Bioinformatics, 34(21), 2018, 3601–3608
doi: 10.1093/bioinformatics/bty387
Advance Access Publication Date: 14 May 2018
Original Paper

OXFORD

Genome analysis

Indexed variation graphs for efficient and accurate resistome profiling

Will P. M. Rowe[1,2,*] and Martyn D. Winn[2]

[1]Institute of Integrative Biology, The University of Liverpool, Liverpool L69 7ZB, UK and [2]Scientific Computing Department, The Hartree Centre, STFC Daresbury Laboratory, Warrington WA4 4AD, UK

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on March 2, 2018; revised on April 30, 2018; editorial decision on May 2, 2018; accepted on May 10, 2018

Abstract

Motivation: Antimicrobial resistance (AMR) remains a major threat to global health. Profiling the collective AMR genes within a metagenome (the 'resistome') facilitates greater understanding of AMR gene diversity and dynamics. In turn, this can allow for gene surveillance, individualized treatment of bacterial infections and more sustainable use of antimicrobials. However, resistome profiling can be complicated by high similarity between reference genes, as well as the sheer volume of sequencing data and the complexity of analysis workflows. We have developed an efficient and accurate method for resistome profiling that addresses these complications and improves upon currently available tools.

Results: Our method combines a variation graph representation of gene sets with a locality-sensitive hashing Forest indexing scheme to allow for fast classification of metagenomic sequence reads using similarity-search queries. Subsequent hierarchical local alignment of classified reads against graph traversals enables accurate reconstruction of full-length gene sequences using a scoring scheme. We provide our implementation, graphing Resistance Out Of meTagenomes (GROOT), and show it to be both faster and more accurate than a current reference-dependent tool for resistome profiling. GROOT runs on a laptop and can process a typical 2 gigabyte metagenome in 2 min using a single CPU. Our method is not restricted to resistome profiling and has the potential to improve current metagenomic workflows.

Availability and implementation: GROOT is written in Go and is available at https://github.com/will-rowe/groot (MIT license).

Contact: will.rowe@stfc.ac.uk

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Antimicrobial resistance (AMR) remains a significant global threat to public health, with directly attributable deaths per annum predicted to rise from 700 000 to 10 000 000 by the year 2050 (O'Neill, 2016). This threat is not restricted to humans, with consequences also for animal health, welfare and food production (Bengtsson and Greko, 2014). In an effort to inform policy that can mitigate the spread of AMR, there has been a recent drive to establish surveillance programmes to monitor the prevalence of AMR (Public Health Agency of Canada, 2016; World Health Organization, 2015). These programmes have traditionally used culture or polymerase chain reaction–based surveillance techniques, restricting monitoring to a few key genes or organisms. However, the use of metagenomics for surveilling AMR is gaining traction as it offers a much greater breadth of testing over these traditional techniques (Baquero, 2012; Miller *et al.*, 2013).

The use of metagenomics to determine the antibiotic resistance gene (ARG) content of a microbial community, hereafter referred to as resistome profiling, has been applied in studies of a wide variety of biomes (Auffret *et al.*, 2017; Jalali *et al.*, 2015; Ma *et al.*, 2017;

Munk *et al.*, 2017; Rose *et al.*, 2017; Rowe *et al.*, 2017, 2016; Tao *et al.*, 2016; Yang *et al.*, 2013). Studies such as these utilize several well-maintained ARG databases (Gupta *et al.*, 2014; Jia *et al.*, 2017; Zankari *et al.*, 2012) and ARG-annotation tools (Hunt *et al.*, 2017; Inouye *et al.*, 2014; Rowe *et al.*, 2015; Yang *et al.*, 2016), of which only a few are designed for resistome profiling (Fig. 1). These databases and tools are used with assembled metagenomic contigs, or alternatively with metagenomic sequence reads. In the case of the latter some tools and studies opt to align reads to reference ARGs and report only fully covered references (Rowe *et al.*, 2017), whereas some bin reads using BLAST and similarity/length thresholds (Tao *et al.*, 2016). All of these strategies offer the user a compromise between ease of use, analysis speed and the accurate typing of ARGs (ability to detect ARG type/subtype).

One of the main limitations of existing tools for resistome profiling (or profiling of other gene sets) in metagenomes is that high similarity shared between reference sequences can result in ambiguous alignments, unaligned reads (false negatives), or mis-annotated reads (false positives); thus reducing the accuracy when typing ARGs (Petersen *et al.*, 2017). A solution to this has been to cluster the reference sequences and then use the cluster representative sequences as a target for read alignment (Rowe *et al.*, 2015); however, this results in a loss of information as ARG subtypes will be masked. Likewise, a related approach has been to collapse final annotations of ARG variants into a common ARG annotation, also masking subtypes (Munk *et al.*, 2017). The ability to accurately detect ARG type/subtype, here termed profiling resolution, is important considering that the variation between subtypes of ARGs can result in different phenotypic activity (Bush and Jacoby, 2010).

Recently non-linear data structures, such as variation graphs, have been used to encode reference sequences for applications such as variant calling (Garrison *et al.*, 2017; Paten *et al.*, 2017). In terms of ARG annotation, the Mykrobe predictor tool applies non-linear reference representation in the form of a de Bruijn graph to identify resistance profiles in *Staphylococcus aureus* and *Mycobacterium tuberculosis* isolates (Bradley *et al.*, 2015). Non-linear reference representation reduces redundancy whilst maintaining information that facilitates classification. Variation graphs, which are directed acyclical graphs (DAGs), are offered as a solution to reference bias in population genomics as they represent sequence variation within a population (Garrison *et al.*, 2017). To align a sequence against a variation graph, the traversals within a graph are indexed. Variation graph indexing approaches to date have included hash-map and Burrow-Wheeler transform encoding (Schneeberger *et al.*, 2009; Sirén, 2016). Index design must balance query length, performance and index size in order to deal with the complexity of variation graphs. Efficient indexing is further complicated in the case of multiple graphs.

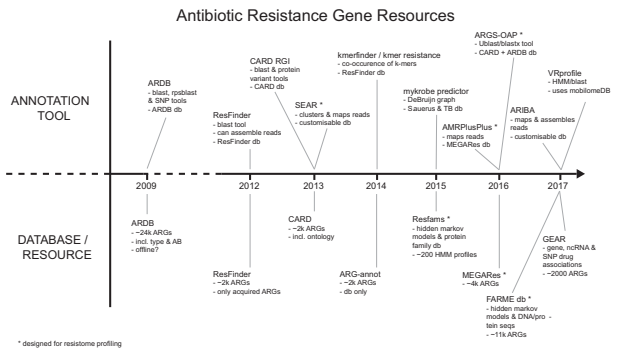

Fig. 1. Summary of ARG annotation tools and databases. This figure shows several published tools and databases that are used to detect ARGs (in sequencing data or assembled contigs) (Bradley *et al.*, 2015; Clausen *et al.*, 2016; Hunt *et al.*, 2017; Jia *et al.*, 2017; Lakin *et al.*, 2017; Li *et al.*, 2017; Liu and Pop, 2009; Rowe *et al.*, 2015; Yang *et al.*, 2016; Zankari *et al.*, 2012). The list is not exhaustive and the tools that are designed specifically for the resistome profiling of metagenomic datasets are highlighted with an asterisk

We propose that variation graph traversals can be indexed using locality-sensitive hashing (LSH), allowing for fast and approximate assignment of a sequence to a specific region of a traversal within one or more graphs. MinHash is a form of LSH that can be used to compress sets into smaller representations, called MinHash signatures. MinHash signatures can be used to estimate the similarity of the original sets independently of the original set size and was first used for duplicate webpage detection (Broder, 1997, 2000). The MinHash technique is increasingly being used in bioinformatics applications for clustering and searching large sequencing datasets (Brown and Irber, 2016; Ondov *et al.*, 2016) and has also recently been applied to read alignment algorithms (Berlin *et al.*, 2015; Popic and Batzoglou, 2017; Quedenfeld and Rahmann, 2017). In applications such as these, the ability to efficiently compare signatures is essential. Additional LSH indexing techniques can therefore be applied to reduce dimensionality and facilitate querying of signatures. One such technique is the LSH Forest indexing scheme that uses multiple prefix trees to store hash tables containing portions of the MinHash signatures and facilitates self-tuning of index parameters, offering a performance improvement over traditional LSH indexes (Bawa *et al.*, 2005).

In this article, we present a method for resistome profiling that utilizes a variation graph representation of ARG databases to reduce ambiguous alignment of metagenomic sequence reads. We store sets of similar ARG reference sequences in variation graphs; collapsing identical sequences whilst retaining unique nodes that allow for accurate typing. By applying an LSH Forest indexing strategy to the variation graph collection that allows for fast and approximate search queries, we show that metagenomic reads can be seeded against candidate graph traversals. Subsequent hierarchical local alignment of reads and scoring enables accurate and efficient resistome profiling. We also provide our implementation, Graphing Resistance Out Of meTagenomes (GROOT), and compare it against ARGS-OAP and AMRPlusPlus, the most recently published tools for resistome profiling from metagenomic data (Lakin *et al.*, 2017; Yang *et al.*, 2016) (Fig. 1).

2 Materials and methods

Here we describe our method to index a collection of ARG reference sequences, align sequence reads using the index and then apply this for resistome profiling of metagenomics samples. We then document our implementation, GROOT, and describe how we evaluated its performance.

2.1 Indexing

The first stage in indexing a collection of ARG reference sequences is to remove redundancy, thus reducing the number of potential alignments to multiple references by collapsing identical sequence regions into a single reference. To do this we employ a DAG (variation graph) representation of ARG reference sequences. By fingerprinting variation graph traversals and using this as an index, we can then perform approximate seeding of sequence reads (Fig. 2) (Supplementary Material, Algorithm 1).

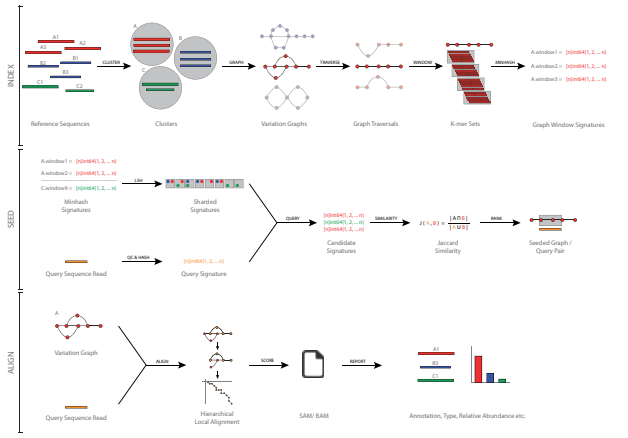

Fig. 2. Overview of our method to index and query ARG variation graphs. This figure shows the indexing (index) and alignment (seed and align) steps of the method. In the alignment step, once a query read has been seeded against a variation graph, a hierarchical local alignment process is performed, and the alignment is scored before being reported

2.1.1 Create variation graphs

To convert a collection of FASTA-formatted ARG reference sequences to a set of variation graphs, all sequences are first clustered based on sequence identity (∼90%) in order to group sequences into distinct sets of similar sequences. Each resulting set of similar sequences can be viewed as a multiple sequence alignment (MSA) that describes the insertions, deletions and polymorphisms in each member of the set, relative to a set representative sequence. The MSA is converted to a variation graph by first using the representative sequence as the graph backbone; each base of the sequence is a node with edges connecting them in series. The other sequences of the MSA are then incorporated; common bases are consolidated as single nodes and all deletions and insertions are added via edges and bubbles.

Each variation graph node contains information such as parent sequence, reference position and encoded base, allowing restriction of graph traversals to known references and the translation of traversals into reference name and sequence location. In the case of sets containing a single sequence, they are still converted to variation graphs but will be a linear series of nodes and only contain one possible traversal. All variation graphs are topologically sorted so that node ordering reflects MSA position prior to fingerprinting graph traversals.

2.1.2 Fingerprint graph traversals

The second indexing stage is to fingerprint the traversals in each variation graph, allowing for fast and approximate matching of query reads to region(s) of a variation graph. To fingerprint a graph, a sliding window of length w is moved across all graph traversals and a MinHash signature is created for each window (Broder, 1997). The windows are typically the same length as the expected query reads. The starting nodes of consecutive windows are spaced by an offset, o, where $o \leq w$.

For each window, the contained nodes are joined to form a sequence (encoded as an array of bytes), which is then decomposed into a set of k-mers of length k, where $k < w$. To then create a MinHash signature (an array of unsigned 64-bit integers) of length s, each k-mer in the set is evaluated in series. A k-mer is hashed to an unsigned 64-bit integer s times, using s distinct hash functions. Each time a k-mer is hashed, the index position in the signature is advanced from 0 and the new hash value is evaluated against the value stored at the current index position; if the stored value is greater than the new value, the stored value is replaced by the new value. The signature index is reset to 0 prior to hashing the next k-mer in the series and the same s hash functions are used. Once all k-mers in a set for a given window have been hashed, the signature for that window is complete; the signature is linked to the graph ID and the window start node before it is stored. A single graph node can have multiple linked signatures if multiple traversals are possible.

2.1.3 Store window signatures

The final indexing stage is to store the window signatures for each variation graph in a data structure that allows for fast and approximate nearest-neighbour queries. To do this, we enlist the LSH Forest self-tuning indexing scheme (Bawa *et al.*, 2005). This indexing scheme, given a query, will give a subset of nearest-neighbour candidates to which the query can be compared, based on the number of hash collisions between query and candidates. In our case, querying the index with a MinHash signature from a sequencing read will return candidate window signatures.

As in a traditional LSH index, two parameters must be tuned in order to maximize the occurrence of collisions between a query and its nearest neighbour: (i) the number of hash functions to encode an item (K) and (ii) the number of hash tables (buckets) to split an item into (L). As we have already hashed the signature during MinHashing, we are essentially splitting the signature uint64 values over a series of buckets. Explicitly, for each of the L buckets we take K uint64 values from the signature (where $K \leq s$) and hash the values again to a single binary string. The original signature of s uint64 values is replaced by a more compact representation of L binary strings.

The challenge with traditional LSH indexing schemes was setting appropriate L and K values. Too small a value for K results in an increased false positive rate due to increased collisions of dissimilar query-neighbour pairs, and a large value for K lowers the collision chances of similar query-neighbour pairs. Therefore, setting $L > 1$ is needed to maximize the occurrence of collisions between similar query-neighbour pairs. To set appropriate K and L values, the distance between query and nearest neighbour is needed, but tuning for one query-neighbour pair can reduce performance for other query-neighbour pairs (Gionis *et al.*, 1999). The LSH Forest indexing scheme addresses the limitations of the traditional LSH indexing by using a unique label of variable length to assign data points to buckets and storing these in a data structure that combines multiple prefix trees, each constructed from hash functions (derived from the MinHash signature).

The LSH Forest data structure is first initialized and tuned using the MinHash signature length and the Jaccard Similarity threshold for reporting query-neighbour matches. To tune the index, multiple combinations of the number of buckets (L) and the number of hash functions (K) are tested (within the bounds of the signature length) for false positive and false negative rates at the specified Jaccard Similarity. L and K are then set according to the lowest error rate possible and a set of L initial hash tables are then created (Supplementary Material, Algorithm 2).

Once the LSH Forest data structure has been initialized, each signature from the variation graph(s) is added to the initial hash tables. Each signature is split into L equally sized chunks of K hash functions. The chunks are hashed to a binary string (little-endian ordering) and stored in the corresponding hash table, with each chunk pointing to the graph and window from which they derive. Once all

signatures have been added, the initial hash tables are transferred into a set of arrays (buckets) and sorted.

2.2 Alignment

To align sequence reads to a variation graph, our method uses an approximate nearest-neighbour search to seed a read against a region of a graph and then employs a hierarchical local alignment to fully align the read.

2.2.1 Nearest-neighbour search

For a given sequence read, a MinHash signature is generated as during indexing, using the same values for k and s, etc. The signature is then queried against the LSH Forest data structure containing the indexed variation graphs (Supplementary Material, Algorithm 3). To run a query, the signature is split into L equally sized chunks and each chunk is hashed to a binary string using the same hash function as during the LSH Forest population (Bawa *et al.*, 2005). Each hashed chunk of the query signature is then queried against the corresponding bucket in the LSH Forest. To improve querying efficiency, each bucket is compressed by constructing a prefix tree (PATRICIA trie) (Morrison, 1968). Binary search is used to search the bucket (in ascending order) and return the smallest index of the bucket where the prefix matches the query chunk; the bucket is then iterated over from this index position, returning graph windows held until the prefix no longer matches the query chunk. Once all chunks of the query sequence have been searched, all the windows (graph IDs and start nodes) are collated and stored as seeds for each read. The nearest-neighbour search is then repeated using the reverse complement of the query read.

2.2.2 Hierarchical local alignment

For a read that has been seeded one or more times, a hierarchical local alignment process is used to align the read to a graph traversal. We assume that most reads do not feature novel variation, so we first try an exact match alignment. We then try an exact match alignment after shuffling the seed n nodes along the graph, followed by a clipped alignment. As soon as the read aligns, no further alignments are tried for that seed. To perform an alignment from a given seed, a recursive depth-first search (DFS) of the seeded graph is performed, beginning at the start node identified during the nearest-neighbour search (Supplementary Material, Algorithm 4). If a match between node and read base is encountered, the position in the read is incremented and the next node in the DFS is checked. When no match is found or the whole read has been iterated over, the DFS of the current traversal is ended.

2.3 Reporting

Reads are classified as ARG-derived if they have successfully aligned to a variation graph. To apply our method to resistome profiling, the classified reads must be evaluated to annotate what gene they derive from and if that entire gene is present in a given sample.

To classify a read alignment, a score is calculated according to the parent information of the nodes of the alignment. That is, if a node was derived from gene X and gene Y, the read would receive a point for each parent. Points are tallied, the top parent(s) in the tally is used to classify the read. If the top parent was not present for every node traversed, the read is ambiguous. If multiple parents score the highest then the read has multiple valid classifications (a multi-mapper).

Once reads have been classified, gene annotations are then made. Classified reads are binned according to the graphs to which they align and then reference information is extracted from the corresponding graph(s). The length of the reference is used to perform a simple pileup of classified reads. Length and coverage thresholds are applied to determine if a gene can be reported as present in the resistome profile.

2.4 Our implementation

We have implemented our method as an easy to use program called GROOT (Fig. 2). GROOT is written in Go (version 1.9) and compiles for a variety of operating systems and architectures.

To create the supplied reference data (used for indexing), ARG sequences were downloaded from the CARD, Resfinder and ARG-annot databases (Gupta *et al.*, 2014; Jia *et al.*, 2017; Zankari *et al.*, 2012) (accessed June 2016). Each database was clustered using the VSEARCH cluster_size command and stored as MSA files (Rognes *et al.*, 2016). The GROOT get command will fetch a specified pre-clustered database, check it and unpack it ready for indexing. Alternatively, a user can provide their own clustered database.

The index command checks MSA files for formatting and discards sequences shorter than the expected read query length. Variation graphs are then built from the MSAs, pruned and topologically sorted. For each graph, a sliding window is moved along each traversal (default length = 100, offset = 1), decomposed to k-mer sets (default $k = 7$) and MinHash signatures are created using the Go implementations of Spooky and Farm hash functions (default signature length = 128, based on XORing the Spooky and Farm hash functions) (Jenkins; Google; Gryski https://github.com/dgryski/go-spooky). An LSH Forest is then initialized, tuned and populated, before being serialized using the Go gob package and written to disk.

The align command loads the index, sets all hashing parameters to match the index and then streams the FASTQ read file(s) (multiple FASTQ files or paired reads can be read but paired-end information is not utilized). MinHash signatures are created for each read and its reverse complement; signatures are then queried against the index. Once seeded, reads are optionally quality trimmed prior to hierarchical local alignment and reporting. All variation graphs that had reads align are saved to disk [in graphical fragment assembly (GFA) format] and can be viewed using Bandage (Wick *et al.*, 2015). All aligned reads are also reported in relation to a linear reference sequence (in BAM format).

The index, align and report subcommands of GROOT all utilize a concurrent pipeline pattern that is driven by the flow of data between structs. This pattern also facilitates the streaming of data from STDIN, as well as from disk, and allows the GROOT commands to be piped together.

2.5 Evaluating performance

The full commands and code used to evaluate the performance of our implementation can be found in the GROOT repository (https://github.com/will-rowe/groot/tree/master/paper). GROOT version 0.7 was used in all experiments (release 0.7, commit b43c32c). For running the accuracy benchmark, simulated FASTQ reads (150 bp read length) were generated from the ARG-annot database using BBMap (Bushnell, 2014; Gupta *et al.*, 2014). An index of the ARG-annot database was created using the GROOT index command (length = 150, all other settings default). Reads were aligned using the GROOT align command with default settings, running on a Linux laptop using 1, 4, and 8 CPUs for each test, respectively.

For running the comparison benchmark, the genomes from the Critical Assessment of Metagenome Interpretation (CAMI) project

were downloaded in FASTA format (Sczyrba *et al.*, 2017). The genomes were screened against the CARD database to mask any ARGs already present in the genomes (Jia *et al.*, 2017). A subset of ARGs were then randomly sampled from the CARD database (v1.1.2) using Bioawk and were combined with the CAMI genomes in a single FASTA file (Jia *et al.*, 2017). The CARD database was selected as ARGs-OAP uses this and it is one of the default databases provided by GROOT (Yang *et al.*, 2016). Sets of metagenomic FASTQ reads (150 bp read length, errors allowed) at varying coverage levels were then generated from the FASTA file using BBMap (Bushnell, 2014). For each set of metagenomics reads, GROOT and ARGs-OAP [stage 1, version 1 (release 1, commit: ed19cd1)] were both run with default parameters (recommended settings) on a LINUX laptop using 1 core (Yang *et al.*, 2016). ARGs-OAP (stage 2, version 2) was completed on the dedicated Galaxy webserver, allowing only full-length, 100% identity read matches (Yang *et al.*, 2016). For comparing GROOT against AMRPlusPlus, reads were simulated (5× coverage) from the CAMI database as above, this time spiked with 10 ARGs from the MegaRes database (version 1.01) (Lakin *et al.*, 2017). Simulated reads were analysed using AMRPlusPlus (default settings, commit: 3086a1f) and GROOT on a Linux cluster, both using eight cores and a reporting threshold of 100%, 99% and 80% gene coverage for each test.

For performing the resistome analysis, we reanalysed the metagenomic data from a recent microbiome study (Winglee *et al.*, 2017). FASTQ reads for all 40 microbiomes were downloaded from the SRA (BioProject: PRJNA349463) and were classified by GROOT using the CARD database (Jia *et al.*, 2017). GROOT was set to report full-length ARG annotations and the variation graph alignments (GFA format) for each annotation were validated by manual inspection using Bandage (Wick *et al.*, 2015). Toxin/antitoxin genes were identified in microbiome samples using GROOT and the TADB database (clustered at 90% sequence identity) (Xie *et al.*, 2018). The full commands used are available in the online GROOT tutorial (https://groot-documentation.readthedocs.io/en/latest/tutorial.html).

3 Results

The results presented here evaluate our implementation of indexed variation graphs for resistome profiling, in terms of both the accuracy of the tool and its performance compared to a recently published resistome profiling tool.

3.1 Accuracy assessment

To assess the accuracy of our implementation, we generated sets of random FASTQ reads from the ARG-annot database, classified the reads with GROOT and then compared the classification to the ARG from which the read derived (Gupta *et al.*, 2014). For six sets of simulated reads, ranging from 100 to 10 000 000 reads, GROOT classified the ARG-derived reads with no recorded false negatives. GROOT took an average of 216.44 wall clock seconds to classify 1 000 000 ARG-derived reads using 8 CPUs. At <1000 reads, runtimes were equivalent regardless of the number of CPUs used, however, as the datasets approached real-world metagenome size (>1 000 000 reads), increasing the number of CPUs resulted in decreased runtime of approximately 2.5-fold per 4 CPUs (Fig. 3). The average RAM occupation recorded during the accuracy assessment was 2.23 GB.

3.2 Performance comparison

To compare the accuracy and speed of our implementation against a recently published resistome profiling tool [ARGs-OAP (Yang *et al.*, 2016)], we used a set of publically available, simulated metagenomes and spiked in full-length ARG sequences. The spiked data was sampled at varying coverage levels and the performance of GROOT and ARGs-OAP was evaluated in terms of the number of ARGs correctly annotated from each sample, as well as the time taken for each tool to process the samples.

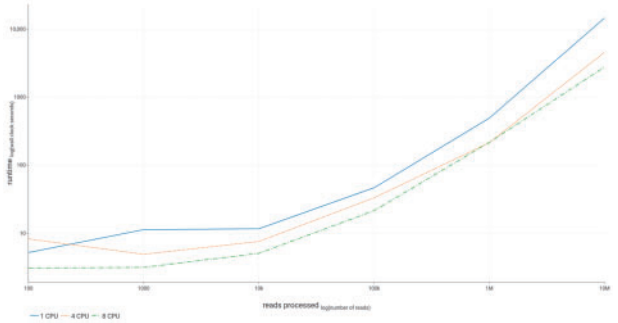

Fig. 3. Runtime performance. This figure shows the runtime performance of our implementation (GROOT) on 1, 4, and 8 CPUs. The data were collected during the accuracy assessment of GROOT, where it classified ARG-derived reads (no false negatives were recorded)

On average GROOT was 6.3 times faster at processing the samples compared to stage 1 of ARGs-OAP. The mean time to process a metagenome with 1× coverage (approximately 1.1 million reads) was 157.87 s for GROOT, compared to 989.15 s for ARGs-OAP (stage 1 only) (Fig. 4A). ARGs-OAP stage 2 took an average of 12 h to run on a dedicated server (not included in Fig. 4A).

In terms of accuracy, GROOT recorded only one false negative across all samples (present in the lowest coverage metagenome), whereas ARGs-OAP (Stages 1 + 2) consistently recorded three false negatives per sample. GROOT had a mean rate of 1.6 false positives per sample, whereas ARGs-OAP had a mean rate of 99.6 false positives per sample. The number of false positives found by GROOT did not increase beyond 10× coverage, compared to the false positive increase observed at every coverage increase by ARGs-OAP (Fig. 4B).

We additionally compared GROOT to AMRPlusPlus. Although GROOT outperformed this software on the basis of runtime (117 s versus 3366 s average run time) and accuracy (Supplementary Results 1), the comparison of GROOT and AMRPlusPlus is not ideal. This is due to AMRPlusPlus being unable to run on a laptop using a single core (as in the above benchmark test) and the fact that AMRPlusPlus is a pipeline that involves a lot of additional data processing, leading to a much longer runtime.

3.3 Resistome analysis

To show the application of our method on real-world data, we reanalysed 40 recently published metagenomes that were derived from the microbiome of rural and urban subjects from the Hunan province of China (Winglee *et al.*, 2017). GROOT made 84777 ARG read classifications across all the samples; the mean number of ARG-classified reads was higher in rural microbiome samples than urban (Fig. 5A). The resistome profiles generated by GROOT identified 11 and 20 ARGs that could be accurately subtyped in the rural and urban microbiome samples respectively (Fig. 5B). The ARG subtypes were also confirmed by inspection of variation graphs; uniform read coverage along graph traversals corresponded to the identified ARGs (see example variation graph for bla-cfxA, Fig. 5C). Finally, to show the utility of GROOT in classifying non-ARG sequences, we used GROOT to classify reads that were derived from toxin genes (Supplementary Fig. S1). We found 11 different full-

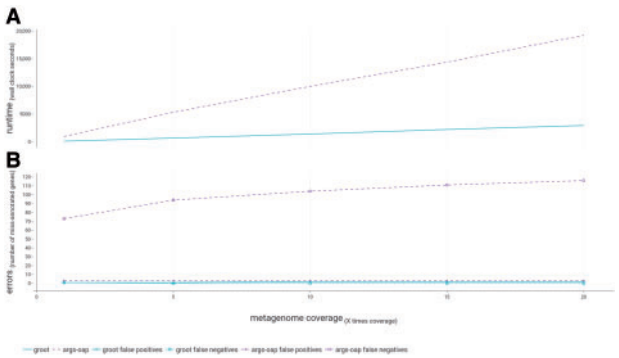

Fig. 4. Benchmarking GROOT and ARGs-OAP. (**A**) The runtime comparison of GROOT and ARGs-OAP when processing metagenomes at varying coverage levels. (**B**) The number of errors (false positives and false negatives) recorded by each program at varying metagenome coverage levels

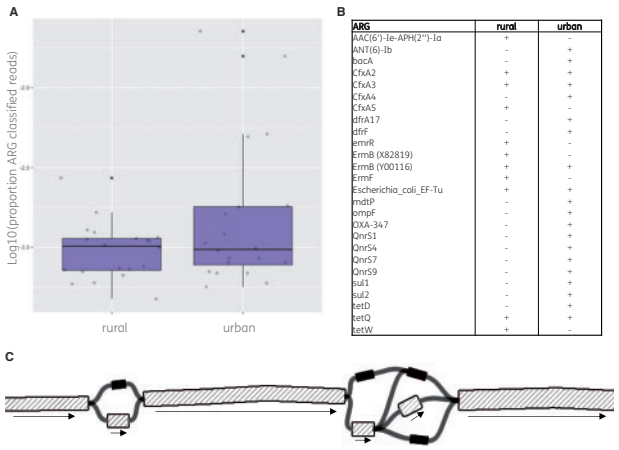

Fig. 5. Resistome analysis using GROOT on 40 human microbiome samples. (**A**) A boxplot comparing ARG-classified reads derived from all rural versus urban subject microbiomes. (**B**) The full length ARGs detected by GROOT during resistome profiling and in which microbiome class they were present. (**C**) A subgraph of the bla-cfxA variation graph used by GROOT in this analysis (the full graph encodes 4 sequences using 10 nodes and 13 edges). Grey shading corresponds to nodes with aligned reads. The bla-cfxA3 traversal is highlighted by the black arrows

length toxin sequences in the microbiomes of urban subjects, whereas no full-length sequences could be identified in the rural samples.

4 Discussion

AMR remains a significant challenge to human and animal health. With the increasing drive for AMR surveillance in order to inform policy and mitigate the spread of resistance, metagenomic studies to identify and monitor ARGs in a wide variety of biomes are becoming commonplace. Sampled biomes range from the human gut, through to water supplies, agricultural land and marine environments. Despite this scale and variety, the question being asked remains simple, what ARGs are present in a given sample? Although simple, resistome profiling still presents a challenge, as illustrated by the many tools available (Fig. 1). Our method improves upon two main issues with these existing tools; the resolution offered and the speed at which a sample can be profiled. Crucially also, as a novel algorithm, it is translatable to identifying other highly related gene sets within metagenomes. Variation graphs can be generated, indexed and searched using any reference set of similar sequences. For example, we have shown that GROOT can be used to annotate toxin genes in microbiome data (see results—resistome analysis). We should note however that our implementation currently restricts the number of graphs that can be generated to around 2000.

In terms of profiling resolution, we are referring to the accuracy of the annotations in a resistome profile. That is, is it possible to annotate the subtype of ARG in a sample (e.g. *bla-SHV-1*) or just the type (e.g. *bla-SHV*)? This resolution is important as different ARG subtypes can provide selective resistance to different antibiotics, for example genes within a single beta lactamase gene class (structural classes, based on amino acid sequence) can confer resistance to different beta lactam antibiotics (Bush and Jacoby, 2010). Therefore, the greater the resolution, the more information can be extracted from a sample. To gain this resolution, we need to both cover an entire reference sequence and be confident in the reads placed. We address this by allowing only exact matches to the whole reference sequence before annotating the gene as present. The counter to this argument for greater resolution is that sequence quality will impact read placement and also, novel ARGs will be missed. Whilst we believe that the main utility of GROOT is its ability to confidently resolve ARGs, we have also added features to allow for these points. Firstly, our implementation has an optional quality trimming algorithm to remove low quality bases prior to read alignment. Secondly, to allow potentially novel ARGs or accommodate low coverage samples, there is an option to relax the scoring system in order to allow non-exact matches or partially covered genes.

Our method offers an improvement in resolution over those that consolidate variant ARGs to a representative sequence (Munk *et al.*, 2017; Rowe *et al.*, 2015), or that have high false positive rates due to allowing partial or inexact matches (Yang *et al.*, 2016). Despite a marked improvement on ARGs-OAP in our benchmark our method did record a false negative and some false positives, likely as a result of introduced sequencing error and the low sample coverage (in the case of the single false negative). These errors could be considered a limitation of the exact local alignment utilized in this method, a more relaxed alignment could be allowed but this would be at the expense of confidence in the ARG annotations.

In terms of speed, our method offers several advantages over other resistome profiling tools. Our method does not require metagenome assembly or the upload of data to remote servers, both of which add significant time to a resistome profiling analysis. The latter requires good bandwidth and the former can require large compute resources; a complex metagenome can take up to 10 h and 500 GB of RAM to assemble (van der Walt *et al.*, 2017). Our implementation can run a typical 2 GB metagenome in 2 min using a single CPU, and scales when run on higher-performance systems. Our benchmark was restricted to a single CPU as ARGs-OAP is limited in the number of CPUs it can use. The benchmark also ignored the time taken during Stage 2 of ARGs-OAP (on the remote server) as this could be variable depending on server load and available bandwidth for upload. Despite this, GROOT still offers a much faster runtime than ARGs-OAP. In addition to runtime, analysis times are also reduced with GROOT due to its ease of use. It runs as a self-contained binary, is packaged with bioconda (Dale *et al.*, 2017) and requires only two commands to run a resistome profiling analysis, offering significant advantage over more complex workflows or those that require upload to remote servers halfway through the analysis (Yang *et al.*, 2016). Our implementation is targeted towards researchers who may not have access to high performance computing and wish to run metagenomics workflows on a laptop, for this reason we elected to compare our tool against ARGs-OAP as it was

one of three published resistome profilers that is targeted for metagenomic workflows (Fig. 1). We also tried using AMRPlusPlus and SEAR in this benchmark but we could not get them to run using our 1-core laptop configuration (Lakin *et al.*, 2017; Rowe *et al.*, 2015). However, as GROOT also scales for use on larger servers, we were able to include a comparison of GROOT and AMRPlusPlus. Although this comparison is still slightly unfair as GROOT is a self-contained resistome profiler and AMRPlusPlus is a pipeline that involves many additional steps, this comparison is useful as it shows GROOT is comparable to an approach using linear reference-based read mapping [as AMRPlusPlus uses BWA to align reads (Li, 2013)].

With the advent of long-read and barcoding protocols for resistome profiling, it is tempting to think that tools for short read resistome profiling may prove less relevant in the near future (Guérillot *et al.*, 2018; Van Der Helm *et al.*, 2017). However, Illumina sequencing currently remains the standard technology for metagenomics applications and historic Illumina datasets will also need to be reanalysed where new sampling is not feasible. In addition, our method can be used in conjunction with other new methods that elucidate gene context in metagenomics samples, allowing for novel insights to be gained from analysis and re-analysis of metagenomic data collections (Olekhnovich *et al.*, 2018).

Our implementation offers efficient and accurate resistome profiling, enabling the identification of full-length ARGs in complex samples. It should be noted that this method is database dependent and is not intended for the identification of novel ARGs. The implementation is easy and quick to run as there is no complicated installation or dependencies, no split local/remote processing, it facilitates streaming of input and is built using Go's concurrency patterns.

5 Conclusions

We present a method for resistome profiling that utilizes a novel index and search strategy to accurately type resistance genes in metagenomic samples. The use of variation graphs yields several advantages over other methods using linear reference sequences. GROOT performed much more quickly than a recent resistome profiling tool, and also recorded few false positives and negatives. Our method is not restricted to resistome profiling and has the potential to improve current metagenomic workflows.

Acknowledgements

Availability and implementation.

The source code for our implementation, as well as the code used to evaluate its performance and plot the manuscript figures, can be found in the GROOT repository (https://github.com/will-rowe/groot) (MIT License. DOI: https://doi.org/10.5281/zenodo.1217889).

Funding

This work was supported in part by the STFC Hartree Centre's Innovation Return on Research programme, funded by the Department for Business, Energy & Industrial Strategy.

Authors' contributions

W.P.M.R. conceived and implemented the method. All authors wrote, read and approved the final manuscript.

Conflict of Interest: none declared.

References

Auffret,M.D. *et al.* (2017) The rumen microbiome as a reservoir of antimicrobial resistance and pathogenicity genes is directly affected by diet in beef cattle. *Microbiome*, **5**, 159.

Baquero,F. (2012) Metagenomic epidemiology: a public health need for the control of antimicrobial resistance. *Clin. Microbiol. Infect.*, **18**, 67–73.

Bawa,M. *et al.* (2005) LSH forest: self-tuning indexes for similarity search. *Proceedings of the 14th International Conference on World Wide Web – WWW '05*, p.651.

Bengtsson,B., and Greko,C. (2014) Antibiotic resistance–consequences for animal health, welfare, and food production. *Ups. J. Med. Sci.*, **119**, 96–102.

Berlin,K. *et al.* (2015) Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat. Biotechnol.*, **33**, 623–630.

Bradley,P. *et al.* (2015) Rapid antibiotic-resistance predictions from genome sequence data for Staphylococcus aureus and Mycobacterium tuberculosis. *Nat. Commun.*, **6**, 10063.

Broder,A.Z. (2000) Identifying and filtering near-duplicate documents. In *Annual Symposium on Combinatorial Pattern Matching*, pp. 1–10.

Broder,A.Z. (1997) On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES (Cat. No.97TB100171)*. IEEE Comput. Soc., pp. 21–29.

Brown,C.T., and Irber,L. (2016) sourmash: a library for MinHash sketching of DNA. *J. Open Source Softw.*, **1**, 27.

Bush,K., and Jacoby,G.A. (2010) Updated functional classification of beta-lactamases. *Antimicrob. Agents Chemother.*, **54**, 969–976.

Bushnell,B. (2014) BBMap: a fast, accurate, splice-aware aligner. In: *9th Annual Genomics of Energy & Environment Meeting*, Walnut Creek, CA, March 17-20, 2014.

Clausen,P.T.L.C. *et al.* (2016) Benchmarking of methods for identification of antimicrobial resistance genes in bacterial whole genome data. *J. Antimicrob. Chemother.*, **71**, 2484–2488.

Dale,R. *et al.* (2017) Bioconda: a sustainable and comprehensive software distribution for the life sciences. *bioRxiv*, 207092.

Garrison,E. *et al.* (2017) Sequence variation aware references and read mapping with vg: the variation graph toolkit. *bioRxiv*, 1–27.

Gionis,A. *et al.* (1999) Similarity search in high dimensions via hashing. *VLDB '99 Proceedings of the 25th International Conference Very Large Data Bases*, **99**, 518–529.

Google. FarmHash. https://github.com/google/farmhash.

Gryski,D. (2014) go-spooky. https://github.com/dgryski/go-spooky.

Guérillot,R. *et al.* (2018) Comprehensive antibiotic-linked mutation assessment by Resistance Mutation Sequencing (RM-seq). *bioRxiv*, 257915.

Gupta,S.K. *et al.* (2014) ARG-ANNOT, a new bioinformatic tool to discover antibiotic resistance genes in bacterial genomes. *Antimicrob. Agents Chemother.*, **58**, 212–220.

Van Der Helm,E. *et al.* (2017) Rapid resistome mapping using nanopore sequencing. *Nucleic Acids Res.*, **45**, gkw1328.

Hunt,M. *et al.* (2017) ARIBA: rapid antimicrobial resistance genotyping directly from sequencing reads. *Microb. Genom.*, **3**, e000131.

Inouye,M. *et al.* (2014) SRST2: rapid genomic surveillance for public health and hospital microbiology labs. *Genome Med.*, **6**, 90.

Jalali,S. *et al.* (2015) Screening currency notes for microbial pathogens and antibiotic resistance genes using a shotgun metagenomic approach. *PLoS One*, **10**, e0128711.

Jenkins,B. SpookyHash: a 128-bit noncryptographic hash. http://burtleburtle.net/bob/hash/spooky.html.

Jia,B. *et al.* (2017) CARD 2017: expansion and model-centric curation of the comprehensive antibiotic resistance database. *Nucleic Acids Res.*, **45**, D566–D573.

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

Li,H. (2013) Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM.

Li,J. *et al.* (2017) VRprofile: gene-cluster-detection-based profiling of virulence and antibiotic resistance traits encoded within genome sequences of pathogenic bacteria. *Brief Bioinform.*, bbw141.

Liu,B., and Pop,M. (2009) ARDB–Antibiotic Resistance Genes Database. *Nucleic Acids Res*., **37**, D443–D447.

Ma,L. *et al*. (2017) Catalogue of antibiotic resistome and host-tracking in drinking water deciphered by a large scale survey. *Microbiome*, **5**, 154.

Miller,R.R. *et al*. (2013) Metagenomics for pathogen detection in public health. *Genome Med*., **5**, 81.

Morrison,D.R. (1968) PATRICIA—practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, **15**, 514–534.

Munk,P. *et al*. (2017) A sampling and metagenomic sequencing-based methodology for monitoring antimicrobial resistance in swine herds. *J. Antimicrob. Chemother*., **72**, 385–392.

O'Neill,J. (2016) *Tackling Drug-Resistant Infections Globally: Final Report and Recommendations. The Review on Antimicrobial Resistance*. London: HM Government and the Wellcome Trust; 2016.

Olekhnovich,E.I. *et al*. (2018) MetaCherchant: analyzing genomic context of antibiotic resistance genes in gut microbiota. *Bioinformatics*, **34**, 434–444.

Ondov,B.D. *et al*. (2016) Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biol*, **17**, 132.

Paten,B. *et al*. (2017) Genome graphs and the evolution of genome inference. *Genome Res*., **27**, 665–676.

Petersen,T.N. *et al*. (2017) MGmapper: reference based mapping and taxonomy annotation of metagenomics sequence reads. *PLoS One*, **12**, e0176469.

Popic,V., and Batzoglou,S. (2017) A hybrid cloud read aligner based on MinHash and kmer voting that preserves privacy. *Nat. Commun*., **8**, 15311.

Public Health Agency of Canada. (2016) Canadian antimicrobial resistance surveillance system – Report 2016. Guelph, Canada.

Quedenfeld,J., and Rahmann,S. (2017) Variant tolerant read mapping using min-hashing. 1–19.

Rognes,T. *et al*. (2016) VSEARCH: a versatile open source tool for metagenomics. *PeerJ*, **4**, e2584.

Rose,G. *et al*. (2017) Antibiotic resistance potential of the healthy preterm infant gut microbiome. *PeerJ*, **5**, e2928.

Rowe,W. *et al*. (2016) Comparative metagenomics reveals a diverse range of antimicrobial resistance genes in effluents entering a river catchment. *Water Sci. Technol*., **73**, 1541–1549.

Rowe,W. *et al*. (2015) Search engine for antimicrobial resistance: a cloud compatible pipeline and web interface for rapidly detecting antimicrobial resistance genes directly from sequence data. *PLoS One*, **10**, e0133492.

Rowe,W.P.M. *et al*. (2017) Overexpression of antibiotic resistance genes in hospital effluents over time. *J. Antimicrob. Chemother*., **72**, 1617–1623.

Schneeberger,K. *et al*. (2009) Simultaneous alignment of short reads against multiple genomes. *Genome Biol*., **10**, R98.

Sczyrba,A. *et al*. (2017) Critical assessment of metagenome interpretation—a benchmark of metagenomics software. *Nat. Methods*, **14**, 1063–1071.

Sirén,J. (2016) Indexing variation graphs. *arXiv*.1604.06605.

Tao,W. *et al*. (2016) High levels of antibiotic resistance genes and their correlations with bacterial community and mobile genetic elements in pharmaceutical wastewater treatment bioreactors. *PLoS One*, **11**, e0156854.

van der Walt,A.J. *et al*. (2017) Assembling metagenomes, one community at a time. *BMC Genom*., **18**, 521.

Wick,R.R. *et al*. (2015) Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, **31**, 3350–3352.

Winglee,K. *et al*. (2017) Recent urbanization in China is correlated with a Westernized microbiome encoding increased virulence and antibiotic resistance genes. *Microbiome*, **5**, 121.

World Health Organization. (2015) *Global Antimicrobial Resistance Surveillance System Manual for Early Implementation Global Antimicrobial Resistance Surveillance System*.

Xie,Y. *et al*. (2018) TADB 2.0: an updated database of bacterial type II toxin-antitoxin loci. *Nucleic Acids Res*., **46**, D749–D753.

Yang,Y. *et al*. (2016) ARGs-OAP: online analysis pipeline for antibiotic resistance genes detection from metagenomic data using an integrated structured ARG-database. *Bioinformatics*, **32**, 2346–2351.

Yang,Y. *et al*. (2013) Exploring variation of antibiotic resistance genes in activated sludge over a four-year period through a metagenomic approach. *Environ. Sci. Technol*., **47**, 10197–10205.

Zankari,E. *et al*. (2012) Identification of acquired antimicrobial resistance genes. *J. Antimicrob. Chemother*., **67**, 2640–2644.