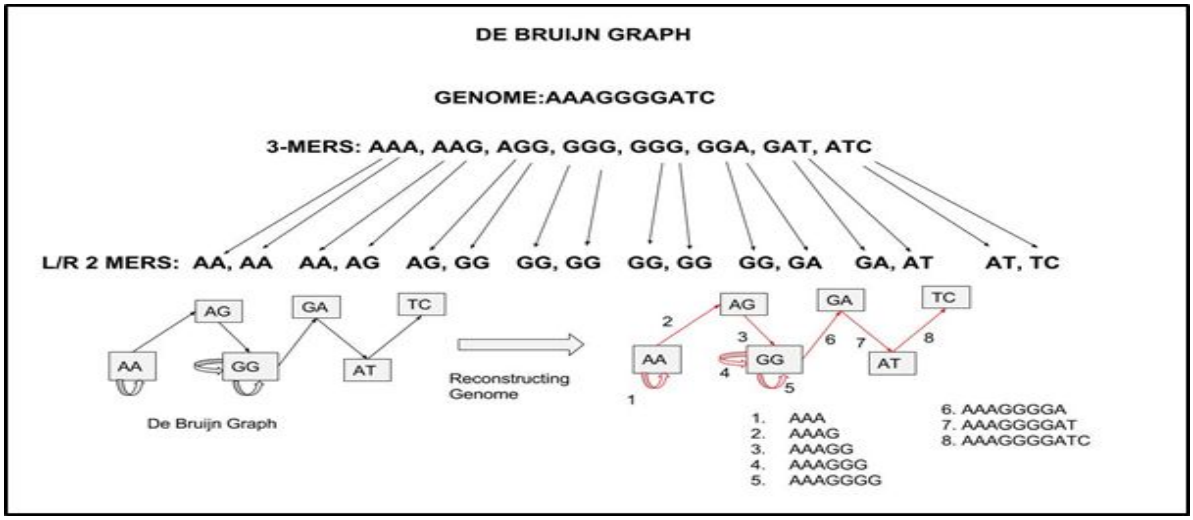

Figure 3: Representation of the de Bruijn graph construction from kmers

3.1.2.5 Rank/Select Data structures.

Let a string of size n be represented by X[0..n-1] where X[0] is the first character, X[1] is the second, X[2] the third and so on till X[n-1]. We implement functions Rank, Select and Access as put forward in the BOSS paper to provide navigational functions in the succinct representation of the de Bruijn graph.

$Rank_c(X,i)$ is defined as a function that returns the number of times the character c occurs in X[0...n-1] till position i.[50]

$Select_c(X, i)$ is defined as a function that returns the position of the ith occurrence of character c in X[0...n-1].[50]

access (X, i) is defined as a function that returns the character at the ith position in X[0...n-1].

All of these functions can be performed in time O(1) with the succinct de Bruijn Graph representation based on Burrows Wheeler Transform.[33]

3.1.2.6 The Succinct de Bruijn Graph Data Structures

The succinct representation of the de Bruijn Graph consists of the following data structures:

An array W = W[1]W[2]...W[m] where each character belongs to the set $A \cup A^-$.
An array Last of length m on on the binary alphabet {0,1}
An array F of length $\sigma = |A|$.

Where A^- represents any set of size |A| such that $A^- \cap A = \varnothing$.

Here, each character W[i] represents the label of an edge in the graph and belongs to the set $A \cup A^-$ If W[i] belongs to A^- then there exists j<i such that W[j] = u(W[i]) and Node[j] and Node[i] have the identical suffix of length k-1.

The array last is defined as Last[i] = 1 iff Node[i] is not the same as Node[i+1] else it is 0. This is used to indicate the range in which a particular node is identical.

The array F stores cumulative frequencies of the last characters of node labels. The array F is represented in $O(\sigma \log m)$ bits.[32,50,51]

An example of these auxiliary data structures is shown in Figure 4 with a sample genomic read and the corresponding array based representations of the structures.

Using all these auxiliary data structure, we can implement operations on the constructed de Bruijn Graph data structures to find paths in the graph that can be used for extensions of the seeds.

The operation forward is the main operation that is used to extend the read starting with a particular vertex. The forwards at every point in the node array can either be calculated or stored in an array based data structure for faster retrieval. This option can be implemented based on the size of the reads that need to be analyzed.

For extending a read for a k-mer, we match the k-mer at a particular index of the node array and continue following the index as per the forward for that index.

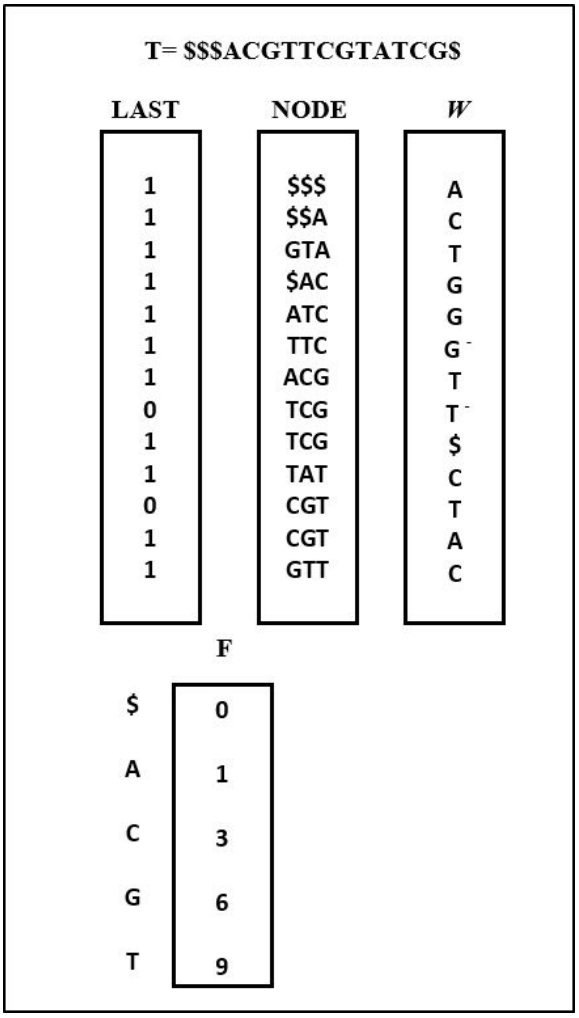

Figure 4: Data structures in the Succinct DBG implementation

3.1.2.7 Operations on sDBP

All standard graph operations are possible on the succinct representation and are defined as follows[32,50,51]:

Outdegree(v): This function returns the number of edges going out of the node v.
Outgoing(v,c): This function returns the node that is reached from v using the edge with label c.
Indegree(v): This function returns the number of edges coming into the node v.
Incoming(v,c): This function returns a node u that precedes v and is connected to v with the label c.
node(v): This function returns the label of the node v.
index(s): This function returns the node v such that Node[v] = s.

Apart from these standard functions, there are two more functions *forward(fwd)* and *backward(bwd)* defined that perform the traversal on the succinct representation of the de Bruijn Graph.

There is a 1 to 1 relationship between indices i with last[i]=1 and the nodes.

Let Node [u] = cs and W[u] = c`. We define Node`[u] = sc`.
For any u such that W[u] belongs to A, Node`[u] is unique and there exists v such that Node[v] = sc`, i.e. there is a 1-1 relationship between indices with last[i] =1 and W[j] which belong to A.
We define *fwd(u) = v* and *bwd(v) = u*. The steps to calculate these two functions are shown in figure 5 and 6 respectively.

Figures 4 and 5 represent the method to calculate the forward and backward at a particular index of the node array. This index is represented by the alphabet 'u' in the following representations.

Fwd(u)

- c=W[u]
- r=$rank_c$(W, u)
- x=F[c] + r – 1
- v=$select_1$(last, x)

Figure 5: Calculation of forward operation

bwd(v)

- c=F^{-1}[x]
- r=x – F[c] + 1
- x=$rank_1$(last, v)
- v=$select_c$ (W, r)

Figure 6: Calculation of backward operation

3.1.3 Seed and Extend

We take partially overlapping reads from the alignment step and then digest them into k-mers and build a succinct de Bruijn Graph. From this graph, we find the Eulerian path to generate contigs that match with the ends of the reads aligned with the AMR database. We then attempt to extend the aligned reads to find the *flanking regions*.

3.1.3.1 The Algorithm:

We first get the last k-mer of the aligned read in the aligned reads file.
We then look for the corresponding k-mer in the de Bruijn graph with a lookup from the hashed list of de Bruijn graph vertices that have been hashed during the construction of the graph.
Next, we look at the outgoing edges from that node and select a particular edge that we must follow.
We continue following other forwarding edges in a similar manner until the desired extension length is achieved.
The length of extension is a parameter that is set as default or can be assigned by the user at the time of running the utility.
Once the extension length is reached, the graph traversal stops and the resulting traversed edges give us a character each upon traversal and finally give us the extended flanking region.
This extended segment is a part of a particular contig that is formed on the regular graph traversal.
We then append the resulting extension to the original aligned read line.
For the extension on the left side of the Aligned read we follow a similar algorithm.
We first reverse the right extended aligned read and follow the algorithm outlined above to extend it in a similar manner.
Once the read has been extended on both sides, it is then written to the output file.
The extension length can be accepted as an argument to the program in the script file.

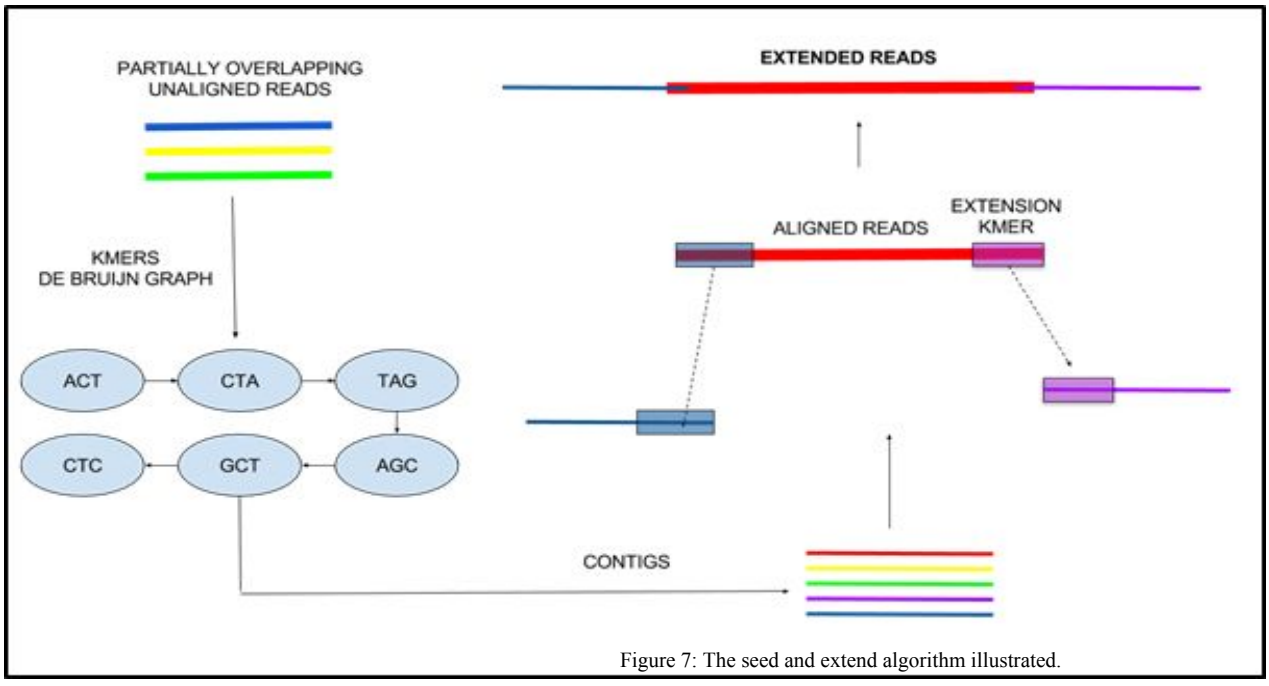

Figure 7: The seed and extend algorithm illustrated.

3.2 Looking up in Other Organisms

Our pipeline outputs the extended AMR reads in a FASTA file. These reads can be further aligned with other bacteria and metagenomic data to gain further knowledge as to where these AMR genes actually lie and what their genomic context is. This can give us important information about AMR genes on bacterial chromosome and help identify if they are potentially mobile or not.

4 Testing and Results

In this section we talk about the results that we achieved with the DBG based implementation and an implementation based on the publically available tools like Velvet and Gapfiller.

To test the results of our implementation, we created two pipelines. The first was our implementation which implements a succinct de Bruijn Graph implementation based on the BOSS methodology to generate contigs from the partially overlapped alignments for the extension step. The second implementation relies on using VELVET [25] a de novo short read assembler that uses traditional de Bruijn Graphs to generate contigs from the partially overlapping reads for the extension step. It uses another de-novo assembler GapFiller[53] for the filling and extension part with different options.
The results from those tests have been presented in Figure 9. For testing the aggregate heap utilization, we used Valgrind-memcheck[52], a DBA tool designed for building heavyweight applications. For checking the time requirements we used the program execution time or the real (wall-clock) time by using the Time command on Linux.

We ran the tests on an Intel i7-3632QM CPU @2.2GhZ with 8GB of RAM, running Ubuntu 16.0.4.

Parameter	Details
Dataset Size	0.88GB
Records in AMR database	3824
BWA	
BWA Alignment	0.88GB
Time elapsed	264sec
SAM File Processing	
Aligned Reads	12982
Partially Overlapping Unaligned Reads	15721
Time elapsed	33sec
Seed and Extend using DBG	
Initial k-mers	1586067
Verices after merging	648815
Reads Extended	12933
Time elapsed	1sec
Memory Used for storage(succinct data structures)	24MB
Time for creation(Data structures)	8sec

Figure 8: Results from initial testing.

Parameter	Details
Dataset Size	3.7GB
Records in AMR database	3824
BWA	
BWA Alignment	3.7GB
Time elapsed	274sec
SAM File Processing	
Aligned Reads	299198
Partially Overlapping Unaligned Reads	37471
Time elapsed	80sec
Velvet	
Initial k-mers	909071
Nodes	526628
Reads Extended	12933
Time elapsed	45sec
Memory Used (aggregate heap)	1.2GB
Gapfiller	
Memory Used (aggregate heap)	11GB
Time elapsed	64sec

Figure 9: Results from alternative tools

5 K-mer Length

Setting the correct K-mer length can be vital for getting the desired results. We tested our implementation on partially overlapping reads generated from our alignment step . The most optimal results keeping in mind meaningful extensions for the short reads that Illumina produces were achieved with a value of 5 for k. We also tested with a kmer value of 11 and achieved good results. The kmer value can be passed as a program argument in the pipeline script to test for different results.

6 Data Used

We decided to test our implementation using MEGARes[1] as the reference AMR database. MEGARes itself has been curated from Resfinder[22], ARG-ANNOT[14] and CARD[15]. CARD contains excessive labelling that makes it unsuitable for high throughput analysis and therefore MEGARes is a better choice for a database to analyze next generation sequencing reads. Further it has been curated for fewer statistical dependencies as it contains no cycles and allows for population level analysis which is one of the primary reasons for the need to find flanking regions of AMR genes.
For the Metagenomic dataset, we used SRR532663, which is the dataset from Illumina whole genome shotgun sequencing of genomic DNA paired-end library 'Pond-151616'. Human Microbiome Project (HMP) Metagenomic WGS Projects, deeper sequencing of the human microbiome. [43]

7 Code Availability and Usage

We have made our code publicly available for use on github and can be reached via our website at https://succintdbgforamr.gitbooks.io/succinctdbgforamr/content/.
The code has been implemented for easy installation and usage. Further details can be found in the README file. The pipeline is easy to use and is wrapped into a complete script file . This pipeline must be used in a Linux based system running java and having gcc.
The data is also available via the website. The data for an initial run must contain an Antimicrobial database and a metagenomic dataset. The AMR database(we use MEGARes) which is available on the github page with the code. The metagenomic dataset is quite large and can be downloaded from the link provided in the README file. The README file can be followed and can be run for a sample run giving the output data.

8 Conclusion

Here we have presented a computational pipeline which uses a BOSS representation based succinct de Bruijn Graph [32] and takes as input reads that aligned to the AMR database, and reads that did not align from metagenomics dataset, and then uses the AMR aligned reads as a "seed" and attempts to "extend" the alignment with partially-overlapping unaligned reads. Lastly, these regions could be annotated to identify flanking region. We have used Burrows Wheeler Aligner [33], a fast short read aligner to prepare the aligned and partially overlapping aligned reads by aligning the given Metagenomic sequences with an AMR database such as MEGARes.

9 Future Work

A lot of progress has been made in the field of succinct de Bruijn Graphs, most notably Variable Order de Bruijn graphs by Chritina Boucher et al.[30] and Rainbowfish by Fatemeh Almodaresi et al.[39]. They have both built upon the BOSS representation put forward by Bowe et al. and improved the space utilization and performance of the assembly step. These works can be further explored and implemented in this pipeline to potentially provide much better performance.

For our AMRUtillity program, we can construct suffix and prefix tries on the aligned and unaligned reads to find the partially overlapping unaligned reads. Synchronized traversal on the two tries, will give us the partially overlapping reads in constant amortized time.

A check for Single nucleotide polymorphisms (SNPs) can be implemented in a future version of the tool and also

10 Acknowledgements

Instructor: Dr. Christina Boucher, Assistant Professor, Department of Computer and Information Science and Engineering, University of Florida.

11 References

1. Lakin S. M., Dean C., Noyes N. R., Dettenwanger A., Ross A. S., Doster E., et al. (2017). MEGARes: an antimicrobial resistance database for high throughput sequencing. Nucleic Acids Res.
2. United Nations News Centre (2016) At Resistance UN, global leaders commit to act on antimicrobial resistance: New York UN, global leaders commit to act on antimicrobial resistance: New York.
3. United States Department of Agriculture, Office of Inspector General
(2016) Resistance USDA's Response to Antibiotic Resistance: Audit
Report 50601-0004-31 : Washington, D.C.
4. European Commission (2016) Evaluation of the Action Plan against the rising threats from antimicrobial resistance.
5. United States Food and Drug Administration (2013) Report National Antimicrobial Resistance Monitoring System—Enteric Bacteria. NARMS Integrated Report: 2012-2013: Silver Spring, MD. http://www.fda.gov/downloads/AnimalVeterinary/SafetyHealth/AntimicrobialResistance/NationalAntimicrobialResistanceMonitoringSystem/UCM453398.pdf
6. EMBL-EBI Metagenomics (2016) Local surveillance of infectious diseases and antimicrobial resistance from sewage: Project (ERP015410).
7. Baquero,F. (2012) Metagenomic epidemiology: a public health need for the control of antimicrobial resistance. Clin. Microbiol. Infect., 18(Suppl. 4), 67–73.
8. Miller,R.R., Montoya,V., Gardy,J.L., Patrick,D.M. and Tang,P.(2013) Metagenomics for pathogen detection in public health.Genome Med., 5, 81.
9. Centers for Disease Control and Prevention (2016) AMD Projects: Combatting Healthcare-associated Infections: Washington, D.C.
10. Interagency Task Force on Antimicrobial Resistance (2012) Update a public health action plan to combat antimicrobial resistance – 2012 Update: Washington, D.C.
11. A review of bioinformatic pipeline frameworks Jeremy Leipzig Briefings in Bioinformatics, Volume 18, Issue 3, 1 May 2017
12. [Klebsiella pneumoniae strain KPNIH39 plasmid pKPN-332, complete sequence. NCBI Reference Sequence: NZ_CP014763.1]
13. Li H. and Durbin R. (2010) Fast and accurate long-read alignment with Burrows-Wheeler Transform. Bioinformatics, Epub. [PMID: 20080505]
14. Gupta,S.K., Padmanabhan,B.R., Diene,S.M., Lopez-Rojas,R., Kempf,M., Landraud,L. and Rolain,J.-M. (2014) ARG-ANNOT, a new bioinformatic tool to discover antibiotic resistance genes in bacterial genomes. Antimicrob. Agents Chemother., 58, 212–220
15. McArthur,A.G., Waglechner,N., Nizam,F., Yan,A., Azad,M.A., Baylay,A.J., Bhullar,K., Canova,M.J., Pascale,G.D., Ejim,L. et al. (2013) The comprehensive antibiotic resistance database. Antimicrob. Agents Chemother., 57, 3348–3357.
16. H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin: The Sequence Alignment/Map format and SAMtools. In: Bioinformatics. 25, 2009, S. 2078
17. Fact sheet on Antimicrobial Resistance, World Health Organisation, November 2017. http://www.who.int/mediacentre/factsheets/fs194/en/

18. World Organisation for Animal Health report, 2015. http://www.oie.int/animal-health-in-the-world/update-on-avian-influenza/2015/
19. Antimicrobial Resistance Diagnostics Challenge, National Institutes of Health, United States Department of Health & Human Services. https://dpcpsi.nih.gov/AMRChallenge
20. National Action Plan for combating antibiotic resistant bacteria, White house Archives. https://obamawhitehouse.archives.gov/sites/default/files/docs/national_action_plan_for_combating_antibotic-resistant_bacteria.pdf
21. Centers for Disease Control and Prevention Antibiotic Resistance Threat report, 2013. https://www.cdc.gov/drugresistance/threat-report-2013/pdf/ar-threats-2013-508.pdf
22. Ea Zankari, Henrik Hasman, Salvatore Cosentino, Martin Vestergaard, Simon Rasmussen, Ole Lund, Frank M. Aarestrup, Mette Voldby Larsen; Identification of acquired antimicrobial resistance genes, Journal of Antimicrobial Chemotherapy, Volume 67, Issue 11, 1 November 2012, Pages 2640–2644
23. Gustavo A. Arango-Argoty, Emily Garner, Amy Pruden, Lenwood S. Heath, PeterVikesland, Liqing Zhang; DeepARG: A deep learning approach for predicting antibiotic resistance genes from metagenomic data
24. The FAO Action Plan on Antimicrobial Resistance 2016-2020; Food and Agriculture Organization of the United Nations; http://www.fao.org/3/a-i5996e.pdf
25. Zerbino DR, Birney E. 2008. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res 18:821– 829. https://doi .org/10.1101/gr.074492.107
26. Martin Hunt, Alison E Mather, Leonor Sánchez-Busó, Andrew J Page, Julian Parkhill, Jacqueline A Keane, Simon R Harris; ARIBA: rapid antimicrobial resistance genotyping directly from sequencing reads doi: https://doi.org/10.1101/118000
27. James J. Davis, Sébastien Boisvert, Thomas Brettin, Ronald W. Kenyon, Chunhong Mao, Robert Olson, Ross Overbeek, John Santerre, Maulik Shukla, Alice R. Wattam, Rebecca Will, Fangfang Xia & Rick Stevens; Antimicrobial Resistance Prediction in PATRIC and RAST; Scientific Reports 6, Article number: 27930 (2016); doi:10.1038/srep27930
28. Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, ; The Sequence Alignment/Map format and SAMtools, Bioinformatics, Volume 25, Issue 16, 15 August 2009, Pages 2078–2079, https://doi.org/10.1093/bioinformatics/btp352
29. Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, Tak-Wah Lam; MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph, Bioinformatics, Volume 31, Issue 10, 15 May 2015, Pages 1674–1676, https://doi.org/10.1093/bioinformatics/btv033
30. Christina Boucher , Alex Bowe , Travis Gagie , Simon J. Puglisi , Kunihiko Sadakane, Variable-Order de Bruijn Graphs, Proceedings of the 2015 Data Compression Conference, p.383-392, April 07-09, 2015 [doi>10.1109/DCC.2015.70]
31. A. Bankevich et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. J. Comput. Biol., 19(5):455–477, 2012.
32. A. Bowe, T. Onodera, K. Sadakane, and T. Shibuya. Succinct de Bruijn graphs.In Proc. WABI, pages 225–235, 2012
33. M. Burrows and D. J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
34. R. Chikhi, A. Limasset, S. Jackman, J. Simpson, and P. Medvedev. On the representation of de bruijn graphs. In Proc. RECOMB, pages 35–55, 2014.
35. R. Chikhi and G. Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. Algorithm. Mol. Biol., 8(22), 2012.
36. T. C. Conway and A. J. Bromage. Succinct data structures for assembling large genomes. Bioinformatics, 27(4):479486, 2011.
37. J. Fischer and V. Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J. Comput., 40(2):465–492, 2011.
38. P. A. Pevzner, H. Tang, and M. S. Waterman. An Eulerian path approach to DNA fragment assembly. Proc. Natl. Acad. Sci., 98(17):9748–9753, 2001.
39. Fatemeh Almodaresi, Prashant Pandey, Rob Patro; Rainbowfish: A Succinct Colored de Bruijn Graph Representation; doi: https://doi.org/10.1101/138016
40. Zhenyu Li, Yanxiang Chen, Desheng Mu, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Xuesong Hu, Binghang Liu, Bicheng Yang, Wei Fan; Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph, Briefings in Functional Genomics, Volume 11, Issue 1, 1 January 2012, Pages 25–37, https://doi.org/10.1093/bfgp/elr035
41. NIH HMP Working Group, Peterson,J, Garges,S., Giovanni,M.,McInnes,P., Wang,L., Schloss,J.A., Bonazzi,V., McEwen,J.E., Wetterstrand,K.A. et al. (2009) The NIH Human Microbiome Project. Genome Res., 19, 2317–2323.
42. EMMES_HMP Illumina whole genome shotgun sequencing of genomic DNA paired-end library 'Pond-151616' containing sample 700108845 from participant 160603188(SRR532663)
43. MacLean,R.C., Hall,A.R., Perron,G.G. and Buckling,A. (2010) The population genetics of antibiotic resistance: integrating molecular mechanisms and treatment contexts. Nat. Rev. Genet., 11, 405–414.
44. Oulas,A., Pavloudi,C., Polymenakou,P., Pavlopoulos,G.A., Papanikolaou,N., Kotoulas,G., Arvanitidis,C. and Iliopoulos,I. (2015) Metagenomics: tools and insights for analyzing next-generation sequencing data derived from biodiversity studies. Bioinform Biol. Insights, 9, 75–88.
45. Ju,F. and Zhang,T. (2015) Experimental design and bioinformatics analysis for the application of metagenomics in environmental sciences and biotechnology. Environ. Sci. Technol., 49, 12628–12640.
46. Peng, Y., et al. (2010) IDBA- A Practical Iterative de Bruijn Graph De Novo Assembler. RECOMB. Lisbon.
47. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJM, Birol İ. ABySS: A parallel assembler for short read sequence data. Genome Research. 2009;19(6):1117-1123. doi:10.1101/gr.089532.108.
48. Okanohara, D., Sadakane, K.: Practical Entropy-Compressed Rank/ Select Dictionary. In: Proc.

of Workshop on Algorithm Engineering and Experiments, ALENEX (2007)
49. de Bruijn; N. G. (1946). "A Combinatorial Problem". Koninklijke Nederlandse Akademie v. Wetenschappen. 49: 758–764.
50. Fariña, Antonio & Ladra, Susana & Pedreira, Oscar & Saavedra Places, Ángeles. (2009). Rank and Select for Succinct Data Structures. Electr. Notes Theor. Comput. Sci.. 236. 131-145. 10.1016/j.entcs.2009.03.019.
51. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and indexing labeled trees, with applications. Journal of the ACM 57(1), 4:1–4:33 (2009)
52. Nicholas Nethercote and Julian Seward: How to Shadow Every Byte of Memory Used by a Program; Proceedings of the Third International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE 2007), San Diego, California, USA, June 2007.
53. Boetzer, Marten and Walter Pirovano. “Toward almost closed genomes with GapFiller.” Genome Biology (2012).