Resistome SNP Calling via Read Colored de Bruijn Graphs

Bahar Alipanahi[1], Martin D. Muggli[2], Musa Jundi[1], Noelle Noyes[3], and Christina Boucher[1]

[1] Department of Computer and Information Science and Engineering,
College of Engineering,
University of Florida, Gainesville, FL.

[2] Department of Computer Science,
College of Natural Sciences
Colorado State University, Fort Collins, CO

[3] College of Veterinary Medicine and Biomedical Sciences,
Colorado State University, Fort Collins, CO

Abstract. The microbiome and resistome, which refers to all the antimicrobial resistant (AMR) genes in pathogenic and non-pathogenic bacteria, are frequently studied using shotgun metagenomics data [13, 50]. Unfortunately, there are few methods capable of identifying single nucleotide polymorphisms (SNPs) in metagenomics data, and to the best of our knowledge, there are no methods that identify SNPs in AMR genes. Nonetheless, the identification of SNPs in AMR genes is an important problem since it allows these genes, which confer resistance to antibiotics, to be "fingerprinted" and tracked across multiple samples or time periods. In this paper, we present LueVari, which allows SNPs to be identified in AMR genes from metagenomes data. LueVari is based on the read colored de Bruijn graph, an extension of the traditional de Bruijn graph that we present and formally define in this paper. We show that read coloring allows regions longer than the k-mer length and shorter than the read length to be identified unambiguously. In addition to this theoretical concept, we present a succinct data structure that allows for large datasets to be analyzed in a reasonable amount of time and space. Our experiments demonstrate LueVari was the only SNP caller that reliably reported sequences that spanned on average 47.5% of the AMR gene. Competing methods (GATK and SAMtools) only reported specific loci and require a reference to do so. This feature, along with the high accuracy of LueVari, allows distinct AMR genes to be detected reliably in a de novo fashion.

1 Introduction

Antimicrobial resistance (AMR) refers to the ability of an organism to stop an antimicrobial from working against it and is described as "an increasingly serious

threat to global public health" since it causes standard treatments (e.g. antibiotics) to be ineffective [40]. This threat prompted the creation of the National Action Plan for Combating Antibiotic-Resistant Bacteria, whose fifth goal is to: "improve international collaboration and capacities for antibiotic-resistance prevention, surveillance, control, and antibiotic research and development" [15, pp. 49]. The plan recognizes that humans, animals, and the environment are sources of AMR and calls for a "one-health approach to disease surveillance" [15, pp. 5]. The *resistome*, which refers to the set of all AMR genes found in pathogenic and non-pathogenic bacteria, defines the potential resistance to known antibiotics. Shotgun metagenomics data has been generated to characterize the resistome in clinical [13, 50] and food production [39, 38, 51] settings. This characterization corresponds to the identification of specific AMR genes, their abundance, and the single nucleotide polymorphisms (SNPs) in the identified genes. Although systems exists that identify AMR genes and their abundance, no such methods exist to indentify SNPs in AMR genes from metagenomics data [13, 21]. The lack of methods is due to both the complexity of the problem and the recentness of this application of metagenomics data [50]. In this paper, we tackle this problem by developing LueVari (Finnish for "read coloring"), which is a scalable reference-free method that is tailored to identify and quantify SNPs in AMR genes.

Although there exist methods to identify SNPs and other variants,the majority of these methods are tailored for eukaryote species—a sentiment expressed by Nijkamp et al. [37] when they state: "Although many tools are available to study variation and its impact in single genomes, there is a lack of algorithms for finding such variation in metagenomes." Current reference-free methods that are specifically designed for metagenomics data require an "assembly" step, which employs an overlap-layout-consensus (OLC) approach [19] or an Eulerian that uses the de Bruijn graph [3, 28, 34, 23, 47, 17]. Thus, SNP detection in metagenomics data is a difficult problem since it exacerbates the weaknesses of these two algorithmic approaches; methods that use an overlap graph are computationally too inefficient to handle large datasets, whereas, de Bruijn graph approaches can be constructed efficiently but are prone to combining read segments inappropriately from different species, which we refer to as *chimeric sequences*. OLC approaches, such as Bambus2 [19], have an advantage over de Bruijn graph approaches that require each read be fragmented into smaller k-length sequences (called k-mers), which further convolutes read-species information. For this reason, many metagenomic variant callers use overlap graphs that relate the entirety of a read to a single region and thus, produce graphs that have minimal complexity. Sequences and variants assembled from such graphs are composed and supported from collections of reads, rather than k-mers—an attribute of overlap graphs known as *read coherence* [33]. This, in theory, reduces the frequency of chimera variants. Unfortunately, computing the overlap between pairs of reads is inefficient and thus, these approaches are unable to handle large datasets [25].

De Bruijn graph approaches have greater capacity to scale to larger datasets, especially in light of existing succinct data structures for constructing and pro-

cessing the de Bruijn graph [44, 6, 7, 52, 4], but lose the sensitivity needed to detect longer variants accurately. Thus, such approaches face having to choose between two undesirable choices: only being able to call short variant sequences constructed from non-branching paths, or risk the introduction of chimeric sequences that arise from spanning branches in the graph. This potential occurrence of chrimeric sequences becomes more frequent in resistome analysis due to sequence homology between AMR genes. AMR genes in the same class share homologous regions that are typically between 60 bp and 150 bp in length [21], which is longer than the typical k-mer value and shorter than the read length. Hence, such regions often correspond to several connected paths in the de Bruijn graph that are difficult to traverse unambiguously, implying that the de Bruijn graph cannot be used to reliably construct the correct sequences corresponding to these genes and their corresponding SNPs. As previously mentioned, these paths are more likely to be read coherent in the overlap graph but the time complexity of OLC methods is too large to be practical for resistome analysis, which encompasses the detection of AMR genes and their SNPs in hundreds or thousands of samples. For example, the USDA is calling for food production facilities to use sequencing to monitor AMR by 2021 [48]—if accomplished, even at a small scale, thousands of samples will be sequenced and analysed to monitor food-borne outbreaks.

Therefore, an approach for identification of SNPs in AMR genes necessitates a method that combines the scalability of de Bruijn graph approaches with the read coherence of OLC approaches. To address this need, we develop LueVari, a read colored de Bruijn graph based SNP caller. It extends the concept of the *colored de Bruijn graph*, which was first introduced by Iqbal et al. [16] for the detection of variants in eukaryote species. Given a set of n samples, the colored de Bruijn graph extends the traditional de Bruijn graph in that each vertex (and edge) in the graph has a set of colors associated with it, where each color corresponds to one of the n samples. In Iqbal et al.'s [16] original application, each sample corresponds to the sequence data of one individual and traversal of the colored de Bruijn graph allows for variation to be detected, along with the individuals containing that variation. In 2017, Muggli et al. [32] created a succinct data structure for constructing and storing the de Bruijn graph. Although these methods that use the colored de Bruijn graph allow for variation to be detected among individuals of a population, it has the same issue of read coherence as the traditional de Bruijn graph and does not assist the identification SNPs in AMR genes. Very briefly, a *read colored de Bruijn graph* annotates each vertex (and edge) by an unique color which corresponds to each individual read (in one or more samples), allowing for read coherence to be preserved among paths longer than the k-mer size (typically, ≤ 60 bp) and shorter than the read length (typically, 150 bp). We formally define this concept later in this paper.

The read colored de Bruijn graph is an attractive concept since chimeric sequences are avoided by keeping each read as a separate color but it does present construction challenges not present in colored de Bruijn graph. One such chal-

lenge is that a metagenomic sample may be too large to store on even the largest servers' hard drives in uncompressed form. For example, a set of metagenomic samples from a cattle production facility [38] contains close to 41 billion 32-mers, with the first sample containing over 57 million reads[1]. Storing each k-mer-read combination with a single bit would require 285 petabytes of space. This mandates that the succinct representation be built in an online fashion such that the complete uncompressed matrix need never be stored explicitly. Therefore, we present a succinct data structure to construct and store the read colored de Bruijn graph, which extends the representation of Muggli et al. [32].

Our contributions. In this paper, we formally define the read colored de Bruijn graph, along with several new concepts, including *multi-colored bulges* and *color coherence*, and demonstrate how they can be used to resolve chimeric sequences that are between 60 bp and 150 bp. Our experiments show the utility of LueVari since it is the only SNP caller that reliably reports sequences (containing the identified SNPs) that spanned on average 47.5% of the AMR gene. It is able to accomplish this without a reference genomes. Whereas, GATK and SAMtools only reported specific loci and require a reference to do so. This is a significant advantage as it allows specific (fingerprinted) AMR genes to be detected in de novo fashion, which is needed for the complete characterization of the resistome. Both SAMtools and LueVari high accuracy on the simulated data but SAMtools reported a high number of false positives. GATK had no false positives but low accuracy ($\leq$ 7%) on simulated data. We demonstrate the utility of our method in characterizing the resistome of a food production facility in the United States. Lastly, we note that LueVari is freely available at `https://github.com/baharpan/cosmo/tree/LueVari`.

2 Related Work

Metagenomics enables the study of microbiomes as whole communities, allowing the assembly and analysis of genomes that belong to both known and unknown bacterial species, many of which remain difficult to culture in isolation [46]. For completeness, we give an brief overview of metagenomic assemblers in addition to our discussion of metagenomic variant callers.

There are two major approaches being used for assembly of metagenomic data: the overlap graph and the de Bruijn graph, the latter of which has become the dominant approach due to its computational efficiency [26]. Some prominent assemblers that are based on the de Bruijn graph include: metaSPAdes [3], SOAPdenovo2 [28], MetaVelvet [34], and MegaHit [23].

For metagenomic variation detection, there are packages, such as metafast [47], crAss [8], Commet [30], compareads [29], and FOCUS [43], that do comparative metagenomics, but return similarity measures instead of specific variants. There are many that align reads to a reference, such as Hansel and Gretal [36], LENS [20], Platypus [42], MIDAS [35], Sigma [1], Strainer [9], and constrains [27].

[1] Reads were trimmed and those with ambiguous base calls removed

Some reference-based read alignment tools, such as QuRe [41], ShoRAH [53], and Vispa [2], additionally use combinatorial optimization techniques to find sequences that best explain the reads and can be used on viral metagenomic samples. The combinatorial optimization approaches are computationally expensive, limiting their applications to only relatively small datasets [12]. In some of these programs, an assembly from one sample may serve as reference for the analysis step. This approach means they will miss variants when the varying section does not align to the reference, so they tend to focus on haplotypes and SNPs. Many programs, such as SAMtools [24] and GATK [31], while commonly used to identify SNPs, indels, and structural variants, were developed to detect variations within diploid organisms, namely eukaryotes, which limits their effectiveness on haploid prokaryotes [54]. Reference-guided assembly packages, like SHEAR [22], also inherently encounter variation relative to a reference, so could be used for structural variant detection by aligning emitted contigs to the reference, if the bubble cannot be directly emitted. However, such programs are not designed for the unique challenges faced in metagenomic samples.

The closest package is MaryGold [37], which does reference-free, metagenomic variant detection. There is some overlap between graph-based variant detection and metagenomic, co-assembly programs. They differ in that co-assembly may focus on maximizing contiguity of each sample independently, whereas variant detection benefits from long contiguous sequences partially shared between two samples, specifically a common start and end. Variant detection also reports the specific alignment of these sequences where co-assembled output would require an all pairs of contigs alignment post processing step to produce the same results. Thus, MaryGold compares itself to a metagenomic scaffolding program called Bambus2 [19], which can report bubbles in a graph constructed from the read data.

3 Methods

As previously discussed, almost all de Bruijn graph approaches for SNP calling share the problem of losing read information when the reads are fragmented into k-mers, introducing the possibility of the sequences containg the variation not being read coherent [33]. This lack of read coherence is more problematic in metagenomics since they can lead to chimeric sequences that contain genomic regions of different species. In this section, we describe the concept of read coloring and demonstrate how it can be incorporated into SNP calling to improve the accuracy in detection of SNPs in AMR genes from metagenomics data.

3.1 Read Colored De Bruijn Graphs

Let $R = \{r_1, .., r_n\}$ be the set of n input reads. We begin by first defining the de Bruijn graph. We use the following standard constructive definition. The de Bruijn graph for R is constructed by first creating an edge for each unique k-mer in R, labelling the vertices of that edge as the prefix and suffix of that k-mer,

and lastly, gluing vertices that have the same label. We let $K = \{k_1, .., k_m\}$ be the set of unique k-mers constructed from R.

Next, we define a *sub-read*, a concept that has not been previously applied to the context of de Bruijn graph based assembly or variant detection. Given a read r and a value k, the sub-reads of r are sets of k-mers constructed by fragmenting r into k-mers until a repeated k-mer occurs, at this point those k-mers are grouped into one sub-read, a new sub-read for r is created and the process of fragmenting into k-mers continues. We note that if there is no repeated k-mer in r then the set of k-mers itself is the sub-read of r. For example, let r be equal to `ACGTACGTACGT` and $k = 3$. The substring `ACGT` is repeated three times in r and therefore, the sets of sub-reads are $S_1 = \{$`ACG, CGT, GTA, TAC`$\}$, $S_2 = \{$`ACG, CGT, GTA, TAC`$\}$, and $S_3 = \{$`ACG, CGT`$\}$.

Lastly, we define the read colored de Bruijn graph $G = (V, E)$ as the de Bruijn graph (constructed for R) that has the modification that there exists a list of colors associated with each edge (k-mer). These colors are stored in a two-dimensional binary array C, where there exists a row for each unique k-mer in K, and a column for each sub-read. Hence, $C(i, j) = 1$ if the k-mer associated with edge $e_i \in E$ is present in jth sub-read; and $C(i, j) = 0$ otherwise. We refer to C as the color matrix. Hence, we first construct the de Bruijn graph on R, split each read in $R = \{r_1, .., r_n\}$ into sub-reads, and lastly, construct the color matrix based on the sub-reads. Genomic repeats that are shorter than or equal to the read length, and longer than or equal to the k-mer length can be resolved using sub-reads as we will see in Subsection 3.3. Figure 1 illustrates an example of a read colored de Bruijn graph.

3.2 Multi-Colored Bulges

We define a *bulge* in G as a set of disjoint paths $(p_1, .., p_n)$ which share a *source vertex* and a *sink vertex*, and refer to p_i for all i from 1 to n as the *branches* of the bulge. We note that $n \leq 4$ due to the alphabet size. We define a path $p = e_1..e_\ell$ in the read colored de Bruijn graph as *color coherent* if the sets of colors corresponding to e_i, e_{i+1}, S_i and S_{i+1}, are such that $S_i \cap S_{i+1} \neq \emptyset$, for all $i = 1, .., \ell - 1$. Next, we refer to *multi-colored bulge* as a bulge whose branches are color coherent but also have disjoint lists of colors. See Figure 2 for an illustration of a multi-colored bulge. Lastly, we define an *embedded multi-colored bulge* in G as a multi-colored bulge that occurs in a branch of another multi-colored bulge.

3.3 Construction of the Read Colored de Bruijn Graph

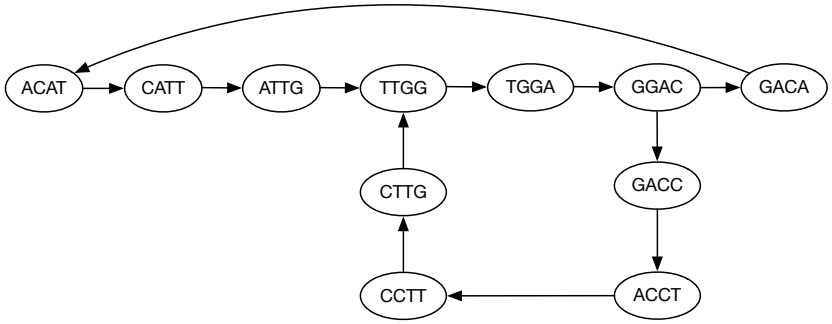
Next, we present a succinct data structure for representing the colored de Bruijn graph that enables it to be constructed and stored efficiently. There are several succinct representations for the de Bruijn graph, including the methods of Simpson et al. [44], Chikhi and Rizk [6], Conway and Bromage [7] and Ye et al. [52]. In this paper, we extend the BOSS data structure [4], which is based on the Burrows-Wheeler Transform (BWT) [5]. We refer the reader to the original paper by Bowe et al. [4] for an overview of BWT, FM-index, and a more thorough

(a) R1: ACATTGGACATTGGACATTGG
R2: ACATTGGACCTTGGACATTGG

(b)
S1: {ACATT,CATTG,ATTGG,TTGGA,TGGAC,GGACA,GACAT}
S2: {ACATT,CATTG,ATTGG,TTGGA,TGGAC,GGACA,GACAT}
S3: {ACATT,CATTG,ATTGG}
S4: {ACATT,CATTG,ATTGG,TTGGA,TGGAC,GGACC,GACCT, ACCTT,CCTTG,CTTGG}
S5: {TTGGA,TGGAC,GGACA,GACAT,ACATT,CATTG,ATTGG}

(c)
{ACATT,CATTG,ATTGG,TTGGA,TGGAC,GGACA,GACAT,GGACC,GACCT,ACCTT,CCTTG,CTTGG}

(d)



(e)

	ACATT	CATTG	ATTGG	TTGGA	TGGAC	GGACA	GACAT	GGACC	GACCT	ACCTT	CCTTG	CTTGG
S1	1	1	1	1	1	1	1	0	0	0	0	0
S2	1	1	1	1	1	1	1	0	0	0	0	0
S3	1	1	1	0	0	0	0	0	0	0	0	0
S4	1	1	1	1	1	0	0	1	1	1	1	0
S5	1	1	1	1	1	1	1	0	0	0	0	0

Fig. 1. **(a)** Illustration of two reads r1 and r2 representing a SNP (shown in grey). **(b)**: Shows the sub-reads of r1 and r2. S1, S2, and S3 originate from r1, and S4 and S5 originate from r2. **(c)** Shows the set of k-mers constructed from r1 and r2. **(d)** Shows the de Bruijn graph constructed from the set of k-mers. **(e)** The color matrix constructed from the sub-reads and de Bruijn graph. By traversing in a color coherent way, the `ACATTGGACATTGGACATTGG` and `ACATTGGACCTTGGACATTGG` will be recovered. In this example we show the transpose of color matrix to save space.

explanation of this data structure. Here, we will give a brief overview of this representation and then demonstrate how it can be extended to construct and store read colored de Bruijn graphs. The first step of constructing this graph G for a given set of k-mers is to add dummy k-mers (edges) to ensure that there exists an edge (k-mer) starting with first $k-1$ symbols of another edges last $k-1$ symbols. These dummy edges ensure that each edge in G has an incoming vertex. After this small perturbation of the data, a list of all edges sorted into right-to-left lexicographic order of their last $k-1$ symbols (with ties broken by the first character) is constructed. We denote this list as F, and refer to its

ordering as co-lexicographic (colex) ordering. Next, we define L to be the list of edges sorted co-lexicographically by their starting vertices with ties broken co-lexicographically by their ending vertices. Hence, two edges with same label have the same relative order in both lists; otherwise, their relative order in F is the same as their labels lexicographic order. The sequence of edge labels sorted by their order in list L is called the *edge-BWT* (EBWT). Now, let B_F be a bit vector in which every 1 indicates the last incoming edge of each vertex in L, and let B_L be another bit vector with every 1 showing the position of the last outgoing edge of each vertex in L. Given a character c and a vertex v with co-lexicographic rank rank(c), we can determine the set of vs outgoing edges using B_L and then search the EBWT(G) for the position of edge e with label c. Using B_F we can find the co-lexicographical rank of es outgoing edge. With repeating this process we can traverse the graph.

After we construct the BOSS representation of the de Bruijn graph on R, we store each k-mers and their lexicographical order in a map M so it can be used for the construction of the color matrix. We use Elias-Fano vector encoding to store C since it permits on-line construction as long as all 1 bits are added in increasing order of their index in the vector. For example, column six of C cannot be filled before column five. Therefore, we build the color matrix by initializing each position to 0 and then updating each row at a time. We recall that each row corresponds to a k-mer and the rows are sorted in lexicographical order. Thus, to efficiently find the row in C corresponding to a particular k-mer, we find its' lexicographical ordering using M, which we denote as i. Next, we align the k-mer to the union of all the sub-reads using BWA [14] in order to find all the sub-reads that contain it. Next, we store the indices of these sub-reads into an array, sort this array, and update the ith row of C in this order, i.e., $C(i, j) = 1$ for all jth sub-reads containing the ith k-mer. Storing and sorting the indices ensures that we meet the construction requirement of Elias-Fano.After we construct and compress (using Elias-Fano) a row, i.e., a binary vector, we append it to growing color matrix C. We continue with this process until all k-mers have been explored.

3.4 Multi-Colored Bulge Search

We for multi-colored bulges by iterating through each vertex in G, determining those that are potential source vertices, and traversing G at those that are potential source vertices to find one or more potential sink vertices. A vertex is determined to be a potential source vertex if the out-degree is greater than one and the sets of colors of the adjacent edges are disjoint. Thus, we can determine whether a vertex v is a potential source vertex by determining the out-degree fusing B_L. Suppose v has index i in colex order. We consider the ith 1 in B_L, if there are ℓ preceding 0s ($\ell < 4$ due to alphabet size) before ith 1 in B_L, then the out-degree of v is $\ell + 1$. Recall that a 1 in B_L indicates that it is the last out-going edge. After locating v's corresponding 1 in B_L, the number of preceding 0s plus one will be the out-degree of v. Next, after finding a vertex with out-degree greater than one, we determine the colors of its' adjacent edges. We

recall that lexicographical order of the edges (k-mers) as their row indices in the color matrix C. Let v be a vertex that has out-going edges e_1 and e_2 with lexicographical order of i and j, respectively. The colors for e_1 correspond to the set of columns k for which $C(i,k)$=1 and for e_2 is the set of columns $k^{'}$ for which $C(j,k^{'}) = 1$. If these two sets are disjoint then v is a potential source vertex.

Next, we perform depth first search at each potential source vertex after it is found, comparing the colors of the next edge with those of the previous one. If these sets are color coherent and there exists at least one unvisited color then we mark the next color as visited and continue with the traversal; otherwise, we terminate. When we encounter a vertex that has in-degree greater than one then it is stored as a potential sink vertex, along with the potential source vertex and the set of branches. We can find the in-degree of vertex by performing a similar process as described above to find the out-degree using B_F. We note that we do not stop the traversal after visiting a potential sink vertex, but instead, continue traversing until all potential sink vertices that are reachable from the source vertex are visited and stored. We stop traversing after all possible outgoing edges have been visited. At the conclusion of the traversal algorithm for a potential source vertex, a list of tuples (sink vertex, source vertex, and branches) describing each bulge is stored and ordered by occurrence. We note that the iterative depth first search algorithm of our traversal method can deal with multiple embedded multi-colored bulges.

This traversal is illustrated in Figure 2. In this illustration, we see that the out-degree of vertex `ACG` is equal to 2 since colex order is four and the fourth 1 in B_L occurs at $B_L[5]$. We determine the two out-edge labels by considering EBWT(G)[3,4] (labels are starting from 0). We see that they are C and T. Next, suppose we want to follow edge e_1 with label `C`. Since there are only two edges with label `A` in the graph, and `C` in EBWT(G)[0,1,2,3] is 2, so e_1 is $F[3]$. Now with counting number of 1s in B_F[0,1,2,3], we see that e_1 has the colex order of 3. With checking the 4th element in L (since labels starts from 0, element with order 3 happens in 4th position) we find the out-vertex which is `CGC`. (Note that the edge e_1 is `ACGC`). With doing the same, we find the other out-edge (e_2) which is `ACGT`. Next, we need to check the colors of e_1 and e_2. With checking the position 0 and 1 in C^T (transpose of C) we see that `ACGC` belongs only to r_1 and `ACGT` only belongs to r_2 (Note that the order of k-mers in the color matrix is lexicographical; hence `ACGC` has order 0 and `ACGT` has order 1). Now that the colors of two out-edges are mutually different, we note the vertex `ACG` as a potential source vertex. With constantly checking the color sets, we will explore the branch starting with `ACGC` (let's call this branch p_1). The potential sink vertex on the way of p_1 is vertex `ATC`. Since its in-degree is two (greater than one) and its incoming edges `CATC` and `TATC` only belongs to one of the reads. We store this vertex as a potential sink vertex of p_1 and since in this case this is the only reachable potential sink vertex from the source vertex `ACG`, the exploring of other branch (p_2 starting with `ACGT`) begins. Since this branch also meets the vertex `ATC` (the potential sink vertex of p_1) through the edge `TATC`, a bulge is detected and traversing will stop.

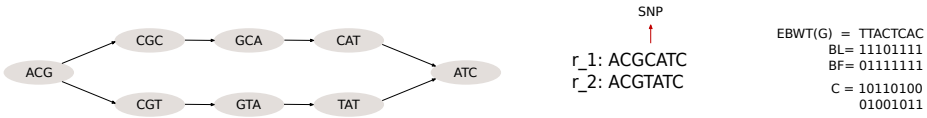


Fig. 2. Left: Example of a bulge which comes from the existence of a SNP in one gene. **Center**: Reads 1 and 2 each supporting one of the branches in the bulge. **Right**: The representation of read colored succinct de Bruijn graph for the bulge on the left side.

3.5 SNP Recovery

Lastly, we process each multi-colored bulge b with branches $\{p_1, .., p_n\}$ by recovering the longest color coherent path that occurs prior to the source vertex s by starting at s and travelling backward on the incoming vertices as long as their exists an unambiguous incoming edge, implying there exists one incoming edge (possibly part of a branch of an embedded bulge) can be added to the current path and have it remain color coherent. If there exists such an edge then it is added to the current path and the traversal backward continues; otherwise the traversal is halted and the current path p_s is saved. Similarly, a color coherent outgoing path is obtained from traversing the graph in a forward direction from the sink vertex t. We refer to this resulting path as p_t. Lastly, p_s is concatenated to each of the branch in $\{p_1, .., p_n\}$, p_t is concatenated to each of these resulting paths, and their corresponding sequences are outputted. The variation between these sequences are recovered by alignment. This process is continues for all multi-colored bulge.

4 Results

We evaluate LueVari by comparing its' performance against competing methods on both simulated and real metagenomic data. The simulated data established the sensitivity and specificity of all the methods; whereas, real dataset demonstrate the ability to identify distinct (fingerprinted) AMR genes in a sample taken from a food production facility. All experiments were performed on a 2 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz server with 1 TB of RAM, and both resident set size and user process time were reported by the operating system.

4.1 Results on Simulated Data

We simulated two metagenomics datasets using BEAR, a metagenomics read simulator [18]. The datasets were generated to imitate the characteristics (number of reads, number of distinct AMR genes, and their copy number) of real shotgun metagenomics data generated for resistome analysis–namely those of Noyes et al. [38] and Gibson et al. [13]. The number of reads is relatively small for a single dataset since eukaryote DNA is filtered prior to resistome analysis [38], which can pose a challenge for many SNP callers. The first simulated

dataset consists of 193,752 paired-end short reads from 30 AMR genes from the MEGARes database [21], two copies of the *E. coli* K-12 MG 1655 reference genome, and two copies of the salmonella reference genome. The average copy number of the AMR genes is 294x. Hence, we refer to this as the "294x dataset".

The second dataset consists of 2,504,238 paired-end sequence reads simulated from 3,824 AMR genes from MEGARes database, two copies of the *E. coli* K-12 MG 1655 reference genome, and two copies of the salmonella reference genome. The average copy number of the AMR genes is 25x, Hence, we refer to this as the "25x dataset". We used a 1% error rate for both these simulations. SNPs were inserted into the AMR genes at a 0.05% polymorphism rate. Hence, 23 and 681 SNPs inserted into the 294x and 25x datasets, respectively.

We compared our method against several SNP and variant callers. Several methods were unable to be compared due to their unsuitability. Marygold and Bambus2 were unable to run and are no longer supported [11]. kSNP3 could not be used for comparison because it is specifcally designed for calling the variants between the samples, i.e., sequence data from different genomes [10]. DiscoSnp [49] ran very quickly (11m CPU time) but did not produce reasonable output. This is unsurprising since we are re-purposing this tool as it specifically designed for eukaryote genomes. GATK [31] and SAMtools [24] were able to be compared against LueVari. The results are summarized in Table 1. Both GATK and SAMtools are reference-guided, and thus, we ran them with their recommended parameters and the MEGARes database as reference genome. LueVari is reference-free and was ran as such. All methods had competitive run-time ($\leq$ 5h CPU time) and used a reasonable amount of memory ($\leq$ 6GB).

LueVari is the only SNP caller that reliably reported sequences (containing the identified SNPs) that spanned on average 47.5% of the AMR gene, which are typically between 1,000 and 1,400 bp in length. Out of the 2,749 reported SNPs, 436 (16%) of them were reported in a sequence spanning more than 80% of the target AMR gene. GATK and SAMtools only reported specific loci and require a reference to do so. This is a significant issue in the analysis of AMR genes since existing AMR databases are largely believed to be incomplete [45]. This will allow specific (fingerprinted) AMR genes to be detected in de novo fashion. Both SAMtools and LueVari detected all the inserted SNPs but SAMtools reported a high number of false positives. GATK had no false positives but only detected a small portion of the inserted SNPs (0% and 6.3% for 294x and 25x, respectively). This performance was due to the filtering step in GATK that removed a large number of reads (50% and 48% for 294x and 25x, respectively).

4.2 Results on Real Metagenomic Data

We demonstrate the ability of LueVari to analyze real shotgun metagenomics data, consisting of 29,415,748 paired-end reads that were sequenced on an Illumina HiSeq 2500 system. The samples were selected across the beef production system, which contain different interventions (such as, high-heat and lactic acid treatment) aimed at decreasing AMR in consumable beef. Hence, this dataset is used to explore how microbial communities surrounding beef production facilities

	LueVari		GATK		SAMtools	
	Total	Total Inserted	Total	Total Inserted	Total	Total Inserted
294x dataset	44	23	0	0	2,038	23
25x dataset	2,749	681	43	43	89,421	681

Table 1. Results on the simulated data with 193,752 and 2,504,238 sequence reads. Total refers to the total number of SNPs reported by all programs. Total inserted reported the total number of inserted SNPs that were detected by all programs.

evolve in the presence of different food production interventions that aim to reduce pathogen load [38]. We ran LueVari on this shotgun metagenomic dataset and detected 2,129 distinct AMR genes, i.e., AMR genes found in MEGARes containing an unique SNP. This was accomplished in 25 CPU hours. In future works, we plan to analyze these SNPs and apply them to track the movement and evolution of the genes.

References

1. T.-H. Ahn, J. Chai, and C. Pan. Sigma: Strain-level inference of genomes from metagenomic analysis for biosurveillance. *Bioinformatics*, 31(2):170–177, 2015.
2. I. Astrovskaya et al. Inferring viral quasispecies spectra from 454 pyrosequencing reads. *BMC bioinformatics*, 12(Suppl 6):S1, 2011.
3. A. Bankevich et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J Comp Bio*, 19(5):455–477, 2012.
4. A. Bowe et al. Succinct de Bruijn graphs. In *Proc. WABI*, pages 225–235, 2012.
5. M. Burrows and D.J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
6. R. Chikhi and G. Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms Mol. Biol.*, 8(22), 2012.
7. T. C. Conway and A. J. Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479486, 2011.
8. B. E Dutilh et al. Reference-independent comparative metagenomics using cross-assembly: crAss. *Bioinformatics*, 28(24):3225–3231, 2012.
9. J.M. Eppley et al. Strainer: software for analysis of population variation in community genomic datasets. *BMC bioinformatics*, 8(1):398, 2007.
10. S.N. Gardner, T. Slezak, and Hall. B.G. SNP detection and phylogenetic analysis of genomes without genome alignment or reference genome. *Bioinformatics*, 31(17):2877–2878, 2015.
11. Jay Ghurye. personal communication, May 2017.
12. J.S. Ghurye, V. Cepeda-Espinoza, and M. Pop. Metagenomic Assembly: Overview, Challenges and Applications. *Yale J Biol Med*, 89(3):353–362, 2016.
13. M.K. Gibson, K.J. Forsberg, and G. Dantas. Improved annotation of antibiotic resistance determinants reveals microbial resistomes cluster by ecology. *ISME*, 9(1):207–216, 2014.
14. Li H. and Durbin R. Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics*, 25:1754–60, 2009.

15. The White House. National action plan for combating antibiotic-resistant bacteria, 2014. `https://www.whitehouse.gov/sites/default/files/docs/national_action_plan_for_combating_antibotic-resistant_bacteria.pdf`.
16. Z. Iqbal et al. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature Genetics*, 44(2):226–232, 2012.
17. Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature genetics*, 44(2):226–232, 2012.
18. S. Johnson et al. A better sequence-read simulator program for metagenomics. *BMC Bioinformatics*, 15(Suppl 9):S14, 2014.
19. S. Koren, T. J Treangen, and M. Pop. Bambus 2: scaffolding metagenomes. *Bioinformatics*, 27(21):2964–2971, 2011.
20. V. Kuleshov et al. Synthetic long-read sequencing reveals intraspecies diversity in the human microbiome. *Nature Biotech*, 34(1):64–69, 2016.
21. S. Lakin, C. Dean, N. Noyes, A. Dettenwanger, A. Ross, E. Doster, P. Rovira, Z. Abdo, K. Jones, J. Ruiz, Belk K., P. Morley, and C. Boucher. MEGARes: an antimicrobial resistance database for high throughput sequencing. *Nucleic Acids Research*, 45(D1):D574–D580, 2017.
22. S.R. Landman et al. SHEAR: sample heterogeneity estimation and assembly by reference. *BMC Genomics*, 15(1):84, 2014.
23. D. Li et al. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics*, 31(10):1674, 2015.
24. H. Li et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.
25. Z. Li et al. Comparison of the two major classes of assembly algorithms: overlap-layout-consensus and de-bruijn-graph. *Brief Funct Genomics*, 11(1):25–37, 2012.
26. Z. Li et al. Comparison of the two major classes of assembly algorithms: overlaplay-outconsensus and de Bruijn graph. *Briefings in Functional Genomics*, 11(1):25, 2012.
27. C. Luo et al. ConStrains identifies microbial strains in metagenomic datasets. *Nature Biotech*, 33(10):1045–1052, 2015.
28. R. Luo et al. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, 1(1):1, 2012.
29. N. Maillet et al. Compareads: comparing huge metagenomic experiments. *BMC Bioinformatics*, 13(19):1, 2012.
30. N. Maillet et al. COMMET: comparing and combining multiple metagenomic datasets. In *In Proc of IEEE BIBM*, pages 94–98, 2014.
31. A. McKenna et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20:1297–303, 2010.
32. M.D. Muggli, A. Bowe, N.R. Noyes, P. Morley, K. Belk, R. Raymond, T. Gagie, S. Puglisi, and C. Boucher. Succinct colored de bruijn graphs. *Bioinformatics*, page To appear, 2017.
33. E.W Myers. The fragment assembly string graph. *Bioinformatics*, 21(Suppl 2):ii79–ii85, 2005.
34. T. Namiki et al. MetaVelvet: an extension of Velvet assembler to de novo metagenome assembly from short sequence reads . *Nucleic Acids Res*, 40(20):e155, 2012.

35. S. Nayfach and K.S. Pollard. Population genetic analyses of metagenomes reveal extensive strain-level variation in prevalent human-associated bacteria. *bioRxiv*, page 031757, 2015.
36. S.M. Nicholls et al. Advances in the recovery of haplotypes from the metagenome. *bioRxiv*, page 067215, 2016.
37. J.F. Nijkamp et al. Exploring variation-aware contig graphs for (comparative) metagenomics using MaryGold. *Bioinformatics*, 29(22):2826–2834, 2013.
38. N.R. Noyes et al. Resistome diversity in cattle and the environment decreases during beef production. *eLife*, 5:e13195, 2016.
39. N.R. Noyes, X. Yang, L.M. Linke, R.J. Magnuson, A. Dettenwanger, S. Cook, R. Zaheer, H. Yang, D. Woerner, I. Geornaras, J. McArt, S.P. Gow, J. Ruiz, K.L. Jones, C.A. Boucher, T. McAllister, P.S. Morley, and K.E. Belk. Characterization of the resistome in manure, soil and wastewater from dairy and beef production systems. *Scientific Reports*, 6:24645, 2016.
40. World Health Organization. Diet, nutrition and the prevention of chronic diseases. Technical Report 916, WHO Technical Report Series, Geneva, Switzerland, 2003. Report of a joint WHO/FAO expert consultation.
41. M.C.F. Prosperi and M. Salemi. QuRe: software for viral quasispecies reconstruction from next-generation sequencing data. *Bioinformatics*, 28(1):132–133, 2012.
42. A. Rimmer et al. Integrating mapping-, assembly-and haplotype-based approaches for calling variants in clinical sequencing applications. *Nature Genetics*, 46(8):912–918, 2014.
43. G.G.Z Silva et al. FOCUS: an alignment-free model to identify organisms in metagenomes using non-negative least squares. *PeerJ*, 2:e425, 2014.
44. Jared T Simpson and Richard Durbin. Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, 26(12):i367–i373, 2010.
45. A.C. Singer, H. Shaw, V. Rhodes, and A. Hart. Review of antimicrobial resistance in the environment and its relevance to environmental regulators. *Front Microbiol*, 7:1728, 2016.
46. Susannah Green Tringe, Christian von Mering, Arthur Kobayashi, Asaf A. Salamov, Kevin Chen, Hwai W. Chang, Mircea Podar, Jay M. Short, Eric J. Mathur, John C. Detter, Peer Bork, Philip Hugenholtz, and Edward M. Rubin. Comparative metagenomics of microbial communities. *Science*, 308(5721):554–557, 2005.
47. V.I. Ulyantsev et al. MetaFast: fast reference-free graph-based comparison of shotgun metagenomic data. *Bioinformatics*, 32(18):2760–7, 2016.
48. United State Department of Agritculture. Food safety and inspection strategic plan 2017-2021. Technical report, USDA Report Series, Washington, DC, 2017.
49. R. Uricaru et al. Reference-free detection of isolated SNPs. *Nucleic Acids Res*, 43(2):e11, 2015.
50. M. Willmann and S. Peter. Translational metagenomics and the human resistome: confronting the menace of the new millennium. *J Mol Med*, 95(1):41–51, 2017.
51. X. Yang, N.R. Noyes, E. Doster, J.N. Martin, L.M. Linke, R.J. Magnuson, H. Yang, I. Geornaras, D.R. Woerner, K.L. Jones, J. Ruiz, C. Boucher, P.S. Morley, and K.E. Belk. Use of metagenomic shotgun sequencing technology to detect foodborne pathogens within the microbiome of the beef production chain. *Applied and Environmental Microbiology*, 82(8):2433–2443, 2016.
52. C. Ye, Z.S. Ma, C.H. Cannon, M. Pop, and D.W. Yu. Exploiting sparseness in de novo genome assembly. *BMC Bioinformatics*, Suppl 6:S1, 2012.
53. O. Zagordi et al. ShoRAH: estimating the genetic diversity of a mixed sample from next-generation sequencing data. *BMC Bioinformatics*, 12(1):1, 2011.

54. M. Zojer et al. Variant profiling of evolving prokaryotic populations. *PeerJ*, 5:e2997, 2017.