Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

OXFORD

Subject Section

Resistome SNP Calling via Read Colored de Bruijn Graphs

Bahar Alipanahi* Martin D. Muggli Musa Jundi Noelle Noyes and Christina Boucher

Department of Computer & Information Science & Engineering, University of Florida, Gainesville, 32611, US.

*To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: The resistome, which refers to all of the antimicrobial resistance (AMR) genes in pathogenic and non-pathogenic bacteria, is frequently studied using shotgun metagenomic data [14, 47]. Unfortunately, few existing methods are able to identify single nucleotide polymorphisms (SNPs) within metagenomic data, and to the best of our knowledge, no methods exist to detect SNPs within AMR genes within the resistome. The ability to identify SNPs in AMR genes across the resistome would represent a significant advance in understanding the dissemination and evolution of AMR, as SNP identification would enable"fingerprinting" of the resistome, which could then be used to track AMR dynamics across various settings and/or time periods.

Results: We present LueVari, a reference-free SNP caller based on the read colored de Bruijn graph, an extension of the traditional de Bruijn graph that allows repeated regions longer than the k-mer length and shorter than the read length to be identified unambiguously. We demonstrate LueVari was the only method that had reliable sensitivity (between 73% and 98%) as the performance of competing methods varied widely. Furthermore, we show LueVari constructs sequences containing the variation which span 93% of the gene in datasets with lower coverage (15X), and 100% of the gene in datasets with higher coverage (30X).

Availability: Code and datasets are publicly available at `https://github.com/baharpan/cosmo/tree/LueVari`.

1 Introduction

Antimicrobial resistance (AMR) refers to the ability of an organism to persist in the face of exposure to an antimicrobial agent, i.e., an antibiotic. 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 metagenomic data has already been generated to characterize the resistome in clinical [14, 47] and food production [38] settings. The characterization of various resistomes includes identification of specific AMR genes and a measure of their abundance. One element of the resistome that has not yet been sufficiently characterized is the profile of single nucleotide polymorphisms (SNPs) within the AMR genes that comprise any given resistome. Characterization of the SNP profile would be very informative, as such a profile would allow specific AMR genes to be tracked across various settings and time points. Such *traceability* would greatly advance our understanding of how AMR disseminates and evolves; however, the ability to perform such traceability studies hinges on the accurate identification of AMR SNPs within metagenomic data. While methods do exist for identification of SNPs within eukaryote species, few current methods are suitable for metagenomic data—a sentiment expressed by Nijkamp et al. [37] when they state: "..there is a lack of algorithms for finding such variation in metagenomes."

In this paper, we develop a scalable, reference-free method for identifying SNPs in metagenomic data, which we call LueVari. In order to be suitable for metagenomics, it is important that any SNP calling method be reference-free, as only a small fraction of the total diversity of microbes are culturable [44] and therefore, the majority of organisms within a given metagenomic sample will likely not have a reference genome in the near future. Hence, reference-guided tools will only detect

© The Author 2015. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com

a small fraction of available SNPs in metagenomes. Current reference-free methods specifically designed for metagenomic data typically require an "assembly" step that uses an overlap-layout-consensus (OLC) approach [18] or an Eulerian (de Bruijn graph) paradigm [3, 27, 34, 23, 45]. SNP detection in metagenomic data is challenging because it exacerbates the weaknesses of these two algorithmic approaches. De Bruijn graph-based SNP callers require each read to be fragmented into k-length sequences (called k-mers), and are thus prone to inappropriate combining of read segments from different species, resulting in *chimeric sequences*. OLC approaches–such as Bambus2 [18]–have an advantage over de Bruijn graph approaches in that they build an overlap graph in which the entirety of a read (rather than read k-mers) corresponds to a single node in the graph. As a result, sequences and SNPs recovered from these graphs are supported by collections of entire reads, rather than k-mers—an attribute of overlap graphs known as *read coherence* [33]. In theory, this reduces the frequency of chimeric SNPs. Unfortunately, OLC approaches suffer from computational inefficiency and thus are unable to handle large datasets such as metagenomic sequence datasets [25].

De Bruijn graph approaches have greater capacity to scale to larger datasets, especially in light of existing succinct data structures for representing de Bruijn graphs [43, 7, 8, 48, 4]. However, current de Bruijn graph-based methods lack the read coherence needed to detect SNPs accurately. Thus, there currently exists a significant trade-off between only being able to call SNPs within short sequences constructed from non-branching paths, or risk of chimeric sequences that arise from spanning branches in the graph. Furthermore, the potential for chimeric sequences becomes more frequent in resistome analysis due to sequence homology between AMR genes. Specifically, AMR genes within the same AMR class can share homologous regions that are typically between 60 bp and 150 bp in length [20], which is longer than the typical k-mer value and shorter than the read length. Hence, such homologous 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 and correctly reconstruct the sequences corresponding to these AMR 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 large-scale resistome analysis, which is likely to encompass 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 phase in use of sequencing to monitor AMR by 2021—if accomplished, even at a small scale, thousands of samples will need to be sequenced and analysed to monitor food-borne outbreaks.

Therefore, an approach for identification of SNPs in AMR genes within metagenomic data 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 a de Bruijn graph based SNP caller. It extends the concept of the colored de Bruijn graph, which was first introduced by Iqbal et al. [15] for the detection of variants in eukaryotic species. Given a set of n samples, the colored de Bruijn graph extends the traditional de Bruijn graph in that each node (and edge) in the graph has a set of associated colors in which each color corresponds to one of the n samples. In Iqbal et al.'s [15] original application, each sample corresponded to the sequence data of one individual and traversal of the colored de Bruijn graph allowed for sequence variation to be detected, along with the individuals containing that variation. Although the colored de Bruin graph allows for detection of genetic variation among individuals of a population, it lacks read coherence. In order to overcome this limitation, we propose an approach that we term a *read colored de Bruijn graph*. Briefly, a read colored de Bruijn graph annotates each node (and edge) by an unique color that corresponds to each individual sequence read (in one or more samples), allowing for read coherence to be preserved among paths longer than the k-mer size (typically, $k \leq 60$ bp). We formally define this concept later in this paper.

The read colored de Bruijn graph is an attractive concept because it avoids chimeric sequences by maintaining each read as a separate color. However, it does present construction challenges not present in colored de Bruijn graphs. One such challenge 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 main contributions. In this paper, we define the read colored de Bruijn graph along with several new concepts and demonstrate how it helps resolve chimeric sequences. This is allows LueVari to not only report the SNP but also correctly reconstruct the sequence containing the SNP. This is vital to metagenome applications where a reference genome is frequently unknown. Hence, ours was the only method to output sequences that span between 93% to 100% of the gene–even in the case of shallow coverage. For example, when the sequence data corresponds to 30X coverage and the SNP rate was 0.005, LueVari constructed all genes containing SNPs whereas DiscoSNP reported sequences that covered 71% (on average) of the gene, and Bubbleparse reported sequences that covered 41.76%. In addition, when compared to current state-of-the-art methods, LueVari was the only method that had reliably high sensitivity (between 73% and 98.5%); DiscoSNP and Bubbleparse had sensitivity between 32.8% and 91%, and 75% and 85%, respectively. Lastly, we show LueVari has the scalability to analyze all the data from a large-scale data collection effort aimed at characterizing the resistome of a commercial food production facility in the United States.

2 Related Work

As previously mentioned, there are both reference-based and reference-free SNP and variant callers. the majority of the SNP callers are developed for diploid organisms, namely eukaryotes, which limits their effectiveness on prokaryotes [51]. However, there do exist some methods that aim to detect variation in metagenomes. Among the reference-based variant callers, there are those that first align reads to the reference and then process this alignment. These methods include Hansel and Gretal [36], LENS [19], Platypus [41], MIDAS [35], Sigma [1], Strainer [11], and ConStrains [26]. Similar to these algorithms, are reference-based read alignment methods that use combinatorial optimization techniques to find sequences that best explain the reads and thus, can be used for variant detection in metagenomic samples. These methods include QuRe [40], ShoRAH [50], and Vispa [2]. The combinatorial optimization approaches are computationally intensive, limiting their applications to only relatively small datasets [13].

There are several reference-free variant callers. MaryGold [37], Bubbleparse [22], DiscoSNP [46],metafast [45], crAss [9], Commet [29], compareads [28], and FOCUS [42] are all reference-free. MaryGold [37], Bubbleparse [22] and DiscoSNP [46] are comparable to LueVari in that they are able to detect variants in metagenomes without a reference. There are several algorithms that do comparative metagenomics and return a similarity measure rather than specific variants that merit mention. These include metafast [45], crAss [9], Commet [29], compareads [28], and

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

FOCUS [42]. Lastly, there are several reference-free variant callers that designed for a specific application and not directly comparable to LueVari, including KSNP3 [12], NIKS [17], Stacks [6] and 2k+2 [49]. Stacks [6] is designed for restriction enzyme based sequencing protocols. NIKS [17] is designed for whole-genome sequencing protocols. KSNP3 [12] and 2k+2 [49] are designed to detect the variations between datasets.

3 Definition of Read Coloring

As we previously discussed, de Bruijn graph approaches for SNP calling lose read information when the reads are fragmented into k-mers, which introduces the possibility of the sequences containg the SNP not being read coherent [33]. This lack of read coherence exacerbates the difficulty of detecting SNPs in metagenomes. In this section, we formally define the concept of the read colored de Bruijn graph and show how it is used by LueVari to detect SNPs accurately.

3.1 Read Colored De Bruijn Graphs

We begin by defining the (traditional) de Bruijn graph. We let $R = \{r_1, \ldots, r_n\}$ be the set of n input reads. We construct the de Bruijn graph for R by creating an edge for each k-mer in R, labelling the nodes of that edge as the $(k-1)$-mer prefix and $(k-1)$-mer suffix of that k-mer, and lastly, gluing nodes that have the same label. Here is an example of gluing: if node v_1 with label `ACG` has outgoing edge with label `C` and node v_2 with the same label `ACG`, has outgoing edge with label `G`, since both of nodes have same label, to make sure that all of labels in de Bruijn graph are distinctive, we glue them which means that the outgoing edge of v_2 will be added to v_1, hence v_1 has two outgoing edges with labels `C` and `G`, and v_2 will be deleted. The story is the same for incoming edges. Next, we define the concept of a *sub-read*, which is necessary for defining the read colored de Bruijn graph. Given a read r of length ℓ it follows that there are $\ell - k + 1$ k-mers. We denote the k-mers of r as $s_1^r, \ldots, s_{\ell-k+1}^r$. We define the *sub-reads* of r as the sets of k-mers, denoted as $S_1^r, \ldots, S_n^r$, $(n \leq \ell - k + 1)$, where the following is true: (1) every k-mer of r is contained in one S_i^r, and (2) $s_x^r \neq s_y^r$ for all s_x^r and s_y^r contained in the same set S_i^r.

More intuitively, we construct the set of sub-reads of r by creating k-mers from r until a repeated k-mer occurs; then, when we see a repeated k-mer, those k-mers are grouped into one sub-read and a new sub-read for r is created. We continue this process until all k-mers are added to a set. We note that if there are no repeated k-mers in r then the set of k-mers itself is the sub-read of r. We now give an an example to illustrate this: 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 $S_1^r = \{$`ACG, CGT, GTA, TAC`$\}$, $S_2^r = \{$`ACG, CGT, GTA, TAC`$\}$, and $S_3^r = \{$`ACG, CGT`$\}$. We note that making sub-reads in the case of repeated k-mers will disambiguate the traversal of whirls (directed cycles) that have length greater than or equal to k and less than or equal to ℓ . For instance in above example, the read `ACGTACGTACGT`, will make a whirl at node `ACG` that makes the traversing ambiguous. Note that this problem will be solved if we follow the sub-reads S_1^r, S_2^r and S_3^r, which lead us to traverse `ACG` three times and finish the traversing at `CGT`.

Next, we define the read colored de Bruijn graph for a set of reads R constructively as follows: (1) we create the sub-reads of r for all $r \in R$ (denoted as $S_1^r, \ldots, S_n^r$, $n \leq \ell - k + 1$.) (2) we assign a color c_i^r for every sub-read S_i^r, and (3) we build the de Bruijn graph as above with the modification that color c_i^r is associated every edge e if the k-mer corresponding to e is contained in S_i^r. Hence, we view the read colored de Bruijn graph as a graph $G = (V, E)$ and a binary matrix C, where there exists a row for each distinct k-mer, and a column for each sub-read and $C(i, j) = 1$ 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. We illustrate a read colored de Bruijn graph in Figure A.1 in the Supplement. Briefly, we mention that the addition of read coloring to the de Bruijn graph resolves repeats that are shorter than or equal to the read length, and longer than or equal to the k-mer length. We discuss this more in-depth in Subsection 4.1.

3.2 Multi-Colored Bulges

We define a *bulge* in G as a set of disjoint paths $(p_1, \ldots, p_n)$ which share a source and sink node, where we refer to paths $(p_1, \ldots, p_n)$ as the branches. Next, we define a path $p = e_1 \ldots e_\ell$ (e_is ($1 \leq i \leq \ell$) are edges) 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 $i = 1, \ldots, \ell-1$. Thus, we refer to *multi-colored bulge* as a bulge where the branches are color coherent and also have disjoint lists of colors. We illustrate a multi-colored bulge in Figure 1. 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.

4 Methods

In this section, we go over the main steps of LueVari in detail.

4.1 Construction of the Read Colored de Bruijn Graph

We build the read colored de Bruijn graph for reads R by first constructing the de Bruijn graph for R, computing the sub-reads of R, and creating the color matrix (in a compressed format). We use the BOSS data structure [4], which is based on the Burrows-Wheeler Transform (BWT) [5] for constructing and storing the graph. Here, we will give a brief overview of this representation and we refer the reader to the original paper by Bowe et al. [4] for a more thorough explanation of this data structure.

Our first step in constructing this graph G for a given set of k-mers is to add dummy k-mers (edges) which ensure that there exists an edge (k-mer) starting with first $k-1$ symbols of another edges last $k-1$ symbols and thus, that the label of each edge and node G can be recovered. After this small perturbation of the data, we construct 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). We denote this list as F, and refer to its ordering as co-lexicographic (colex order). Next, we define L to be the list of edges sorted co-lexicographically by their starting nodes with ties broken co-lexicographically by their ending nodes. Thus, we note that two edges with same label have the same relative order in both lists; otherwise, their relative order in F is the same as the lexicographic order of their labels. The sequence of edge labels sorted by their order in list L is called the *edge-BWT* (EBWT). Now, we let B_F be a bit vector in which every 1 indicates the last incoming edge of each node in L, and let B_L be another bit vector with every 1 showing the position of the last outgoing edge of each node in L. Given a character c and a node v with co-lexicographic rank rank(c), we can determine the set of outgoing edges of v, using B_L and then search the EBWT(G) for the position of edge e with label c. We can find the co-lexicographical rank of outgoing edge of e using B_F, and thus, traverse the graph by repeating this process,.

After we construct the BOSS representation of the de Bruijn graph on R, we align all k-mers to the union of all the sub-reads using Bowtie [21] in order to find all the sub-reads that contain each of k-mers. (We only consider perfect matches). Then we store the 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 [10, 30, 39] 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, we cannot

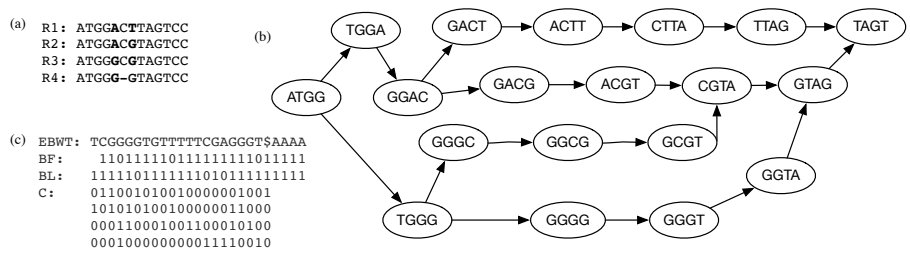

Fig. 1. In this example, we illustrate a complex multi-colored bulge that arises from the existence of multiple SNPs in a single gene. In (a), we show the reads with multiple variations. In (b), we illustrate the de Bruijn graph which was constructed for the set of reads in (b). Lastly, in (c) we give the succinct representation of read-colored de Bruijn graph.

fill column six of C before column five is filled. 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, we first find its lexicographical ordering of a k-mer s_k using M, which we denote as i, in order to find the row in C corresponding to s_k. Using the alignment of k-mers to sub-reads, we store the indices of the sub-reads that contain s_k in a vector A_{s_k}. We sort A_{s_k} and update the ith row of C in this order, i.e., $C(i, j) = 1$ for all jth sub-reads containing the ith s_k. We store and sort the indices in this manner to ensure that we meet the construction requirement of Elias-Fano. After we construct A_{s_k} in a compressed format, we append it to growing color matrix C. We continue with this process until all k-mers have been explored.

4.2 Search for Multi-Colored Bulges

Next, we traverse G in a read-coherent manner in order to determine all multi-colored bulges. In the first step of this traversal we iterate through all nodes in G and determine those that are potential source nodes of a multi-colored bulges, meaning that the out-degree is greater than one and the sets of colors of the outgoing edges are disjoint. Thus, given a node v with index i (in colex order), we determine whether v is a source node as follows. We first calculate out-degree of v using B_L by finding i-th '1' bit in B_L, and counting the number of preceding '0' bits B_L. If the number of '0' bits is ℓ then the out-degree of v is $\ell + 1$. If the out-degree of v is not greater than one then we do not consider it further; otherwise, we determine the colors of its outgoing edges. We recall that the lexicographical order of the edges (k-mers) corresponds to their row index in the color matrix C. Therefore, if v has outgoing edges e_1 and e_2 with lexicographical order of x and y, respectively then we can determine the colors of e_1 by finding all positions ℓ where $C(x, \ell) = 1$ and those for e_2 by finding all positions ℓ' where $C(y, \ell') = 1$. If these two sets are disjoint then v is a potential source node.

Next, we perform (modified) depth first search at each potential source node v to find one or more potential sink nodes. We modify the standard depth first search algorithm by adding a constraint to ensure all paths are color coherent, and by storing multi-colored bulges during the traversal. We ensure the paths of the search are color coherent by determining the intersection of the set of colors of the next outgoing edge with those of the previous one at each iteration, and terminating if there this intersection is empty. Further, we determine the in-degree of v at each iteration[2], and store v as a potential sink node (along with the source node and branches) if the degree is greater than one. We note that we do not stop the traversal after encountering a potential sink node but instead, continue until all nodes reachable from v have been visited.

If we encounter a potential source node u while traversing a path p_v starting at v then we determine whether u has been previously visited. If it has, then we search for u in the set of multi-colored bulges. If u is a source node of a multi-colored bulge with branches $\{p_{u1},\ldots,p_{u\ell}\}$ and sink node t_u, then we do not traverse $p_{u1},\ldots,p_{u\ell}$. Instead, we add each path in $\{p_{u1},\ldots,p_{u\ell}\}$ to p_v and continue traversing from t_u. We note that v will have ℓ branches $\{p_v{+}p_{u1},\ldots, p_v{+}p_{u\ell}\}$ after concatenating the paths. If u has not been previously visited then depth first search is performed, starting with u as a potential source node, to determine whether there exists a multi-colored bulge with source node u. If there does not exist such a bulge then we resume the prior depth first search where v was the starting node. Otherwise, we store the multi-colored bulge (with source node u, set of branches $\{p_{u1},\ldots, p_{u\ell}\}$, and sink node t_u), process the bulge in an analogous manner as just described, and resume the depth first search at t_u. This ensures that we will detect all complex multi-colored bulges.

We illustrate this traversal using an example of a complex multi-colored bulge in Figure 1. We note that dummy incoming edges and one dummy outgoing edge (labeled with \$) have been added to construct the succinct de Bruijn graph but since they do not clarify the traversal, we omit them in the depicted graph. In this example, we iterate through all nodes until we reach `GGAC`, which is the 6th node in colex order. In B_L, the 6th '1' bit has one preceding '0' bit which means that `GGAC` has out-degree 2 since each '1' bit in B_L indicates the last outgoing edge. Now we determine if `GGAC` is a potential source node by first checking if its outgoing edges have disjoint color sets. We find the outgoing edges, which are G and T, by finding the range of $\mathsf{EBWT}[5, 6]$. We determine the lexicographical order of `GGACG` which is 10. By considering the 10th column in color matrix C^T, we can identify the sub-read(s) that `GGACG` comes from, which is only sub-read 2. By doing the same with the other edge `GGACT` of lexicographical order 11, we identify its supporting sub-read(s), which is sub-read 1. We witness that the color sets of outgoing edges are disjoint which implies `GGAC` is a potential source node. Hence, we start traversing at this node and follow the edge with label G (`GGACG`). For finding the destination node of this edge, we count the preceding edges in colex order. We witness that there are five edges with label A, two edges with label C in whole graph and we are at 4th G in EBWT. Thus, we count 1's in $B_F[0, \ldots, 11]$ and determine there are 10. This means that `GGACG` arrives at the 10th nodes in colex order that has incoming edges. Since first node (`$$$$`) has no incoming edge, `GGACG` arrives at 11th node in colex order. The node with colex order 11 is `GACG`. After finding the destination, we check the read coherency by finding the following edge, which is `GACGT`. As mentioned

[2] This is performed in an analogous way as described to find the out-degree; however, in this case we use B_F rather than B_L.

before, we can find the color set for this edge, which only includes sub-read 2. Since color sets of two consecutive edges have at least one intersection, traversing continues. Along the way nodes with in-degree greater than one will be saved as potential sink nodes (similar to what mentioned on B_L to find out-degree, in-degree can be found using B_F). In this example, `TAGT` is such a node. While traversing the other branch, if this node is visited again, the bulge will be detected and the start node, branches, and the end node will be stored. The exact same process will be performed on the second bulge with start node `TGGG`. Finally when the algorithm hits node `ATGG`, by hitting the already visited start nodes, it will not traverse the already known multi-colored bulges on nodes `GGAC` and `TGGG` again. By calling them from the list of multi-colored bulges, their branches will be added to the currently growing branches and by jumping to their end nodes, traversing finishes faster. Essential to our method is that, even though the multi-colored bulge at node `ATGG` comes from a variation of A and G, the reported branches, covers all embedded SNPs along the way, and for the complex bulge at start node `ATGG`, the total of four branches will be reported, which retrieve all four sub-reads with their SNPs.

4.3 Recovery of SNPs

Next, we process each multi-colored bulge b with branches $\{p_1, \ldots, p_n\}$ by recovering the longest color coherent path that occurs prior to the source node s by starting at s and travelling backward on the incoming nodes as long as their exists an unambiguous incoming edge, implying there exists one incoming edge (possibly part of a branch of an embedded multi-colored 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 node t. We refer to this resulting path as p_t. Lastly, p_s is concatenated to each branch in $\{p_1, \ldots, p_n\}$, p_t is concatenated to each of these resulting paths, and their corresponding sequences are emitted. The SNPs in the sequences are recovered by alignment. This process is continued for all multi-colored bulges.

5 Results and Discussion

In this section, we compare the performance of LueVari to other SNP detection methods. In particular, we present the efficiency, sensitivity and precision of LueVari and the competing methods on simulated data, and demonstrate the scalability of LueVari by using it to identify distinct ("fingerprinted") AMR genes in 34 samples taken from a food production facility that had previously-identified AMR genes. 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.

5.1 Results on Simulated Data

We note that majority of SNP calling methods output SNPs along with sequences flanking or containing the variant—the lengths of these sequences has an important role in SNP detection in metagenomes. The sequence must be long enough to locate the SNPs (e.g. gene and loci) in a unambiguous manner. This challenge is compounded in metagenomics variant detection, where the majority of bacteria is unculturable and unlikely to contain a reference genome. Therefore, we use the simulated data to evaluate the accuracy of LueVari and competing methods, as well as, to compare the relative length of the sequences outputted by the methods.

5.1.1 Accuracy of SNP detection.

We simulated four metagenomics datasets using BEAR, a metagenomics read simulator [16], in a manner that imitates 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. [14]. We varied the copy number and number of paired-end reads for each dataset. Hence, in order to simulate a dataset with copy number x and y number of paired-end reads, we performed the following steps: (1) we selected 400 AMR genes from the MEGARes data at random without replacement, (2) we made x copies of each gene, (3) we added two copies of the *E. coli* K-12 MG 1655 reference genome, and two copies of the salmonella enterica subspecies I, serovar Typhimurium (S. typhimurium) reference genome, and lastly, (4) we simulated y paired-end reads from this resulting set of sequences. We used a 1% error rate for both these simulations. We note that all paired-end datasets contained 150 bp reads and 200 bp insert size. For this experiment, we simulated 270,598, 527,913, 1,110,150 and 2,110,753 paired-end short reads with 3, 5, 9, and 16 average copy number of the AMR genes, respectively. In brief, we call these datasets 270K, 500K, 1M and 2M. Please see the Supplement (Table A.7) for the results of comparison of sensitivity and precision of the reference free tools on datasets with same number of reads and varying number of gene copies.

We compared LueVari against DiscoSNP (DiscoSNP++) [46] and Bubbleparse [22], and note that the most recent version of these was used. Although MaryGold [37] and Bambus2 [18] are comparable methods, they are not available for current sequencing technologies. We also show the comparison of LueVari to SAMtools [24] and GATK [31] in the supplement (see Table A.4 and Table A.5) but note that these are reference-based methods. We used the MEGARes database as the reference. For example, we witnessed that GATK filtered 51.34%, 57.35%, 64.91% and 71.01% of reads in the 270K, 500K, 1M and 2M datasets, respectively, and thus, had low sensitivity for these datasets.

We summarize the results of this experiment for the reference-free methods in Table 1 and Table 2. We calculated the sensitivity and precision, based on the alignment of outputted sequences to the MEGARes database. LueVari and DiscoSNP are able to remove k-mers that have low multiplicity prior to SNP calling. Therefore, we ran DiscoSNP with two different thresholds: with the default setting (which is 3), and with the setting that achieve the highest performance (which is 0). For comparison, we ran LueVari with identical settings. We can see, with a slight penalty on precision and time, LueVari(0) has the highest sensitivity. Further, filtering the low abundant k-mers increased the precision of both LueVari and DiscoSNP; however, the sensitivity of DiscoSNP dropped remarkably (e.g. from 90.1% to 57.5%) whereas LueVari retained a high sensitivity (greater than 90% for all samples containing more than 500K reads). LueVari(3) had the highest sensitivity for three of the four datasets–with the one exception being the 270K dataset in which the sensitivity of LueVari is $\tilde{2}$% less than Bubbleparse but has a precision that is 9.8% higher than Bubbleparse. We report the memory and time usage (CPU time) of all the methods in Table 2. All methods required less than 40 minutes and 12 GB of RAM on these datasets. DiscoSNP was the most efficient, yet the sensitivity was significantly lower than LueVari.

One important feature of LueVari is the ability to correctly reconstruct the sequences containing the SNPs. As shown in Table 1, when we only considered sequences that have at least 200 bp, we see that the sensitivity of the competing methods dropped dramatically. LueVari the sensitively of LueVari remained unchanged. This is an important feature since in metagenome application, a reference genome is frequently unknown. In this application, longer sequences are needed in order to unambiguously compare the SNP profiles between samples.

	270K		500K		1M		2M	
	Sensitivity	Precision	Sensitivity	Precision	Sensitivity	Precision	Sensitivity	Precision
LueVari(0)	95.6 **(95.6)**	11.2	96.2 **(96.2)**	12.5	98.5 **(98.5)**	5.6	97.5 **(97.5)**	6.5
DiscoSNP(0)	81.7 **(59.1)**	14.1	91.2 **(63.7)**	14.2	90.1 **(56.1)**	11	86.8 **(72.5)**	12.5
LueVari(3)	73 **(73)**	17.9	90.6 **(90.6)**	16	94.6 **(94.6)**	7.2	95 **(95)**	6.3
DiscoSNP(3)	32.8 **(0)**	26.7	67.5 **(0)**	8.4	57.5 **(0)**	15.0	55.6 **(0)**	15
Bubbleparse	75.18 **(43)**	8.1	85 **(67)**	6.5	85.6 **(46.2)**	4.4	85.6 **(49.4)**	5.1

Table 1. In this table, we report the sensitivity and precision (reported as percentage) of all methods on the simulated datasets. We note the 270K, 500K, 1M and 2M datasets have 270,598, 527,913, 1,110,150 and 2,110,753 paired-end simulated reads, respectively. k=32 for all experiments except for Bubbleparse in which k=31 (k can not be even). We report in brackets the the sensitivity when interest is restricted sequences that have length greater than 200 bp.

	270K		500K		1M		2M	
	Time	Memory	Time	Memory	Time	Memory	Time	Memory
LueVari(0)	0:10:22	1.09	0:12:17	1.12	0:26:23	1.39	0:39:12	1.63
DiscoSNP(0)	0:00:44	0.153	0:00:55	0.152	0:01:15	0.94	0:01:37	0.74
LueVari(3)	0:04:17	0.398	0:11:34	0.878	0:18:39	1.35	0:31:48	1.62
DiscoSNP(3)	0:00:16	0.152	0:00:30	0.152	0:00:48	0.152	0:01:22	1.43
Bubbleparse	0:01:45	10.92	0:01:43	11.31	0:02:38	11.42	0:05:08	11.44

Table 2. In this table, we give the performance results on the simulated datasets. 270K, 500K, 1M and 2M with 270,598, 527,913, 1,110,150 and 2,110,753 paired-end simulated reads. We report the peak memory in gigabytes (GB), and the running time as hh:mm:ss. k=32 for all experiments except for Bubbleparse in which k=31 (k can not be even).

	low-0.00125		high-0.00125		low-0.005		high-0.005	
	CRG	GF	CRG	GF	CRG	GF	CRG	GF
LueVari	90.19	99.19	100	100	61.76	93.49	83.82	100
DiscoSNP	56.86	92.38	58.82	93.01	14.7	62.03	17.6	71.09
Bubbleparse	0	49.51	0	68.48	0	39.59	0	41.76

Table 3. We give the metrics that describe the length of the sequences outputted by LueVari and competing methods. The low-0.00125 and high-0.00125 datasets contain a total number of 16 AMR genes with 0.00125 SNP rate. The low-0.05 and high-0.05 datasets contain a total number of 10 AMR genes with 0.05 SNP rate. k=32 for all experiments except for Bubbleparse in which k=31 (k can not be even).

5.1.2 Comparison of sequence lengths.
We evaluate the ability of LueVari, DiscoSNP (DiscoSNP++) [46], and Bubbleparse [22] to detect SNPs in a unambiguous manner. To perform this comparison, we simulated four datasets using BEAR in an identical manner as described above, varying the SNP rate and copy number. First, we constructed two datasets with 258,180 and 2,991,107 paired-end sequence reads with mean copy number of 15 (range [10, 20]) and mean copy number of 30 (range [25, 35]), respectively. Here, we kept the SNP rate constant at 0.00125, and selected 16 AMR genes to be in the set. In addition, we simulated reads from *E coli* and salmonella in the manner that was described above. For simplicity we call these samples low-0.00125 and high-0.00125. Next, we simulated two additional datasets with 245,979 and 2,844,899 paired-end reads with mean coverage 15 (range [10, 20]) and 30 (range [25, 35]), respectively. Here, we set the SNP rate to 0.005, selected 10 AMR genes, and as in the previous experiment, included reads simulated from *E coli* and salmonella. We refer to these datasets as low-0.005 and high-0.005.

By default DiscoSNP output sequences of length $2k-1$ where k is the value used to construct the de Bruijn graph, and the recommended length of sequences for Bubbleparse is 200 bp. Thus, we changed the parameters of DiscoSNP and Bubbleparse to output the longest possible sequences containing the identified SNPs. We ran DiscoSNP with `-b2`, and `-T` to ensure that it does not filter branching bubbles (bulges), and extends bubbles to unitigs/contigs. We ran Bubbleparse with `-w 2,4000` to extract sequences around 4,000 nodes.

We report the results in Table 3 and refer the reader to the supplement for additional details about the results (Figure A.3 and Figure A.4). We define gene fraction to be the percentage of base pairs in a gene that are covered by the outputted sequence containing the identified SNP. Thus, we report the *mean gene fraction* (denoted as GF), which is mean of the gene fraction of all sequences (containing an identified SNP) outputted by a method. In addition, we report the percentage of genes that were *correctly recreated* (denoted as CRG), where we define a correctly recreated gene as one where the corresponding outputted sequence has 100% GF. If there are multiple unique SNP profiles for a single gene then this metric counts each profile as an individual gene that should be reconstructed. For example, if we have a dataset with a gene that has 8 different SNP profiles then there should be 8 genes outputted with 100% gene fraction—one for each unique SNP profile.

We see that LueVari identifies the largest mean gene fraction. Hence, Bubbleparse, DiscoSNP and LueVari had 39.6% and 68.5%, 62% and 93%, and 93.5% and 100% as minimum and maximum GF, respectively. Furthermore, Bubbleparse, DiscoSNP and LueVari had a CRG as 0%, between 14.7% and 58.82%, and between 61.76% and 100%, respectively. These results reflect the algorithmic difference between the methods. In case of embedded bulges (where there exists a bulge(s) within bulge), LueVari reports all possible branching paths within a bulge, leading to a higher CRG. Whereas, competing methods only consider a single path within a bulge. This reflects the results in Table 3; the performance of DiscoSNP and Bubbleparse was degraded with a higher SNP rate which causes greater branching within the de Bruijn graph. LueVari was the only method that consistently had high mean gene fraction (greater than 93%), regardless of the coverage and SNP rate. This is an important feature since the resistome (and microbiome) consume a small amount of the biological sample and thus, will have very low coverage and varied polymorphism rate [38, 14] . Lastly, we note that although the gene fraction of DiscoSNP was relatively high for two of the four datasets we considered (e.g. 92%

and 93% for low-0.00125 and high-0.00125, respectively) the CRG was low for all samples, ranging from 15% to 59%.

5.2 Results on Metagenomic Data from Food Production

We demonstrate the ability of LueVari to analyze real shotgun metagenomic data that were sequenced on an Illumina HiSeq 2500 system. The samples were selected across a 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 evolve in the presence of different food production interventions that aim to reduce pathogen load [38]. We ran LueVari on method on the shotgun metagenomic datasets, first filtering for eukaryote species (hen bovine and human DNA) and then filtering for k-mers that have low multiplicty, We report the number of the reads, distinct k-mers, detected bulges, total time of the pipeline, time for constructing the read-color matrix, time for the traversing, peak memory and size of the read-color matrix in Table A.6. As we can see in Table A.6, the parameters affect the traversal time are color matrix size, number of distinct k-mers, and number of bulges. For example, the largest traversing time belongs to sample 3, with largest number of bulges (23,782), largest color matrix (5.2 GB) and second greatest number of distinct k-mers (40,759,656). The size of color matrix depends on number of reads, number of distinct k-mers (size of de Bruijn graph) and total number of k-mers, which indicates the sparseness of the matrix (this value is not reported in the table). We emphasize that both the number of reads and number of k-mers of sample 1 are greater than those of sample 3, and still the color matrix of sample 3 is larger, which is due to a greater total number of k-mers on sample 3. In other words, the color matrix of sample 3 is more condensed. As one can see in Table A.6, with less than 21 GB of memory, and 6 GB of disk space, LueVari can scale for large datasets (more than 55 million read) in less than 19 hours.

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*, pp. 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]J. Catchen et al. building and genotyping loci de novo from short-read sequences. *Nature Biotech*, 31(5):642–646, 2011.

[7]R. Chikhi and G. Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Alg Mol Bio*, 8(22), 2012.

[8]T. C. Conway and A. J. Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479âŁ“486, 2011.

[9]B. E Dutilh et al. Reference-independent comparative metagenomics using cross-assembly: crAss. *Bioinformatics*, 28(24):3225–3231, 2012.

[10]P. Elias. Efficient storage and retrieval by content and address of static files. *J of ACM*, 21(2):246–260, 1974.

[11]J.M. Eppley et al. Strainer: software for analysis of population variation in community genomic datasets. *BMC Bioinformatics*, 8(1):398, 2007.

[12]S.N. Gardner et al. SNP detection and phylogenetic analysis of genomes without genome alignment or reference genome. *Bioinformatics*, 31(17):2877–2878, 2015.

[13]J.S. Ghurye et al. Metagenomic Assembly: Overview, Challenges and Applications. *Yale J Biol Med*, 89(3):353–362, 2016.

[14]M.K. Gibson et al. Improved annotation of antibiotic resistance determinants reveals microbial resistomes cluster by ecology. *ISME*, 9(1):207–216, 2014.

[15]Z. Iqbal et al. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature Genetics*, 44(2):226–232, 2012.

[16]S. Johnson et al. A better sequence-read simulator program for metagenomics. *BMC Bioinformatics*, 15(Suppl 9):S14, 2014.

[17]Nordström KJV. et al. Mutation identification by direct comparison of whole-genome sequencing data from mutant and wild-type individuals using k-mers . *Nature Biotech*, 31:325–330, 2013.

[18]S. Koren et al. Bambus 2: scaffolding metagenomes. *Bioinformatics*, 27(21):2964–2971, 2011.

[19]V. Kuleshov et al. Synthetic long-read sequencing reveals intraspecies diversity in the human microbiome. *Nature Biotech*, 34(1):64–69, 2016.

[20]S. Lakin et al. MEGARes: an antimicrobial resistance database for high throughput sequencing. *Nucleic Acids Res*, 45(D1):D574–D580, 2017.

[21]B. Langmead et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10, 2008.

[22]R.M. Leggett et al. Identifying and classifying trait linked polymorphisms in non-reference species by walking coloured de Bruijn graphs. *PLOS ONE*, 8:60058–10, 2013.

[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]C. Luo et al. ConStrains identifies microbial strains in metagenomic datasets. *Nature Biotechnology*, 33(10):1045–1052, 2015.

[27]R. Luo et al. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, 1(1):1, 2012.

[28]N. Maillet et al. Compareads: comparing huge metagenomic experiments. *BMC Bioinformatics*, 13(19):1, 2012.

[29]N. Maillet et al. COMMET: comparing and combining multiple metagenomic datasets. In *In Proc of IEEE BIBM*, pp. 94–98, 2014.

[30]F.R. Mario. *On the number of bits required to implement an associative memory*. Massachusetts Institute of Technology, Project MAC, 1971.

[31]A. McKenna et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res*, 20:1297–303, 2010.

[32]M.D. Muggli et al. Succinct colored de Bruijn graphs. *Bioinformatics*, p. 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 Research*, 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*, p. 031757, 2015.

[36]S.M. Nicholls et al. Advances in the recovery of haplotypes from the metagenome. *bioRxiv*, p. 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]D. Okanohara and K/ Sadakane. Practical entropy-compressed rank/select dictionary. In *Proc of ALENEX*, pp. 60–70, 2007.

[40]M.C.F. Prosperi and M. Salemi. QuRe: software for viral quasispecies reconstruction from next-generation sequencing data. *Bioinformatics*, 28(1):132–133, 2012.

[41]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.

[42]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.

[43]J.T. Simpson and R. Durbin. Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, 26(12):i367–i373, 2010.

[44]E.J. Stewart. Growing unculturable bacteria. *J Bacter*, 194:4151–4160, 2012.

[45]V.I. Ulyantsev et al. MetaFast: fast reference-free graph-based comparison of shotgun metagenomic data. *Bioinformatics*, 32(18):2760–7, 2016.

[46]R. Uricaru et al. Reference-free detection of isolated SNPs. *Nucleic Acids Research*, 43(2):e11, 2015.

[47]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.

[48]C. Ye et al. Exploiting sparseness in de novo genome assembly. *BMC Bioinformatics*, Suppl 6:S1, 2012.

[49]R. Younesi and MacLean. D. Using 2k+2 bubble searches to find single nucleotide polymorphisms in k-mer graphs . *Bioinformatics*, 1:171–182, 2015.

[50]O. Zagordi et al. ShoRAH: estimating the genetic diversity of a mixed sample from next-generation sequencing data. *BMC Bioinformatics*, 12(1):1, 2011.

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

A.1 Supplement

A.1.1 Illustration of the Read Colored De Bruijn Graph

In Figure A.1 we illustrate concepts related to the read colored de Bruijn graph, and their role in resolving cycles.

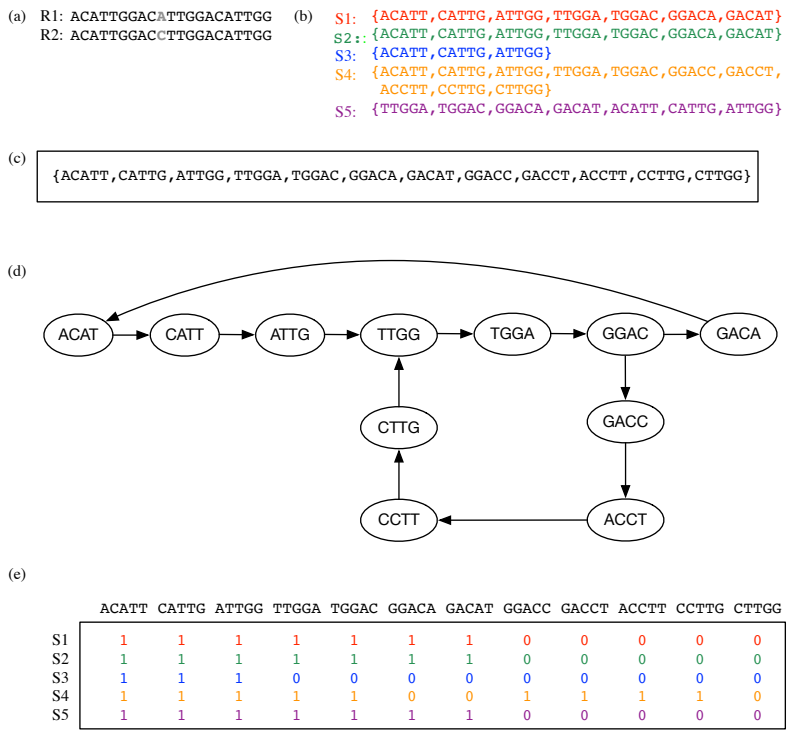

Fig. A.1. In(a) we illustration of two reads R1 and R2 representing a SNP (shown in grey). In (b), we show the sub-reads of R1 and R2; S1, S2, and S3 originate from R1, and S4 and S5 originate from R2. In (c), we give the k-mers that were constructed from R1 and R2. In (d), we show the de Bruijn graph constructed from the set of k-mers. And lastly, in (e), we show the color matrix constructed from the sub-reads and de Bruijn graph. If we traverse the read colored de Bruijn graph in a color coherent manner then `ACATTGGACATTGGACATTGG` and `ACATTGGACCTTGGACATTGG` are recovered. We note that we show the transpose of color matrix to save space.

A.1.2 Gene Fraction Histogram for LueVari

In Figure A.2 we give a histogram of the gene fraction of the sequences outputted by LueVari.

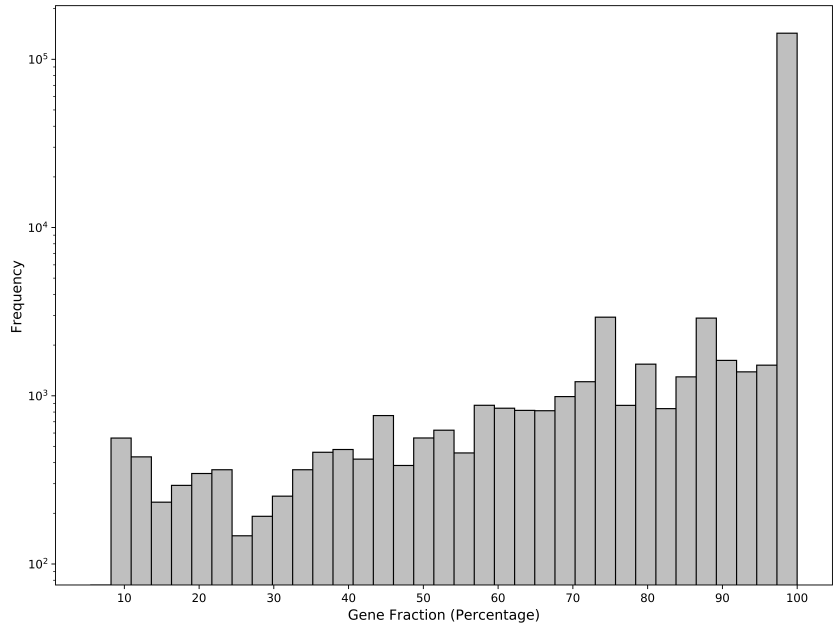

Fig. A.2. In this figure, we give a histogram of the gene fraction of the sequences outputted by LueVari. The y-axis gives the frequency in log scale. The x-axis gives the percentage of the gene that was covered by the sequence containing the detected SNPs. We note that this illustrates one of the main benefits of LueVari, which is that it allows the location of SNPs to be identified without disambiguity.

A.1.3 Comparison of Gene fractions

We compared LueVari gene fraction with DiscoSNP and Bubblepars–two other reference free SNP callers, in four experiments. low-0.00125 and high-0.00125 refer to simulated metagenomic read with SNP rate of 0.00125 and mean coverage of 15 and 30 respectively. Accordingly low-0.005 and high-0.005 refer to simulated metagenomic read with SNP rate of 0.005 and mean coverage of 15 and 30 respectively. As one can see mean gene fraction of LueVari is the highest among all tools. See graphs A.3 and A.4.

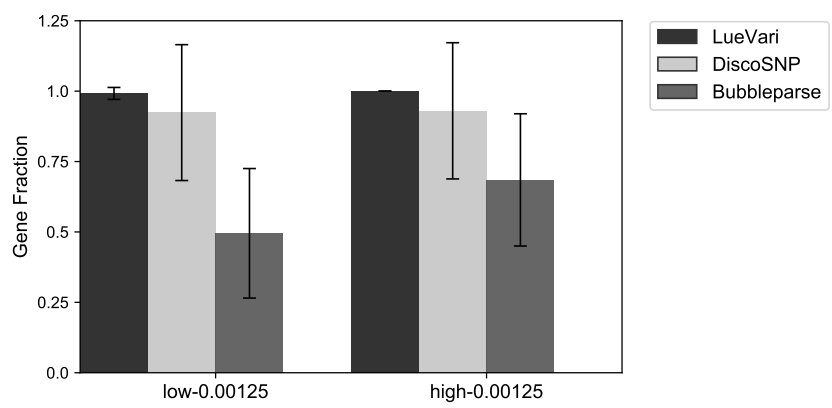

Fig. A.3. Mean gene fraction of three reference free tools on low-0.00125 and high-0.00125.

A.1.4 Comparison Between Reference-Guided Methods

We used MEGARes dataset as the reference for GATK and SAMtools since they are reference-guided methods, this is an added advantage to the other methods that do not have a reference. Tables A.4 and A.5

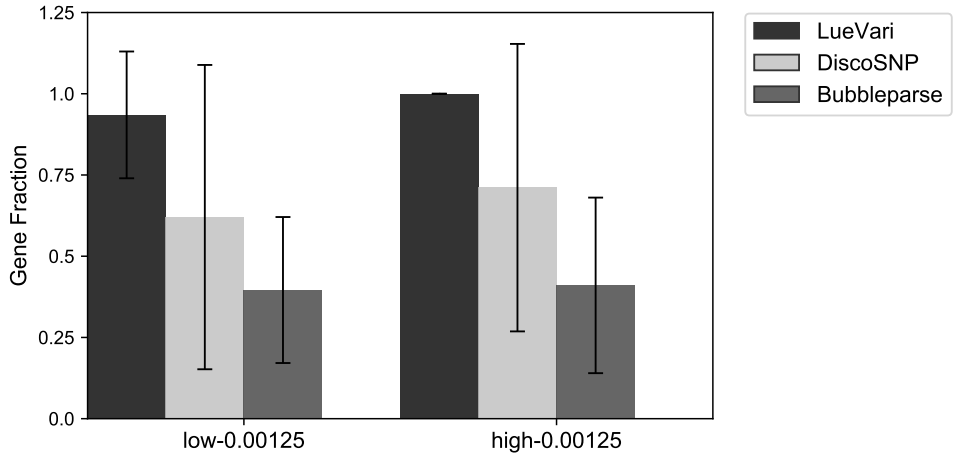

Fig. A.4. Mean gene fraction of three reference free tools on low-0.005 and high-0.005.

	270K		500K		1M		2M	
	Sensitivity	Precision	Sensitivity	Precision	Sensitivity	Precision	Sensitivity	Precision
LueVari(0)	95.6	11.2	96.2	12.5	98.4	5.6	97.5	6.5
DiscoSNP(0)	81.7	14.1	91.2	14.2	90.1	11	86.8	12.5
LueVari(3)	73	17.9	90.6	16	94.6	7.2	95	6.3
DiscoSNP(3)	32.8	26.7	67.5	8.4	57.5	15	55.6	15
Bubbleparse	75.2	8.1	85	6.5	85.6	4.4	85.6	5.1
GATK	6.6	100	6.9	61.1	2.3	37.5	2.5	33.3
SAMtools	83.2	23.8	93.7	21.3	81	7.8	92.5	7.5

Table A.4. We show the accuracy of LueVari, reference-free SNP callers, and reference-guided SNP callers. We note the 270K, 500K, 1M and 2M datasets have 270,598, 527,913, 1,110,150 and 2,110,753 paired-end simulated reads, respectively. We report the sensitivity and precision as percentage. k=32 for all experiments except for Bubbleparse in which k=31 (k can not be even)

	270K		500K		1M		2M	
	Time	Memory	Time	Memory	Time	Memory	Time	Memory
LueVari(0)	0:10:22	1.09	0:12:17	1.12	0:26:23	1.39	0:39:12	1.63
DiscoSNP(0)	0:00:44	0.15	0:00:55	0.15	0:01:15	0.94	0:01:37	0.74
LueVari(3)	0:04:17	0.40	0:11:34	0.88	0:18:39	1.35	0:31:48	1.62
DiscoSNP(3)	0:00:16	0.15	0:00:30	0.152	0:00:48	0.15	0:01:22	1.43
Bubbleparse	0:01:45	10.92	0:01:43	11.31	0:02:38	11.42	0:05:08	11.44
GATK	0:02:22	1.08	0:02:47	2.18	0:02:56	3.2	0:05:08	2.18
SAMtools	0:00:18	0.12	0:00:29	0.12	0:04:16	0.725	0:08:04	1.11

Table A.5. We illustrate the comparison between all competing methods. 270K, 500K, 1M and 2M with 270,598, 527,913, 1,110,150 and 2,110,753 paired-end simulated reads. We report the peak memory in gigabytes (GB), and the running time as hh:mm:ss. k=32 for all experiments except for Bubbleparse in which k=31 (k can not be even)

A.1.5 Results of LueVari on real datasets selected across a beef production system

We ran LueVari to identify SNPs in AMR genes in 34 samples taken from a food production facility that had previously-identified AMR genes. 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.

ID	Type	Location	Reads	k-mers	Bulges	Traverse Time	Color Time	Total Time	Color Matrix	Memory
1	Fecal	Arrival	55,242,003	42,634,941	10,279	03:22:32	00:07:50	14:27:31	4,668.35	17.95
2	Fecal	Arrival	44,035,851	6,398,385	3,947	00:24:19	00:06:51	05:01:57	736.05	14.19
3	Fecal	Arrival	52,833,978	40,759,656	23,782	07:14:10	00:10:12	18:26:17	5,200.4	20.25
4	Fecal	Arrival	54,930,129	27,250,005	6,592	01:33:39	00:10:59	08:58:49	2,818.48	17.62
5	Fecal	Arrival	54,631,823	33,535,641	7,367	02:22:51	00:10:04	11:22:40	3,323.67	17.74
6	Soil	Arrival	40,026,007	11,646,311	2,233	00:31:27	00:06:05	05:42:52	1,229.17	12.65
9	Soil	Arrival	51,173,157	11,920,881	6,388	01:15:53	00:10:06	08:01:59	1,336.16	16.64
22	Fecal	Arrival	57,094,174	24,601,684	14,745	03:37:34	00:12:30	12:44:03	2,876.48	18.48
23	Fecal	Arrival	22,698,431	6,947,493	998	00:06:47	00:03:02	02:51:18	614.9	9.08
24	Soil	Arrival	41,494,968	25,299,886	5,844	01:13:50	00:49:28	08:43:09	2,524.29	13.41
25	Soil	Arrival	58,579,293	20,844,208	6,622	01:28:27	00:10:20	09:24:18	2,246.48	19.05
26	Fecal	Arrival	36,803,397	17,115,596	2,708	00:30:12	00:03:38	04:49:38	1,549.48	11.19
27	Soil	Arrival	44189400	30,657,274	6,005	01:37:29	00:07:08	10:09:55	3,184.84	14.38
33	Fecal	Holding	51,103,450	11,022,596	4,222	00:42:28	00:12:56	07:18:44	1,197.94	16.61
34	Fecal	Exit	53,158,172	12,713,564	3,915	00:46:22	00:36:35	07:47:07	1,389.15	17.32
35	Soil	Arrival	44,192,866	5,816,523	5,642	01:14:18	00:16:34	07:13:20	760.72	14.31
50	Fecal	Exit	39,463,395	13,879,127	3,726	00:30:10	00:10:11	05:01:02	1,524.37	12.94
51	Soil	Exit	53,502,945	12,476,454	4,896	00:56:36	00:13:20	07:44:54	1,435.25	17.45
82	Water	Holding	36,501,759	3,550,203	1,534	00:11:28	00:17:22	03:52:40	416.63	11.18
83	Fecal	Holding	16,511,213	2,472,929	777	00:03:33	00:02:28	01:26:51	235.36	6.82
85	Fecal	Exit	16,954,069	2,608,952	650	00:02:48	00:03:24	01:31:11	252.36	7.05
86	Fecal	Holding	21,417,145	5,396,458	1,470	00:06:36	00:07:19	02:00:17	566.00	7.73
87	Fecal	Holding	18,706,201	1,251,940	818	00:03:05	00:01:47	01:34:16	146.89	7.67
88	Fecal	Holding	17,169,647	1,055,167	713	00:02:23	00:03:23	01:18:48	140.81	7.27
89	Fecal	Exit	12,844,115	807,467	311	00:00:56	00:00:34	00:55:00	90.52	2.90
90	Fecal	Holding	16,433,171	743,280	376	00:01:46	00:00:54	01:05:00	91.93	6.84
91	Fecal	Holding	17,206,031	1,442,855	1,011	00:03:55	00:05:46	01:30:28	178.70	7.30
92	Fecal	Exit	11,136,889	688,444	375	00:01:06	00:05:56	00:56:04	94.39	2.56
102	Soil	Exit	41,320,307	9,643,208	2,334	00:24:38	00:33:41	06:49:54	1,098.3	13.40
105	Soil	Exit	18,258,328	4,171,382	655	00:04:00	00:02:23	01:31:46	390.56	7.09
106	Fecal	Exit	21,703,330	1,187,703	812	00:03:18	00:04:27	01:46:28	152.21	6.96
107	Fecal	Exit	24,090,178	1,462,679	821	00:04:39	00:32:45	03:15:21	198.82	5.98
108	Fecal	Exit	36,055,457	3,913,005	2,857	00:15:56	00:04:34	03:13:13	466.36	11.17
120	Sponges	Truck	55,889,892	14,525,103	9,287	01:52:21	01:35:31	11:22:03	1,813.58	17.97

Table A.6. Performance of LueVari on real data samples of Noyes et al. [38]. ID, Type and the Location that each sample collected from is mentioned in first three columns. Number of Reads in each sample, number of distinct k-mers with frequency greater than or equal to 12, and number of found Bulges are mentioned in columns four to six. Time for Traversing the graph, time for construction of Color Matrix and Total Time are reported in columns seven to nine in format of hh:mm:ss. Size of the Color Matrix on disc is indicated in tenth column (MB), and on last column peak Memory for the whole pipeline is reported (GB).k=32 for all experiments except for Bubbleparse in which k=31 (k can not be even)

A.1.6 Comparison of sensitivity and precision on simulated datasets with same number of reads and different copy number of the genes

In this set of experiments we simulated three datasets (the same way as what was explained in section 5.1.1), each with 300,000 paired-end reads and varying copy number of the genes. The copy numbers of the genes in these datasets are two, five and ten. We show them in table as 2, 5 and 10 respectively. As one can see in table A.7, on all experiments, LueVari has the highest sensitivity ranging from 94% to 100%.

	2			5			10		
	LueVari	DiscoSNP	Bubbleparse	LueVari	DiscoSNP	Bubbleparse	LueVari	DiscoSNP	Bubbleparse
Sensitivity (%)	100	50	72	100	57.14	76.19	94	63.15	52.63
Precision (%)	2.7	1.7	2.1	2.5	1.8	2	2.2	1.9	1.3

Table A.7. We illustrate the comparison between competing methods on datasets 2, 5 and 10 each with 300,000 paired end reads and the gene copy number of 2, 5 and 10 respectively. k = 32 for all experiments except for Bubbleparse in which k=31 (k can not be even). In all experiments all tools were run with the setting that achieve the highest performance (the threshold for filtering weak k-mers is 0).