May 21, 2024

MetaAMRSpotter: Automated workflow with shell scripting for uncovering hidden AMR hotspots from metagenomes

DOI
dx.doi.org/10.17504/protocols.io.e6nvw1jyzlmk/v1

Chandrashekar K[1], Anagha S Setlur[1], S Pooja[1], M Purushotham Rao[1], Vidya Niranjan[1]

[1]Department of Biotechnology, RV College of Engineering, Bangalore - 560059, affiliated to Visvesvaraya Technological University, Belagavi - 590018

Vidya Niranjan
R V College of Engineering

DOI: dx.doi.org/10.17504/protocols.io.e6nvw1jyzlmk/v1

Protocol Citation: Chandrashekar K, Anagha S Setlur, S Pooja, M Purushotham Rao, Vidya Niranjan 2024. MetaAMRSpotter: Automated workflow with shell scripting for uncovering hidden AMR hotspots from metagenomes. **protocols.io** https://dx.doi.org/10.17504/protocols.io.e6nvw1jyzlmk/v1

Protocol status: Working
We use this protocol and it's working

Created: May 20, 2024

Last Modified: May 21, 2024

Protocol Integer ID: 100114

Keywords: AMR gene prediction, metagenomics, automated pipeline, shell scripting

Disclaimer

This protocol can be run on Linux & Ubuntu systems with enough RAM and memory (for databases and tools) to enable appropriate data run and generation.

Abstract

This protocol employs a novel, open-source automated pipeline scripted entirely in shell for analyzing metagenomic data from various samples. Designed to streamline the workflow, the pipeline integrates functionalities for pathogen identification, antimicrobial resistance (AMR) gene detection, and listing the probable antibiotics to which the genes are resistant. This user-friendly pipeline eliminates the need for manual tools installation and configuration, simplifying the analysis process. It directly analyzes raw sequencing reads, if there is presence of appropriate reference genomes and runs through the pipeline for each sample. This protocol runs nine tools together, with just one input given at the start of the program. Demonstrated using publicly available data on both a desktop Linux system and a high-performance computing cluster, the pipeline acknowledges potential variations arising from different software tools and versions, providing users the flexibility to modify them as needed. This approach offers a robust solution for metagenomic data analysis from varied samples, facilitating efficient and accurate detection and uncovering of hidden AMR hotspots.

Keywords: AMR gene prediction, metagenomics, automated pipeline, shell scripting

Guidelines

This protocol works well for paired end metagenome samples.

Materials

This protocol employs nine tools that are automated to run one after the other.

Safety warnings

NA

Ethics statement

NA

Before start

1. All necessary tools and databases must be downloaded and installed.
2. Make sure the reference file for the respective selected organism is indexed and placed in the reference folder.

RETRIEVAL OF METAGENOME SAMPLES

1 The metagenomic samples of various organisms can be retrieved from NCBI SRA. Respective organisms' reference genomes can also be downloaded from NCBI Ref-Seq and indexed.

The following command can be used for indexing the reference:

Command

Indexing

```
bowtie2-build <reference.fasta> <index_name>
```

DIRECTORY SPECIFICATION AND UNZIPPING FILES

2 Specify the directory of the file and unzip all .gz files in the directory. Check if the file is found first and if yes, then proceed to the next step.

Code provided below:

Note

```
# Specify the
directory where your .gz files are located

# Unzip all .gz
files in the directory
for file in *.gz;
do
  # Check if the file exists and is a .gz file
  if [ -e "$file" ]; then
    echo "Unzipping $file..."
    gunzip "$file"
    echo "Done."
  else
    echo "File not found or not a .gz
file: $file"
  fi
done
```

RUNNING FASTQC

3 This tool describes the quality of the raw sequence data which is a result of high throughput sequencing techniques. The tool measures length distribution, GC content and level of duplications. Quality score for the sequence which has the potency to have low-quality regions will be detected and the tool also analyzes the adapter sequence and overexpressed k-mers which could lead to errors.

The following code was used to run FastQC for selected genomes.

Note

```
# Check if the "fastqc" directory exists,
and create it if not
if [ ! -d "fastqc" ]; then
  mkdir fastqc
fi

# Create an array to store unique prefixes
prefixes=()

# Loop through all FASTQ and FASTQ.gz files in the
current directory
for input_file in *.fastq; do
  # Extract the
prefix from the input file name

prefix=$(basename "$input_file" | sed -E
's/_[12]\.(fastq|fastq.gz)//')

  # Check if the
prefix is already in the array
  if [[ ! "
${prefixes[@]} " =~ " $prefix " ]]; then

prefixes+=("$prefix")

    # Find the
input files for this prefix

input_r1="${prefix}_1.fastq"

input_r2="${prefix}_2.fastq"

    # Define
output file names based on the prefix

output_r1_paired="result/${prefix}/trimmomatic/1_paired.fastq"

output_r2_paired="result/${prefix}/trimmomatic/2_paired.fastq"

output_r1_unpaired="result/${prefix}/trimmomatic/1_unpaired.fastq"

output_r2_unpaired="result/${prefix}/trimmomatic/2_unpaired.fastq"
    mkdir -p "result/${prefix}/fastqc"
    mkdir -p
"result/${prefix}/trimmomatic"
    echo
"Processing prefix: $prefix"

    # Run fastqc on the input files and save the results
in the "fastqc" directory
    fastqc "$input_r1" -o
"result/${prefix}/fastqc/" -t 16
    fastqc
"$input_r2" -o "result/${prefix}/fastqc/" -t 16

    echo "FastQC analysis completed for
$prefix."
```

RUNNING TRIMMOMATIC

4 This tool is designed to pre-process the next-generation sequencing data. Trimming will enhance the quality of the file. The tool supports both single-end read and paired-end reads data. The tool eliminates low-quality reads which optimizes ensuring all the high-quality data are retained.

The below code runs Trimmomatic.

Note

```
# Define Trimmomatic command with desired parameters
   java -jar
trimmomatic-0.39.jar PE -phred33 "$input_r1" "$input_r2"
"$output_r1_paired" "$output_r1_unpaired"
"$output_r2_paired" "$output_r2_unpaired" ILLUMINACLIP:TruSeq3-PE.fa:2:30:10
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
   # Check the
exit status of the Trimmomatic command
   if [ $? -eq
0 ]; then
     echo
"Trimmomatic completed successfully for $prefix."
   else
     echo
"Trimmomatic encountered an error for $prefix."
   fi
```

ALIGNMENT USING BOWTIE2

5 Bowtie aligns sequences against the references and it supports gapped, local and paired-end alignment. It generates genome index using a technique called Burrows-Wheeler Transform (BWA) via similar algorithms such as Needleman-Wunch and Smith Waterman algorithms. This tool will optimize the sequence read by alignment process.

The provided code runs Bowtie2 tool.

Note

```
# Define the index
file

index="reference/indexed_file_name"


output_dir="result/${prefix}/mapping"
   mkdir -p "$output_dir"

input_file_1="result/${prefix}/trimmomatic/1_paired.fastq"

input_file_2="result/${prefix}/trimmomatic/2_paired.fastq"

  # Check if input files were found
   if [ -z "$input_file_1" ] || [ -z
"$input_file_2" ]; then
      echo "Input files not found in the
current directory."
      exit 1
   fi

  # Run bowtie2
   bowtie2 -x "$index" -1
"$input_file_1" -2 "$input_file_2" --un-conc
"$output_dir/unmapped.fastq"


# Check the exit
status of bowtie2
   if [ $? -eq 0 ]; then
      echo "bowtie2 alignment completed
successfully."
   else
      echo "bowtie2 encountered an
error."
Fi
```

SPADES ASSEMBLY

6 SPAdes is a genome assembly tool which works on de Bruijn Graph algorithm that reconstructs the entire genomes sequence by reading the fragments. It provides simplified graphs for the user. The tool measures the distance between k-mers and adjusts the scores

to accurate distances. The contig file generated has valuable information and are high-quality assemblies that are optimized and analyzed for sequenced data.

The below code runs the Spades assembly.

Note

```
echo "SPAdes assembly with error correction"


forward="result/${prefix}/mapping/unmapped.1.fastq"

reverse="result/${prefix}/mapping/unmapped.2.fastq"

output_path="result/${prefix}/assembly/spades/"

  # Run SPAdes
    spades.py
--meta --pe1-1 "$forward" --pe1-2 "$reverse" -o
"$output_path" --phred-offset
33 -t 8 -m 16 --only-error-correction

  # Rename and
compress output files

output_dir="$output_path/corrected"

output_file_1="$output_dir/unmapped.1.00.0_0.cor.fastq.gz"

output_file_2="$output_dir/unmapped.2.00.0_0.cor.fastq.gz"

    echo
"SPAdes assembly with error correction done"
    echo
"SPAdes assembly"

forward="result/${prefix}/assembly/spades/corrected/unmapped.1.00.0_0.cor.fastq.g
z"

reverse="result/${prefix}/assembly/spades/corrected/unmapped.2.00.0_0.cor.fastq.gz
"

output_path="result/${prefix}/assembly/spades/contigs"

  # Run SPAdes
    spades.py
--meta --pe1-1 "$forward" --pe1-2 "$reverse" -o
"$output_path" -t 16 --only-assembler

  # Rename and
compress output files

output_dir="$output_path/corrected/contigs"

output_file="$output_dir/contigs.fasta"

echo "SPAdes assembly is done"
```

RUNNING QUAST

7 The tool abbreviation stands for Quality Assessment Tool to analyze the genome assembled. The tool compares the sequence by either comparing with the available reference genome to identify the gaps in the contigs or performs de novo comparison without the reference genome and predicts the assembly quality. This tool optimizes the assembled file and predicts the low gene-coverage and provides possible results with tables and graphs.

PROFILING USING METAPHLAN

8 Metaphlan is the most diversely used computational tool to perform microbial profiling. It mainly focuses on metagenomic shot-gun sequencing data. The database has pre-defined markers specific to the microbial community and the sequencing data aligns against the database. The assigned reads are taxonomical labels to the given samples. It provides

insights in composition and diversity of microbial populations. It is essential to determine the microbes in agriculture, health-disease, pathology and food production.

The codes for running Quast and Metaphlan are provided below:

Note

QUAST & METAPHLAN CODE:

```
# Rename and compress output files

output_dir="$output_path/corrected/contigs"

output_file="$output_dir/contigs.fasta"

   echo
"SPAdes assembly is done"
   echo "QUAST"

input="result/${prefix}/assembly/spades/contigs/contigs.fasta"
   output_path="result/${prefix}/quast"

   # Run quast
   quast.py
"$input" -o "$output_path" --min-contig 100

   echo
"QUAST is done"
   echo
"Metaphlan"

output_dir="result/${prefix}/metaphlan"
   mkdir -p
"$output_dir"

output_file_2="result/${prefix}/metaphlan/metagenome.bowtie2.bz2"

output_file_1="result/${prefix}/metaphlan/profiled.txt"


input_r1_pattern="result/${prefix}/trimmomatic/1_paired.fastq"

input_r2_pattern="result/${prefix}/trimmomatic/2_paired.fastq"

 # Check if
input files were found

   metaphlan
"$input_r1_pattern","$input_r2_pattern" --bowtie2out
"$output_file_2" --nproc 32 --input_type fastq -o
"$output_file_1"
   echo
"Metaphlan is done"
```

IDENTIFICATION OF AMR GENES

9 **ABRICATE AMR**

Abricate is a computational tool to identify antimicrobial resistance genes and virulence genes. Microbial genome is considered as the input. The tool uses database which contains AMR genes and is specific to sequences associated to resistance to antibiotics. Followed by comparison of sequence against the reference to determine the high similarity to sequence in AMR gene database.

10 **ABRICATE PLASMID FINDER**

This tool is used to identify the plasmids in the bacterial genome that could have adapted antimicrobial resistance genes, this serves as reference to identify the similarity. The report generated has names corresponding to the matched plasmid and the alignment coverage scores.

11 **ABRICATE VIRULENCE FACTOR**

This is used to find similarity against virulence factor using a pre-built database. The report generated consist of specific virulence factors' names, the alignment coverage and the

virulence factor associated. The virulence factors indicate the risk of pathogenicity specific to the bacterium and this helps to understand the crucial need of developing potential therapies.

Codes for running abricate and detection of AMR genes:

Note

OVERALL ABRICATE CODE TO BE RUN:

```
echo
"abricate"


input="result/${prefix}/assembly/spades/contigs/contigs.fasta"
   output_dir="result/${prefix}/abricate"
   mkdir -p "$output_dir"

output_amr="result/${prefix}/abricate/amr.txt"

output_pf="result/${prefix}/abricate/pf.txt"

output_vf="result/${prefix}/abricate/vf.txt"

log_amr="result/${prefix}/abricate/amr.log"

log_pf="result/${prefix}/abricate/pf.log"

log_vf="result/${prefix}/abricate/vf.log"


   echo "Abricate AMR"
   abricate --threads 16 --mincov 60 --db ncbi
"$input" > "$output_amr" 2> "$log_amr"

   echo "Abricate Plasmidfinder"
   abricate --threads 16 --mincov 60 --db
plasmidfinder "$input" > "$output_pf" 2>
"$log_pf"

   echo "Abricate Virulencefactor"
   abricate --threads 16 --mincov 60 --db vfdb
"$input" > "$output_vf" 2> "$log_vf"

   echo "abricate is done"

   echo "abricate summary"

input_amr="result/${prefix}/abricate/amr.txt"

input_pf="result/${prefix}/abricate/pf.txt"

input_vf="result/${prefix}/abricate/vf.txt"

output_amr="result/${prefix}/abricate/amr_summary.txt"

output_pf="result/${prefix}/abricate/pf_summary.txt"

output_vf="result/${prefix}/abricate/vf_summary.txt"


   echo "Abricate AMR Summary"
   abricate --summary "$input_amr"
> "$output_amr"

   echo "Abricate Plasmidfinder
Summary"
   abricate --summary "$input_pf"
> "$output_pf"

   echo "Abricate Virulencefactor
Summary"
   abricate --summary "$input_vf"
> "$output_vf"

   echo "abricate summary is done"
```

STITCHING THESE CODES TOGETHER - FORMULATING WHOLE PIPELINE

12 These individual codes for each tool were stitched together thereby automating the entire protocol for easy use. All users who would like to use this protocol may choose to stitch the code to run the workflow.

EXPECTED OUTCOMES - HUMAN, POULTRY AND GOAT DEMO

13 This shell scripted workflow has been run for human genomes, poultry and goat to identify the AMR genes and the possible antibiotics they are resistant to. Expected outcomes are provided below.

	A	B	C	D	E	F	G	H	I
	#FILE	SEQUENCE	START	END	STRAND	GENE	COVERAGE	COVERAGE_MAP	GAPS
	result/ERR4083685/assembly/spades/contigs/contigs.fasta	NODE_12457_length_561_cov_2.610672	2	402	-	dfrA40	1-401/513	#ERROR	0/0
	result/ERR4083685/assembly/spades/contigs/contigs.fasta	NODE_1391_length_1533_cov_4.531800	367	1155	-	aadA27	1-789/789	#ERROR	0/0
	result/ERR4083685/assembly/spades/contigs/contigs.fasta	NODE_1775_length_1342_cov_4.503497	399	1235	+	aph(6)-Id	1-837/837	#ERROR	0/0
	result/ERR4083685/assembly/spades/contigs/contigs.fasta	NODE_2203_length_1199_cov_3.133741	55	879	-	blaOXA-1044	1-825/825	#ERROR	0/0
	result/ERR4083685/assembly/spades/contigs/contigs.fasta	NODE_391_length_3677_cov_10.529266	2203	3390	+	tet(39)	1-1188/1188	#ERROR	0/0
	result/ERR4083685/assembly/spades/contigs/contigs.fasta	NODE_46_length_16755_cov_30.079760	384	894	+	dfrA44	1-511/513	#ERROR	0/0

AMR genes detected in Goat metagenome sample

	A	B	C	D	E	F	G	H	I	J
	#FILE	SEQUENCE	START	END	STRAND	GENE	COVERAGE	COVERAGE_MAP	GAPS	%
	result/ERR5295139/assembly/spades/contigs/contigs.fasta	NODE_10225_length_1216_cov_3.423773	211	1077	-	aadE	1-867/867	#ERROR	0/0	1
	result/ERR5295139/assembly/spades/contigs/contigs.fasta	NODE_1051_length_7106_cov_4.404482	1337	1978	+	catD	1-639/639	#ERROR	04-May	9
	result/ERR5295139/assembly/spades/contigs/contigs.fasta	NODE_10859_length_1170_cov_2.605381	71	613	+	sat4	1-543/543	#ERROR	0/0	1
	result/ERR5295139/assembly/spades/contigs/contigs.fasta	NODE_11304_length_1141_cov_5.878453	92	955	+	aadS	1-864/864	#ERROR	0/0	1

Human AMR genes from human metagenome sample

	A	B	C	D	E	F	G	H	I
	#FILE	SEQUENCE	START	END	STRAND	GENE	COVERAGE	COVERAGE_MAP	GAP
	result/SRR6323357/assembly/spades/contigs/contigs.fasta	NODE_10852_length_623_cov_1.850352	1	623	-	aph(6)-Id	136-758/837	..==========.	0/0
	result/SRR6323357/assembly/spades/contigs/contigs.fasta	NODE_13674_length_514_cov_2.904139	137	477	-	lnu(C)	156-495/495	....====/======	01-Jan
	result/SRR6323357/assembly/spades/contigs/contigs.fasta	NODE_14335_length_498_cov_1.162528	155	498	+	aac(6')_E64	1-344/435	#ERROR	0/0
	result/SRR6323357/assembly/spades/contigs/contigs.fasta	NODE_15256_length_476_cov_2.216152	1	476	-	catA9	9-484/651	#ERROR	0/0

Detection of poultry AMR genes from poultry metagenome sample

The reference genomes must be taken according to the genome in question being studied.

Expected result

1. As noted above, the AMR genes can be detected for various samples, with the antibiotics they are resistant to.

2. This workflow may be applied to diverse range of samples to uncover any hidden AMR hotspots.

CONCLUSION

14 This study thus introduces an open-source pipeline for streamlined and quick analysis of metagenomic data from various samples. Scripted entirely in shell, it integrates pathogen identification, AMR gene detection, and antibiotic resistance prediction. The pipeline directly analyzes raw reads whose quality checks have been completed priorly, eliminating manual tool setup and simplifying workflows. Demonstrated on diverse samples, it offers flexibility for customization and facilitates efficient AMR gene detection. Thus, this workflow may be applied to diverse range of samples to uncover any hidden AMR hotspots.

ACKNOWLEDGEMENTS

15 We would like to thank Dr. Akshatha Prasanna, Assistant Professor, Department of Biotechnology, Dayananda Sagar College of Engineering for her inspiring work that led us to this study. The authors are also extremely grateful to Mr. Akshay Uttarkar, Research Scholar at RV College of Engineering, for providing all his valuable inputs.
Special thanks to our research interns Vasupradha SH, Shreya Vinod and Rajnee Joel for helping the authors run the protocol for different samples.

Protocol references

1. Prasanna, A., & Niranjan, V. (2021). Clin-mNGS: Automated Pipeline for Pathogen Detection from Clinical Metagenomic Data. Current Bioinformatics, 16(2), 306-314. https://doi.org/10.2174/1574893615999200608130029

2. de Sena Brandine, G., & Smith, A. D. (2019). Falco: high-speed FastQC emulation for quality control of sequencing data. F1000Research, 8, 1874. https://doi.org/10.12688/f1000research.21142.2

3. Bolger, A. M., Lohse, M., & Usadel, B. (2014). Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics (Oxford, England), 30(15), 2114–2120. https://doi.org/10.1093/bioinformatics/btu170

4. Langdon W. B. (2015). Performance of genetic programming optimised Bowtie2 on genome comparison and analytic testing (GCAT) benchmarks. BioData mining, 8(1), 1. https://doi.org/10.1186/s13040-014-0034-0

5. Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., Pyshkin, A. V., Sirotkin, A. V., Vyahhi, N., Tesler, G., Alekseyev, M. A., & Pevzner, P. A. (2012). SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. Journal of computational biology : a journal of computational molecular cell biology, 19(5), 455–477. https://doi.org/10.1089/cmb.2012.0021

6. Gurevich, A., Saveliev, V., Vyahhi, N., & Tesler, G. (2013). QUAST: quality assessment tool for genome assemblies. Bioinformatics (Oxford, England), 29(8), 1072–1075. https://doi.org/10.1093/bioinformatics/btt086

7. Davies, T. J., Swann, J., Sheppard, A. E., Pickford, H., Lipworth, S., AbuOun, M., Ellington, M. J., Fowler, P. W., Hopkins, S., Hopkins, K. L., Crook, D. W., Peto, T. E. A., Anjum, M. F., Walker, A. S., & Stoesser, N. (2023). Discordance between different bioinformatic methods for identifying resistance genes from short-read genomic data, with a focus on Escherichia coli. Microbial genomics, 9(12), 001151. https://doi.org/10.1099/mgen.0.001151