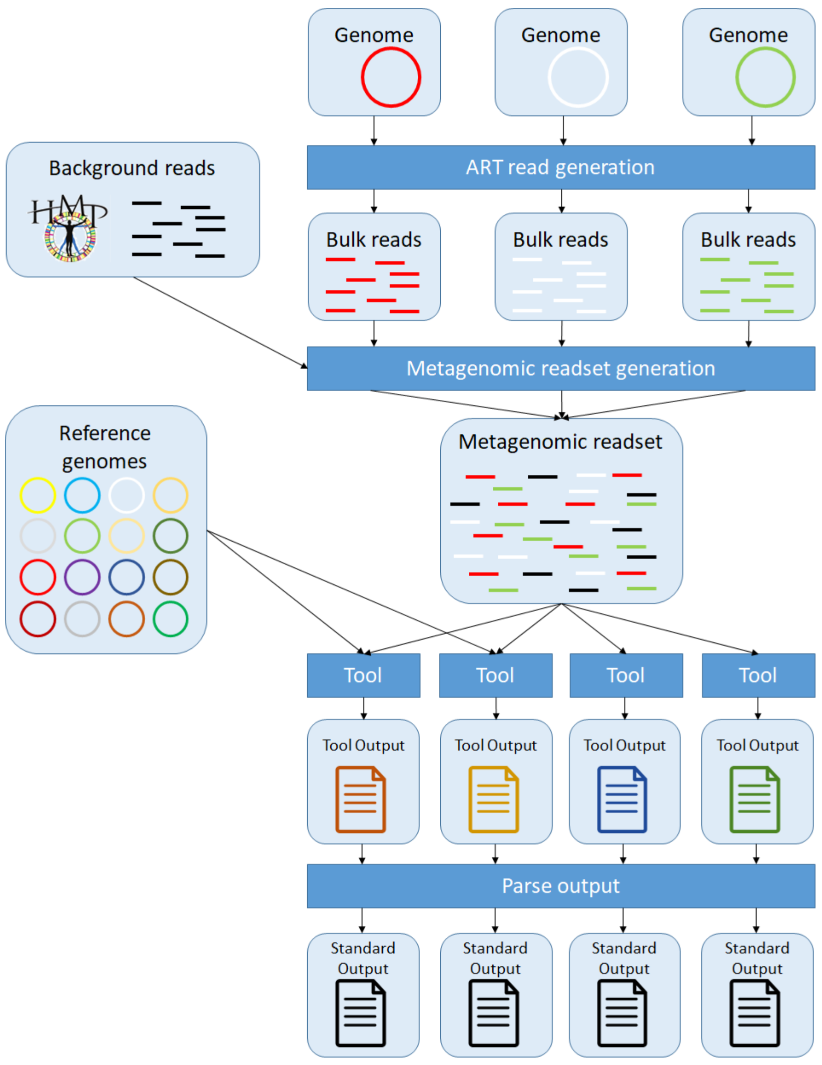
Classification of diverse bacterial populations



Bachelor Thesis Report

Authors: Jasper Uljee and Yorick de Vries

Client Supervisor: Christine Anyansi

Coach: Dr. Thomas Abeel

Coordinator: Ir. Otto Visser

Preface

This thesis is written as part of the Bachelor Computer Science program of the Delft University of Technology. We have benchmarked the most common metagenomic analysis tools and we present the results of this benchmark in this thesis. We hope that our report also gives the reader an insight in both the design of our final product and our development process.

We would like to thank our supervisor Christine Anyansi for her valuable input during our weekly meetings. Furthermore, we would like to thank our coach Thomas Abeel for his feedback and recommendations.

Jasper Uljee,

Yorick de Vries

Delft, February 2018

Abstract

Accurate diagnosis and treatment of patients infected with multiple strains of a pathogen is a challenging task. The use of whole genome sequencing techniques provide high potential to give proper insight into the microbial composition of human metagenomic samples. Distinguishing multiple strains of a certain species is difficult due to the high similarity in genetic content. Currently several tools aimed at the identification of different strains in metagenomic sequence data are available. We present an independent benchmark to compare the performance of several of these tools. The tools have been evaluated with a variety of synthetic metagenomic samples containing strain mixtures of the species *Enterococcus*, *Escherichia coli* and *Mycobacterium tuberculosis*.

To facilitate this research, a benchmark framework in Python 3 was built. This framework made it possible to test the performance of tools aiming at unraveling the composition of sequence data. It is able to automatically generate batches of metagenomic readsets with custom predefined properties. The tools can easily do their analysis on those reads in a streamlined fashion. The output of the tools are put in a standardized format to make the complete comparison of tools easier.

This framework has been built as part of our Bachelor End Project over the course of 10 weeks. In the first few weeks we became familiar with the domain of bioinformatics and the type of tools that had to be included in this research. The implementation of the framework required thorough understanding of the tools and took quite some time to implement. Towards the end of the project, the framework has been used to run the tools with a large variety of synthetic readsets. Analysis of these outputs resulted in an insightful overview of the tools capabilities as presented in this paper.

Table of Contents

Preface 3

Abstract 5

Table of Contents 7

Chapter 1 Introduction 8

Chapter 2 Materials and Methods 10

Chapter 3 Results 15

Chapter 4 Discussion 23

Chapter 5 Conclusion 25

Chapter 6 Recommendations 26

Chapter 7 References 27

Appendix A Project Description 29

Appendix B Project Plan 30

Appendix C Progress 33

Appendix D Design choices 41

Appendix E Framework manual 43

Appendix F Ethical discussion of the project 60

Appendix G Info Sheet 61

Appendix H Terminology 62

Appendix I Additional results: Mutated strains 63

Appendix J Additional results: Total strains found per tool 69

Appendix K Implementation 78

Appendix L Strain databases 80

Appendix M Taxonomic trees 82

Chapter 1 Introduction

When a patient suffers from an infection by multiple strains of a certain bacterial species, providing accurate treatment is a challenge. These so called mixed infections can arise due to multiple transmissions or evolution within the host [1]. Bacteria playing a role herein can genetically be very similar but can have specific mutations causing differences in disease progression, antibiotic susceptibility or virulence [2], [3]. When one does not take such mixed infections into account while making a diagnosis, the outcome of treatment targeted at one particular pathogen strain is uncertain and can be poor [3]. There is a risk that a highly virulent strain, with possibly a relatively low abundance, remains untreated, while the patient is treated solely for another strain. Ideally, a mixed infection should be treated by antibiotics targeted at all pathogens present in the patient.

Classical techniques used for pathogenic bacterium identification rely on the isolation and enrichment of bacteria prior to analysis [4], [5]. The bacterial species is determined based on metabolic or morphologic characteristics. Depending on the pathogen, this process can take up to several days before an accurate diagnosis and is solely based on the isolated strain. However, when a patient is infected with multiple strains, the amount referred to as multiplicity of infection [6], faster and more accurate diagnosis by detecting individual strains is desired.

A suitable alternative to classical techniques to get insight in the microbial composition of samples is the use of metagenomic sequencing [5], [7]. Whole genome sequencing (WGS) techniques can give an overview of the complete metagenomic composition of a sample of interest and therefore provide a platform for mixed infection detection [8]. With the rise of faster and cheaper WGS methods, metagenomic sequencing provides the potential to obtain insightful knowledge about the microbial composition, as well as individual abundance levels [2], [9]. Treatment can therefore be better adapted to the different bacterial strains present [10].

Metagenomic analysis has high potential in both medical and non-medical areas. Besides diagnostics, metagenomic analysis can also be applied to prevent outbreaks via pathogen surveillance [2], [11]. It can detect transmission of pathogens across disease cases, which would not be possible when relying on the analysis of single isolates. This has been demonstrated for *Clostridium difficile* in hospital data [12]. In contrary to medical applications, the ability to detect specific mutations within a mixed bacterial culture has potential for evolutionary strain selection. When the presence of a specific strain of interest is known to be present in a certain mixed culture, one could focus on the isolation from that mixture. This ability is highly desired in disciplines where one relies on natural mutation mechanisms.

WGS can already be applied to mixed viral infections due to their high mutability as a response to the environment [13]. However, the high similarity between bacterial strains poses challenges for metagenomic analysis and requires specialized tools. Currently several metagenomic analysis tools claim to accurately classify and quantify distinct bacterial strains within metagenomic sequencing datasets and it is of interest to what extent these existing genetic tools are able to do so. The tools that are currently used to detect and classify bacterial strains have not been independently tested and benchmarked. As the diagnosis of patients with mixed infections is impacted by the metagenomic analysis tools used, it is of importance to know the accuracy and performance of the tools used for strain-level metagenomic analysis. These tools have been compared and there is a distinction in their approach to identify the strains in a mixed sample [14].

Firstly, a common approach is to compare WGS reads against a reference database to detect and estimate the relative abundance of different strains with the usage of probabilistic models [11]. A reference database with strains similar to what is expected in the sample is required. In case strains are present in a sample which cannot be correctly mapped against the database, strains can be either categorized as similar to certain strains in the database, unknown strains or, in the worst case, they are not detected at all [11], [15]. A common approach is to completely align the reads against the reference database to deduce the sample composition [2], [11], [16]. These tools tend to be computationally demanding. An alternative approach to complete alignment to a reference database is by matching the k-mer profile of the metagenomic reads to the k-mer profiles of the reference strains [17]. Such approaches are able to give an accurate result within a matter of a few minutes.

Secondly, pattern-based methods do not use large reference databases and focus on specific genomic markers instead of the whole read set. Markers used are often SNPs, but also GC-content or specific genes could be used to reconstruct haplotypes [14], [18]. Single markers however, like the 16S rRNA gene, do not provide the resolution needed for accurate strain identification [19]. Such tools are less reliant on reference databases compared to alignment-based methods and have more potential to detect unexpected strains in a sample. This has already been applied to assemble virus genomes from mixed samples. However, the reconstruction of genomes from mixed bacterial cultures is even harder due to the lower mutation rate [12], [20]. Furthermore, such methods tend to be computationally demanding [21], [22].

Metagenomic samples vary on several aspects that could influence the accuracy of the output of the tools. In a clinical setting it is uncommon that a metagenomic sample only contains bacteria of a sole species [23]. Samples with multiple species may make it harder for tools to correctly detect and classify strains as there is more variation in the data [17]. In a mixed infection, usually the patient is infected by one dominant strain compared to other strains with lower abundance. Low abundance strains are harder to classify because relatively few reads of those strains are present. This makes it harder to distinguish these variations in reads from sequencing errors [9]. Furthermore, highly similar strains are also more difficult to classify as the reads of the different strains will also be similar and therefore hard to distinguish. *Mycobacterium tuberculosis* strains, for example, have high similarity compared to strains other bacteria and are thus harder to distinguish [24].

We present an independent comprehensive benchmark of the most common tools for metagenomic analysis of bacterial samples at the strain level. The benchmarking strategy is applied with new, synthetic metagenomic readsets generated from available genome assemblies. In this manuscript, we investigated the effect of properties like sample size and complexity on the quality of the output and runtime of the different tools and put these in perspective.

Chapter 2 Materials and Methods

The performance of several specialized tools aiming at unraveling the microbial composition of metagenomic sequence data have been tested with a variety of inputs. To facilitate this, a framework was developed in Python 3 which enables the automatic creation and analysis of synthetic metagenomic sequence readsets. Synthetic readsets are desired as their properties can be precisely tweaked and the influence of these properties on the output of the tools can directly be measured.

In this framework, bulk readsets were generated from genome assemblies (.fna) of the strains that should be included in the metagenomic samples. These reads are subsequently subsampled from the different portions of bulk reads to make metagenomic read pairs with desired properties. These batches of samples were varied in terms of species, amount of strains, distribution of the strains and coverage. Additionally, several experiments have been spiked with background noise in the form of reads from the Human microbiome project (HMP) [25]. Appropriate samples were made to investigate the influence of these parameters on the performance of the different metagenomics tools.

Experimental Setup

Mixed strain metagenomic readsets have been generated with the framework. The samples differ on several aspects to investigate the effect on the output of the tools. Reads based on several species were made to investigate whether the strain similarity within a species influence the performance of the tools. Additionally, the amount of strains in the sample was varied to investigate to what degree the tools can distinguish multiple strains. For these mixed samples the distribution was varied to see whether the relative abundance of the strains influenced the ability of the tools to discover them.

Furthermore the experiments have been performed at several coverage levels and with added background noise from HMP readsets to see to what extend the tools are able to cope with these more challenging datasets. These experiments have been performed with both mutated versions and the original genomes as strains in the sample to investigate the natural variation expected from real world samples.

Species

Metagenomic test samples were created as either a mix of *Enterococcus* (*faecium* and *faecalis*), *Escherichia coli* or *Mycobacterium tuberculosis* strains to represent different intra-species similarity. *M. tuberculosis* strains are known to be very similar compared to *E. coli* [26]. Additionally two similar *Enterococcus* species have been chosen to represent samples with higher diversity between strains.

For these species 180, 273 and 200 genome assemblies were obtained for *E. coli*, *Enterococcus* and *M. tuberculosis* respectively. These genomes have been used for readset (Table 1) and database (Appendix L) creation. For each of the species the taxonomic tree of the strains was extracted from NCBI on January 2, 2018 [27] and strains were picked uniformly from different clades spread over the tree representing the intra species diversity well (see appendix M for the respective trees).

Strain #	*Enterococcus*	*E. coli*	*M. tuberculosis*
1	79-3 (*faecalis*)	O157-H16_Santai	BTB07-283
2	ATCC8459 (*faecium*)	H7	KT-0078
3	7430821-4 (*faecalis*)	FORC_028	I0001498-0
4	Aus0004 (*faecium*)	O157-H7_FRIK2455	PanR0902
5	UAA702 (*faecalis*)	D8	Erdman
6	T18 (*faecalis*)	S1	M0003138-6
7	E1972 (*faecium*)	0127-H6_E2348-69	UT0055
8	DS5 (*faecalis*)	M10	01-R1278

Table 1 Strains picked for metagenomic sample creation. In samples with ascending distributions the strains with the highest number have the lowest abundance

Distribution

Mixes of 1, 2, 4 or 8 strains have been generated at a fixed total coverage to see whether the tools are able to distinguish different strains within one metagenomic sample. It was both tested how well the tools are able to identify the strains when they are present in even amounts in the sample and when they are present in an ascending distribution where the strains differ a factor two between each other (Table 2). This has been analyzed to reflect the case where there is a dominant strain present along with one or several minor strains. It is of interest to know whether this distribution influences the performance of the tools.

	Even distribution					Ascending distribution			
Amount of strains	1	2	4	8		1	2	4	8
Strain 1	1.0	0.500	0.250	0.125		1.0	0.667	0.533	0.502
Strain 2		0.500	0.250	0.125			0.333	0.267	0.251
Strain 3			0.250	0.125				0.133	0.125
Strain 4			0.250	0.125				0.067	0.063
Strain 5				0.125					0.031
Strain 6				0.125					0.016
Strain 7				0.125					0.008
Strain 8				0.125					0.004
Total Fraction	1.0	1.0	1.0	1.0		1.0	1.0	1.0	1.0

Table 2 Distribution of strains within samples as fractions. Strains corresponding to the strain numbers are indicated in Table 1

Sample complexity

When interested in the sole effect of the amount of strains present in the sample, samples have been made with 1, 2, 4 and 8 strains present. These strains were present at a fixed coverage level per strain regardless of the amount of strains in the sample (Table 3). Additionally, the ascending-incremental distribution is used to analyze the performance of the tools in cases where the strains are not present in equal amounts.

	Even-incremental distribution					Ascending-incremental distribution			
Amount of strains	1	2	4	8		1	2	4	8
Strain 1	1.0	1.0	1.0	1.0		1.0	1.0	1.0	1.0
Strain 2		1.0	1.0	1.0			0.5	0.5	0.5
Strain 3			1.0	1.0				0.25	0.25
Strain 4			1.0	1.0				0.125	0.125
Strain 5				1.0					0.0625
Strain 6				1.0					0.03125
Strain 7				1.0					0.015625
Strain 8				1.0					0.0078125
Total Coverage	1.0	2.0	4.0	8.0		1.0	1.5	1.875	1.9921875

Table 3 Distribution of strains within samples for even-incremental or ascending-incremental samples starting with strain 1 at 1x coverage. This distribution is used to analyze the effect of sample complexity.

Total coverage

The total coverage has been investigated to see to what extend a metagenomic sample should be sequenced to be able to get insightful information. Total coverages of 0.1x, 1x, 10x and 100x were tested to check the sensitivity of the tools when different amounts of reads are available.

Mutation of strains

Strains present in natural metagenomic samples can be similar, but are never exactly the same as strains for which the genomes are known and in a database. In order to mimic such similar strains, mutations have been introduced to genomes before generating readsets. For *Enterococcus* and *E. coli,* 300 SNPs per genome have been introduced as this represents the natural variety among closely related strains [28]. For the more conserved *M. tuberculosis* genomes, only 50 SNPs have been introduced [26]. The 6 fold difference in conservation is also seen in the taxonomic trees (appendix M). To investigate the effect of these mutations, both readsets were generated and analyzed with non-mutated and mutated strain mixtures and compared.

HMP Background

It is common in the real world that metagenomic samples contain background noise originating of other species. Therefore, the tools have also been evaluated on datasets spiked with background data from the Human Microbiome Project. We have constructed datasets with a 9-1 ratio of HMP to strains in order to evaluate the tools in a situation that may arise when detecting strains in a sample taken from a patient.

Metagenomic Tools

9 metagenomic tools have been run and analyzed for their output (Table 4). The tools differed in approach for identifying the amount and identity of the strains present by using either a reference or pattern based approach.

Tool Name	Version	Type	Input	Database	Output			Reference
					Number of strains	Strain Identity	Abundance	
BIB	Oct 3, 2016	Reference	Paired reads (.fq)	Multifasta (made from genome files)	Yes	Yes	Yes	[2]
Pathoscope	2.0	Reference	Paired reads (.fq)	Multifasta (made from genome files)	Yes	Yes	Yes	[16]
Sigma	1.0.3	Reference	Paired reads (.fq)	Genome files	Yes	Yes	Yes	[11]
StrainSeeker	10-feb-2016	Reference (k-mer)	Unpaired reads (.fq)	k-mer database based on genome files and genomic tree	Yes	Yes	Yes	[17]
StrainGR	Jan 2018	Reference (k-mer)	Paired reads (.fq)	k-mer database based on genome files	Yes	Yes	Yes	Unpublished
GOTTCHA	1.0c	Reference	Paired reads (.fq)	GOTTCHA_BACTERIA_c4937_k24_u30_xHUMAN3x.strain	Yes	Yes	yes	[9]
Constrains	2016-04-20	SNP pattern	Paired reads (.fq)	Constrains database (1 genome per species)	Yes	Yes, anonymous strains	Yes	[19]
EstMOI	1.03	SNP pattern	Alignment file (.bam), variant regions (.vcf)	One representative genome (.fasta)	Yes	No	No	[6]
EVORhA	n/a	SNP pattern	Sorted alignment file (.bam)	One representative genome (.fasta and .gff)	Yes	No	Yes*	[20]

*Table 4 Tools used for analysis, *EVORhA does not give direct abundances, but certainties that a haplotype is present*

The tools were implemented and run on readsets as indicated by the tools' authors. The tools that do not natively support paired end reads were tested with both readsets of the pair as if they were single end read sets.

The tools BIB and Pathoscope work with a multifasta as reference database wherein all genomes were concatenated after each other. These tools return their output in the form of the contigs that could be matched. To make sure that every contig of this multifasta corresponds to a single strain, the contigs of every genome assembly were merged together by inserting 150 N's between the contigs.

All experiments were run with 2 CPUs and 16GB memory on a computer cluster. The CPUs used were Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz.

Synthetic readset generation

The process of making synthetic reads consisted of making reads per genome of interest and combining these reads to metagenomic reads in a subsample step. The synthetic readsets used by the tools were made with the ART program, version mountrainier 20160605 [29]. Whole genome assemblies were used as input to create one batch of reads per genome. The reads generated were 150bp long paired end reads using the built in Illumina HS25 profile with a mean fragment size of 200 and standard deviation of 10. These values were indicated by the authors of ART as a proper profile for simulating Illumina reads.

The different batches of ART reads were thereafter subsampled and combined to yield the desired metagenomic samples as described in the experimental setup. Some of the readsets in the experimental setup are spiked with background data originating from HMP reads [25]. The reads were then shuffled to ensure that the reads originating from different genomes were evenly distributed over the output. Hereby read parity was maintained, so the tools can use this parity information in their algorithms. The output files of this read generation step yields a pair of two files in fastq format.

Evaluation

An appropriate measure to evaluate the tools accuracy is how often the tools predict the correct strains (True positives, TP) and how often they make incorrect predictions (False positives, FP). Nonetheless it is just as important to measure the amount of strains that were in the sample but that were not predicted by the tool (False negatives, FN).

These scores have been combined in the form of precision and recall, which can subsequently be combined to the F1 score:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

The measures TP, FP, FN give a good overview of the performance of the tool on a specific dataset, whereas the F1-score is a summary of the overall performance. Tools which do not indicate the identity of the strains were analyzed by comparing the amount of detected strains with the expected amount of strains.

Chapter 3 Results

In this section we evaluate the performance of the tools on our synthetic metagenomic readsets as described in Materials and Methods. These readsets differ in aspects like species, composition and sequence depth and the influence of these parameters on the output of the tools was investigated.

There was a distinction made between tools that use our custom databases of reference genomes and tools that use their own database or a single genome per species as reference. The tools BIB, Pathoscope, Sigma, StrainGR and StrainSeeker allow the construction of a custom reference database with reference genomes while for Constrains, estMOI, EVORhA and GOTTCHA it is not possible to provide such a database.

The tools that used our custom database have been compared in terms of false positives and false negatives plus the F1 scores which can be deduced from these values. For the tools which do not provide the identity of the strains, an overview was made indicating the amount of strains the tools predict to be present in the metagenomic sample.

Reference-based tools

This study researched the influence of several factors on the performance of the reference based tools BIB, Pathoscope, Sigma, StrainGR and StrainSeeker. Firstly, the performance of the tools using mixed samples of either *E. coli*, *Enterococcus* or *M. tuberculosis* were compared. Furthermore the sample complexity in terms of the number of different strains and the distribution of these strains have been analyzed. Lastly, also the influence of sequence depth and the presence of background noise was investigated.

The analysis has been performed on both metagenomic samples consisting of strains with new mutations introduced and with the original strains. It was found that the results of the metagenomic samples with mutations are highly similar to those without mutations (data not shown), implying the robustness when natural variations of strains are present in a sample.

Species

Figure 1 depicts the effect of different species on the performance of the tools on samples with 4 strains present. The tools are better able to detect the correct strains in the samples containing *E. coli* and *Enterococcus* strains than for samples containing *M. tuberculosis* strains. This is as expected due to the higher similarity of *M. tuberculosis* strains in comparison to *E. coli* and *Enterococcus*.

BIB and Pathoscope tend to provide many false positive results for metagenomic samples containing *Enterococcus* and *M. tuberculosis* while this is not the case for *E. coli*. For *M. tuberculosis*, BIB even provides the majority of the database in the output. Sigma is able to correctly predict *E. coli* and *Enterococcus* mixtures without false positives, while for *M. tuberculosis* it tends to have some false positives. StrainGR and Strainseeker appear to perform better on mixtures of *E. coli* strains while having a decent performance on the other two species yielding some false negatives.

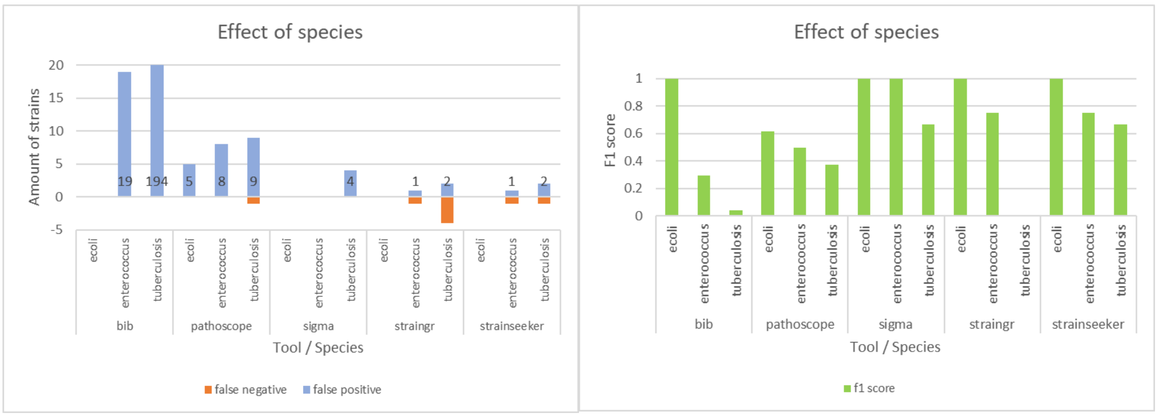


Figure 1 Performance of the tools for different species. Samples contained 4 evenly distributed strains and have a total coverage of 1x. Left: false positive and false negative scores (clipped at 20 strains). Right: F1 scores.

Complexity

Figure 2 shows the influence of the complexity of the metagenomics samples in terms of the amount of different strains present. It is expected that the complexity would negatively influence the performance of the tools due to the fact that more strains need to be accurately distinguished. This is seen for StrainGR and StrainSeeker and to a limited extend for Sigma in the *M. tuberculosis* samples. However, the other tools are well able to identify more complex mixture of strains in the sample.

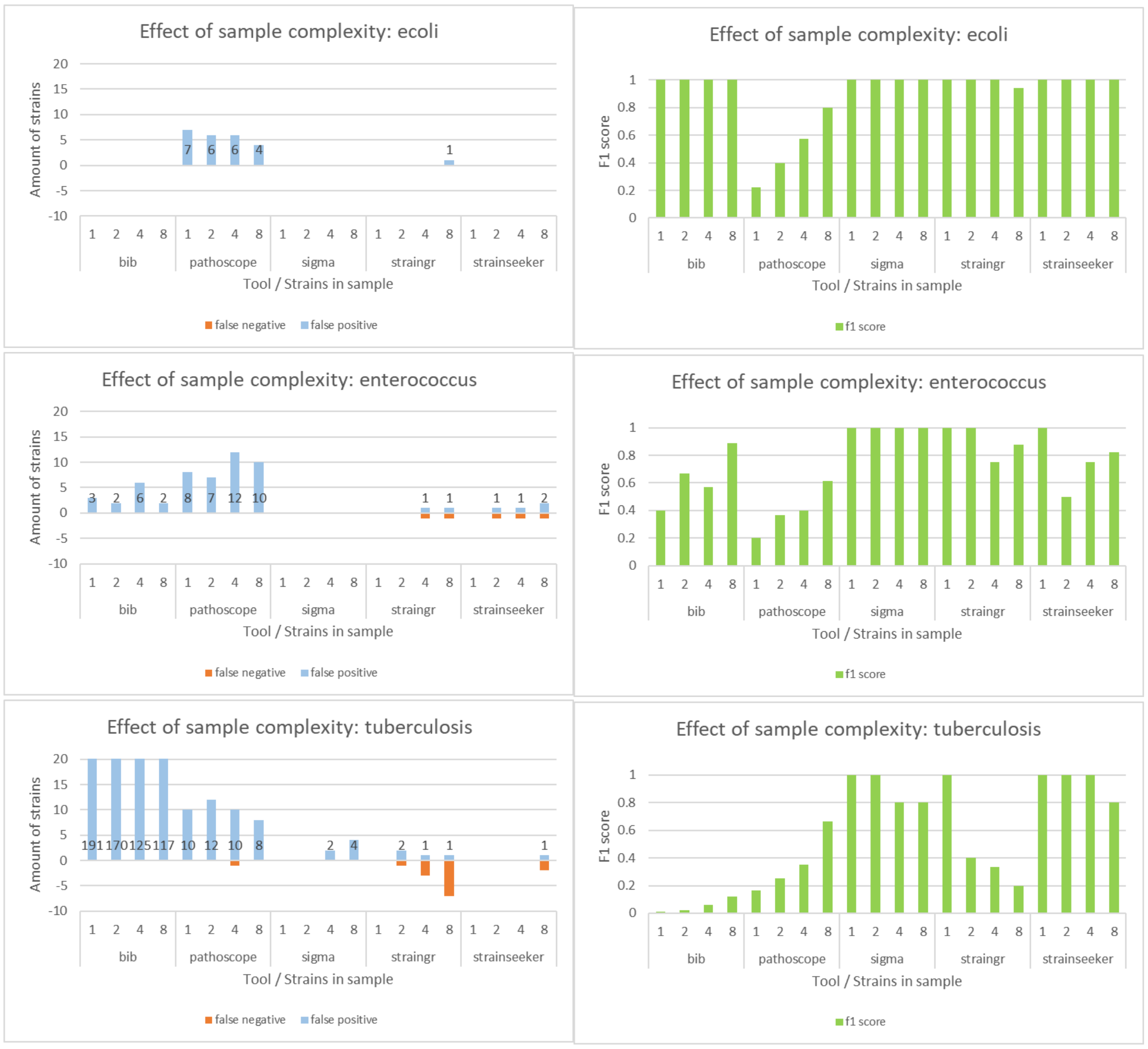


Figure 2 Performance of the tools for different number of strains in the sample. The samples are all evenly distributed mixtures of strains where every strain is present at 1x coverage. Left: false positive and false negative scores (clipped at 20 strains). Right: F1 scores.

The BIB and Pathoscope tools have a quite consistent amount of false positive results in their output regardless of the amount of strains in the sample. For *M. tuberculosis*, BIB found less false positive strains in samples with higher complexity. This can also be related to the higher amount of reads present in samples with more strains. Sigma is well able to identify more complex mixtures of up to 8 strains, while having only a few false negatives when considering *M. tuberculosis* mixtures. In the case of StrainGR and Strainseeker, both tools are almost able to fully identify mixed strains in *Enterococcus* samples. For *M. tuberculosis* however, StrainGR is not able to predict multiple strains, while Strainseeker is able to do so.

Coverage

To investigate the effect of the performance of the tools for varying coverage levels, we looked at the performance of the tools on samples with a total coverage between 0.1x and 100x (Figure 3).

The Sigma tool works remarkably well at all coverage levels tested and is often significantly better than the other tools at lower coverage depths. At 0.1x coverage Sigma usually predicts the correct strains with few false negatives or false positives. The other tools often require a higher coverage level either to reduce the number of false positives or to find all the correct strains in the sample.

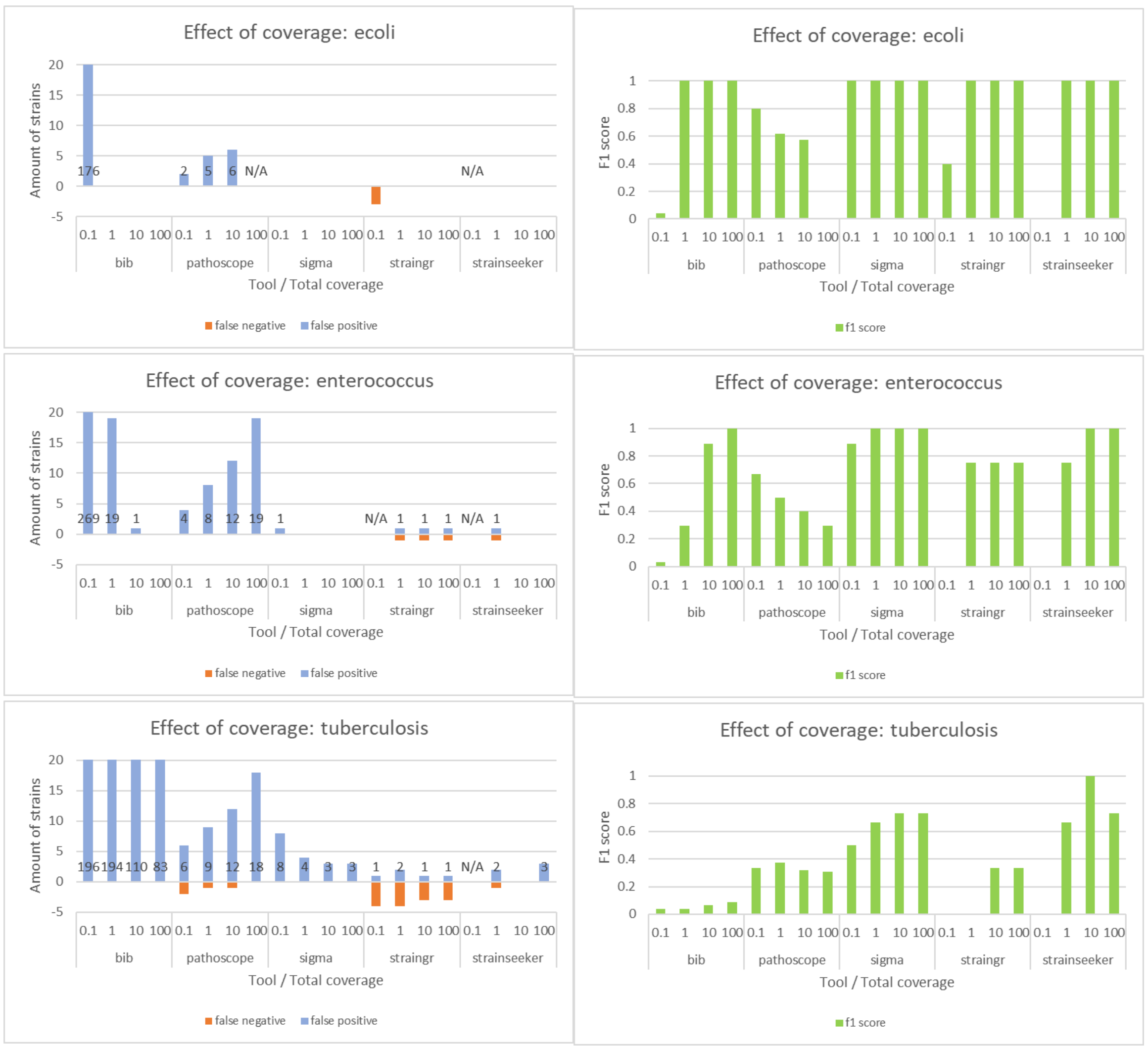


Figure 3 Performance of the tools for different total coverage depths. Samples contain 4 evenly distributed strains. Left: false positive and false negative scores (clipped at 20 strains total). Right: F1 scores. N/A indicates that no result could be obtained.

BIB and PathoScope are good at predicting the correct strains, but tend to show quite some false positive results in their output. For BIB the number of false positives decreases as the coverage increases. At sufficient high coverages it performs similar to Sigma. PathoScope on the other hand tends to have fewer false positives for lower coverages but this amount increases when the coverage level is higher.
Both StrainSeeker and StrainGR perform better at higher coverages than on coverages around 0.1x and 1x. Most of the time, StrainSeeker does not to have any results at a total coverage level of 0.1x. StrainGR also does not seem to benefit from very high coverages as the performance flattens out at coverages higher than 1x.

Distribution

Metagenomic samples consisting of four different strains were made where the strains were either present at the same abundance (even) or in a distribution where the fraction of strains differed among each other with a factor of two (ascending). The effect of these distributions on the performance of the tools is depicted in Figure 4.

Generally, the distribution did not notably influence the tools' ability to predict the correct strains. Strainseeker did however give less accurate results for samples where the strains are distributed ascendingly compared to when the strains are evenly distributed for *Enterococcus*. The opposite is present for Sigma and StrainGR in the case of *M. tuberculosis*.

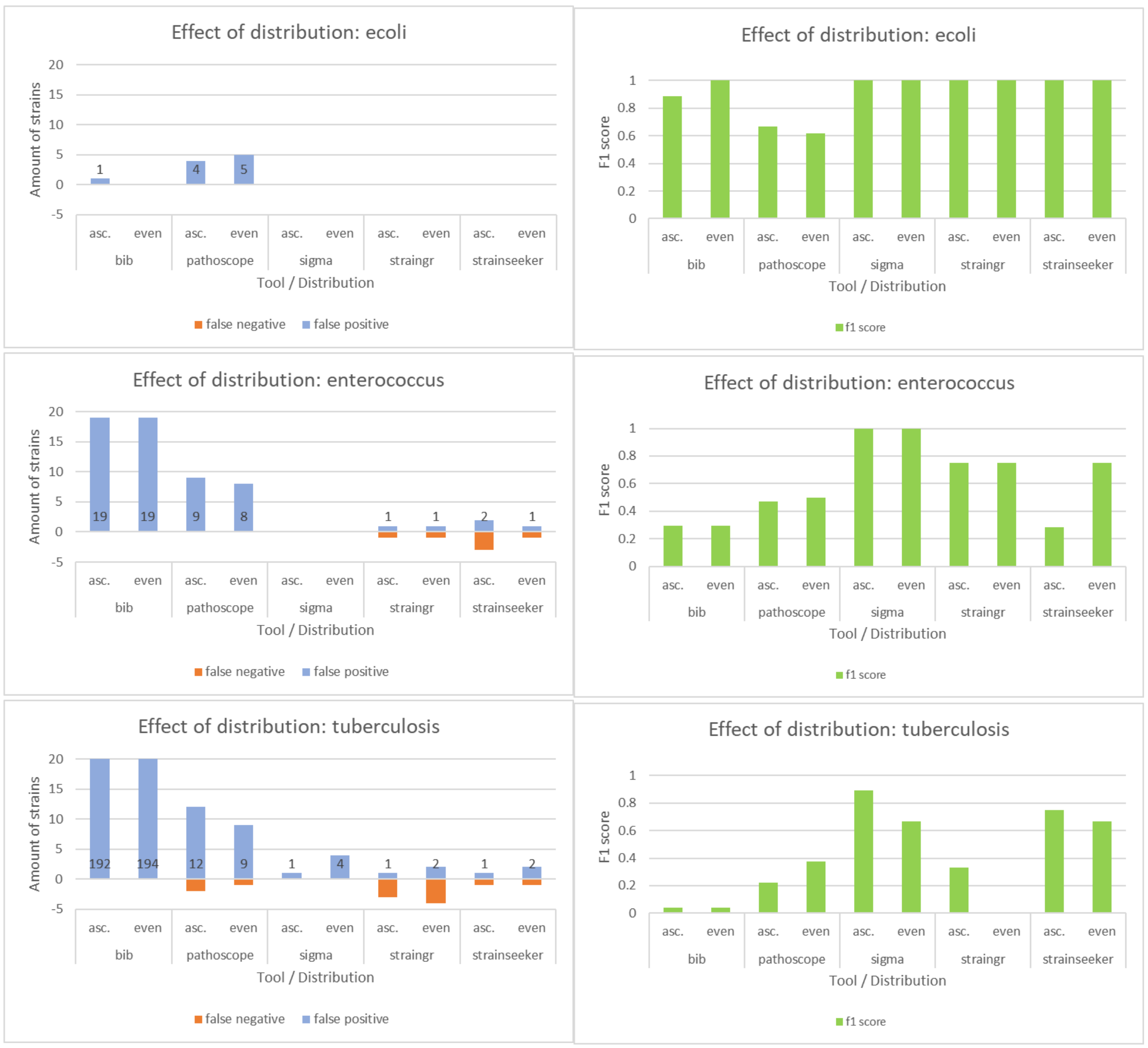


Figure 4 Performance of the tools for either even or ascending distribution of the strains. Samples contain 4 strains at a total coverage level of 1x. Left: false positives and false negative scores (clipped at 20 strains total). Right: F1 scores

HMP Background

Samples with and without HMP background reads have been run with the tools to test the performance when there is noise among the reads (Figure 5).

The tools BIB and Pathoscope both show an increased amount of false positive results when HMP background is present for *E. coli* and *Enterococcus*. For the other tools, there was no significant difference between samples with and without HMP background.

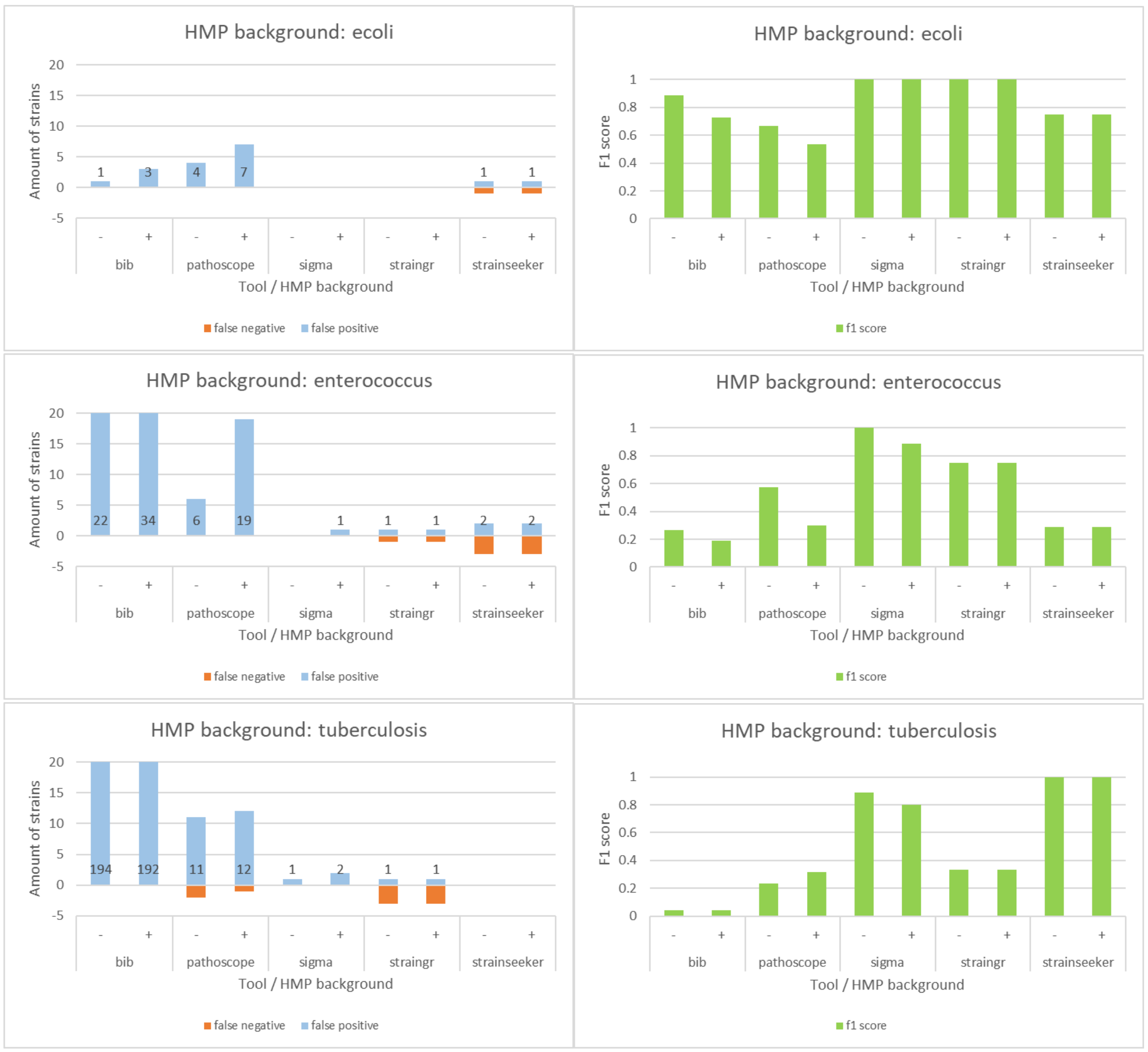


Figure 5 Performance of the tools either without (-) or with background HMP reads (+). Samples contain 4 strains at a total coverage level of 1x. In samples with HMP background, 9x coverage of HMP reads were added. Left: false positive and false negative scores (clipped at 20 strains total). Right: F1 scores

Tools without custom database

It is not possible to run the tools Constrains, estMOI, EVORhA and GOTTCHA with our custom database of complete genome assemblies. Constrains and GOTTCHA were run with databases provided by the authors of the respective tools, while estMOI and EVORhA used a single genome to deduce the complexity of the metagenomic samples. The amount of strains the tools returned for the different datasets at a total coverage of 10x are depicted in Table 5. Additional results for all coverages and tools can be found in appendix I.

The GOTTCHA tool appears to overestimate the amount of strains in the sample and is not able to give output for *M. tuberculosis* samples. For *E. coli* and *Enterococcus* samples, the amount of strains correlates to the amount of strains that are expected. This was also the case at lower coverages. Constrains, estMOI and EVORhA tools are not able to properly deduce the amount of strains in the samples to what is expected. For EVORhA there was a trend found in the amount of deduced strains for *E. coli* and the expected amount of strains. The tool was not able to give any output for mixed *Enterococcus* samples. This is likely due to the fact that a sole *Enterococcus faecalis* genome was used as a reference while the sample also contained *Enterococcus faecium* strains. Additionally the EVORhA and estMOI tools are not able to provide proper do both not give proper output at coverages below 10x.

Tool:	Constrains					
	E. coli		*Enterococcus*		*M. tuberculosis*	
# Strains	ascending	even	ascending	even	ascending	even
1	1	1	0	0	1	1
2	2	2	1	1	1	1
4	2	2	3	4	1	1
8	2	2	3	4	1	1

Tool:	estMOI					
	E. coli		*Enterococcus*		*M. tuberculosis*	
# Strains	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	1	1	1	1	1	1
4	1	1	1	1	2	2
8	1	2	1	1	1	2

Tool:	EVORhA					
	E. coli		*Enterococcus*		*M. tuberculosis*	
# Strains	ascending	even	ascending	even	ascending	even
1	3	4	0	0	4	4
2	4	6	0	0	4	4
4	5	7	0	0	5	5
8	5	8	0	0	5	4

Tool:	GOTTCHA					
	E. coli		*Enterococcus*		*M. tuberculosis*	
# Strains	ascending	even	ascending	even	ascending	even
1	12	12	6	6	0	0
2	22	22	6	6	0	0
4	27	33	7	8	0	0
8	30	45	7	11	0	1

Table 5 Amount of strains found by the tools for the different samples. The samples are either evenly distributed mixtures of strains where every strain is present at 10x coverage, or ascending distributed samples where, starting at the first strain at 10x, every subsequent strain is present at half coverage (see even-incremental and ascending-incremental distributions in materials and methods).

Runtime

The runtime of the tools have been investigated for different input sizes and are plotted in Figure 6. The runtime of the reference based tools BIB, Pathoscope and Sigma is quite long and take up to a day when the input is around 80x coverage.

The k-mer based reference tools StrainGR and Strainseeker are able to provide a result in significant less amount of time. It should be noted however, that both of these tools require a specially constructed database which need to be made before being able to run. The generation of these databases take up a few hours, but only need to be generated once.

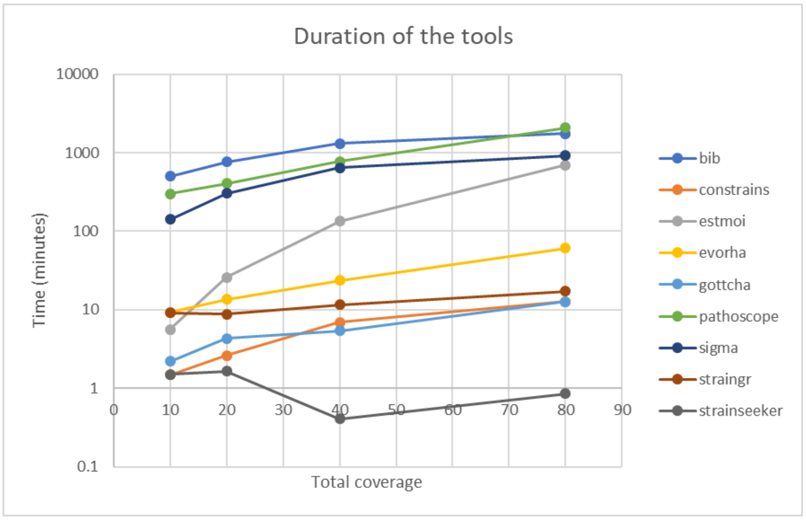


Figure 6 Duration of the tools. The tools have been run with metagenomic samples containing either 1, 2, 4 or 8 E. coli strains of 10x coverage each. The time is plotted on log scale on the y-axis.

Chapter 4 Discussion

The goal of this research is to get insight into how well existing metagenomic tools are able to identify different strains in mixed metagenomic readsets. Additionally we want to know what the effect in terms of sample composition and sequence depth were on the performance of these tools. A pipeline was constructed which automatically generated and analyzed readsets which enabled us to investigate the effect of these properties.

There is a clear distinction between reference based tools which use complete genomes to construct their database (BIB, Pathoscope, Sigma, StrainGR, StrainSeeker) and methods for which this is not possible (Constrains, estMOI, EVORhA, GOTTCHA). The reference based tools appear to provide much more reliable results compared to the other tools. The reason for this is that such tools have more information about what can be expected from metagenomic samples than tools without such a reference database.

While analyzing reads of artificially mutated strains, tools were similarly able to identify the corresponding strains as when analyzing reads with the original strains. This is as expected as the mutations which have been introduced were at random spots in the genome thus the mutated strains and are very similar to the original strains. When the mutations would have been introduced at known SNP locations, this would result in a more natural way of generating new artificially mutated strains. However, this similar output for mutated and non-mutated strains indicates that the tools are well able to cope with natural variation when encountering new or mutants of strains.

Mixtures of *M. tuberculosis* strains are harder to identify than mixtures of *E. coli* or *Enterococcus*. *M. tuberculosis* strains are more conserved among each other [26] and are thus harder to distinguish in a mixed metagenomic sample. The tools were often similarly able to identify strains in mixed *E. coli* and *Enterococcus* metagenomics samples. For PathoScope and BIB however, the outputs for *Enterococcus* and *M. tuberculosis* readsets contained more false positives compared to *E. coli*. This can be attributed to the fact that the used reference database is of lesser quality as quite some genomes consisted of a large number of contigs.

Most tools were well able to identify a high number of strains in more complex samples. Interestingly, while Pathoscope gives quite some false positive results, this is not related to the amount of strains present in the sample. This yields higher F1 scores for Pathoscope when analyzing more complex samples. StrainGR is able to correctly identify pure *M. tuberculosis* readsets, however, the tool is not able to correctly identify more strains in more complex samples, likely due to the high similarity of *M. tuberculosis* strains.

The tools can give insightful results with total coverages as low as 1x. For lower coverages the results are still valid when analyzed by Sigma, but less reliable for BIB and Pathoscope. We see more false positives at a low coverage for BIB and at a high coverage for Pathoscope. We would therefore recommend BIB for higher coverage depths and Pathoscope for lower coverage depths. For the k-mer based tools, StrainGR and StrainSeeker both perform well at total coverages of 1x and up. For lower coverages the tools are often not able to produce results from such small readsets.

The tools were found not to be significantly influenced by the distribution of the strains in the samples. Strains with a lower abundance in the sample were expected to be overshadowed by major strains present, but this was not the case when comparing with samples where all strains were present in equal amounts. Additionally, there is barely any influence when background noise is present in the metagenomic sample. This provides the usefulness of these tools for making diagnoses as classical ways of making diagnoses often miss minor strains in their analysis.

Reference based tools tend to take up quite some time to process the reads. When analyzing high coverage samples, these analyses can take up a few days. Additionally the size of the database influences the runtime as these tools align the reads to all entries in the database. On the other side, the k-mer based tools are able to process the reads in much less time. The computationally intensive part of these tools is constructing the database, which fortunately only has to be done once. This overhead can therefore be considered insignificant when used in practice. However, if the running time is not a concern, the alignment based tool Sigma is preferred due to its accurate output.

There is a clear correlation between the amount of strains expected and what the GOTTCHA tool indicates to be present. The tool however overestimated the amount of strains that are actually in the sample. Among the SNP based tools (Constrains, EVORhA and estMOI), no clear relation could be found in the amount of strains found and the amount of strains that are actually present in the sample. This is likely due to the presence of a sole reference genome which makes it hard to deduce strains. Coverages higher than 100x could possible provide better results, but to our concern, these tools are not suitable for proper mixed strain identification. The tools however, could provide insight into the genomic profile of the sample.

Chapter 5 Conclusion

The current state of the art metagenomic tools are well able to give insight in the microbial composition of metagenomics readsets. With these tools, it is possible to get useful information about readsets, even at coverages as low as 0.1x or in complex samples with up to 8 different strain present. However, mixed samples of the conserved species *M. tuberculosis* remain a challenge to fully unravel.

Of the tested alignment based tools (Pathoscope, BIB and Sigma), Sigma consistently outperforms the other two while having a similar runtime. StrainSeeker and StrainGR, the two k-mer based tools investigated in this research, were found to perform similarly to each other. These tools additionally provide the advantage of a low runtime, desired for making quick diagnoses. However, the performance of these k-mer based tools do not provide as accurate results as Sigma. Lastly, while giving insight into the genomic composition of the samples, the tested SNP based tools are found to be unable to accurately identify the amount or identity of the strains present in a mixed sample.

Chapter 6 Recommendations

Our research has given insight into the performance of several metagenomic tools and the effect of various parameters of the input readsets on the outcome of the tools. It is of interest however how the tools perform when these parameters are extended to yet unexplored values.

We would like to know how many strains the tools are able to identify for metagenomic samples with more strains than already done in this research. Additionally, one could look how much the abundances between several strains can differ until a minor strain is not detected anymore. Lastly, it is not yet known how much background noise could be present in a sample before strains are not detected anymore.

This research covered the performance of the tools in terms of discovery rate. Perhaps more insight into the tools can be obtained when taking the abundances into account which the tools estimate. Furthermore, we solely focused on the performance of the tools on synthetic readsets. Future work should also take readsets of real world data into account to see how accurate the tools perform on such data. Additionally other tools, not taken into consideration should be evaluated too. This is for example the recently published tool StrainEst [30].

In the papers provided with the tools, none of the tools evaluated their own performance on strains of a highly conservative species like *M. tuberculosis*. Yet, it is the species where we see the largest differences in performance between the tools. It is clear that it is the hardest species to predict strains from compared to the other two investigated species. Therefore, it would stand to reason to compare new tools on more highly conservative strains.

For the BIB and Pathoscope tool, quite some false positives which were indicated by the tools as having low abundance in the sample were found. Analysis could be done to see how these tools perform when not taking such low abundance strains into account.

The GOTTCHA tools provided a clear correlation with the amount of strains present and the amount of strains given in its output. Further development of this tool with a custom database could provide a higher performance of this tool on our dataset.

The authors of EVORhA and estMOI do not indicate how the input bam and vcf files should be generated. To our consent we did this in a proper way, however other ways to make these files might improve the performance of these methods on our dataset. Furthermore, these pattern based tools could perform better at higher coverages than tested within this research.

Chapter 7 References

[1] O. Balmer and M. Tanner, “Prevalence and implications of multiple-strain infections,” *Lancet Infect. Dis.*, vol. 11, no. 11, pp. 868–878, 2011.

[2] A. Sankar *et al.*, “Bayesian identification of bacterial strains from sequencing data,” no. December 2015, 2015.

[3] M. Gan, Q. Liu, C. Yang, Q. Gao, and T. Luo, “Deep whole-genome sequencing to detect mixed infection of mycobacterium tuberculosis,” *PLoS One*, vol. 11, no. 7, pp. 1–14, 2016.

[4] J. W. Sahl, J. M. Schupp, D. A. Rasko, R. E. Colman, J. T. Foster, and P. Keim, “Phylogenetically typing bacterial strains from partial SNP genotypes observed from direct sequencing of clinical specimen metagenomic data.,” *Genome Med.*, vol. 7, no. 1, p. 52, 2015.

[5] M. J. Pallen, “Diagnostic metagenomics: potential applications to bacterial, viral and parasitic infections,” *Parasitology*, vol. 141, no. 14, pp. 1856–1862, 2014.

[6] S. A. Assefa, M. D. Preston, S. Campino, H. Ocholla, C. J. Sutherland, and T. G. Clark, “EstMOI: Estimating multiplicity of infection using parasite deep sequencing data,” *Bioinformatics*, vol. 30, no. 9, pp. 1292–1294, 2014.

[7] C. Ford *et al.*, “Mycobacterium tuberculosis - Heterogeneity revealed through whole genome sequencing,” *Tuberculosis*, vol. 92, no. 3, pp. 194–201, 2012.

[8] J. D. O’Brien, Z. Iqbal, J. Wendler, and L. Amenga-Etego, “Inferring Strain Mixture within Clinical Plasmodium falciparum Isolates from Genomic Sequence Data,” *PLoS Comput. Biol.*, vol. 12, no. 6, pp. 1–20, 2016.

[9] T. A. K. Freitas, P. E. Li, M. B. Scholz, and P. S. G. Chain, “Accurate read-based metagenome characterization using a hierarchical suite of unique signatures,” *Nucleic Acids Res.*, vol. 43, no. 10, pp. 1–14, 2015.

[10] T. Cohen *et al.*, “Mixed-strain Mycobacterium tuberculosis infections and the implications for tuberculosis treatment and control,” *Clin. Microbiol. Rev.*, vol. 25, no. 4, pp. 708–719, 2012.

[11] T. H. Ahn, J. Chai, and C. Pan, “Sigma: Strain-level inference of genomes from metagenomic analysis for biosurveillance,” *Bioinformatics*, vol. 31, no. 2, pp. 170–177, 2015.

[12] D. W. Eyre *et al.*, “Detection of Mixed Infection from Bacterial Whole Genome Sequence Data Allows Assessment of Its Role in Clostridium difficile Transmission,” *PLoS Comput. Biol.*, vol. 9, no. 5, 2013.

[13] N. Beerenwinkel and O. Zagordi, “Ultra-deep sequencing for the analysis of viral populations,” *Curr. Opin. Virol.*, vol. 1, no. 5, pp. 413–418, 2011.

[14] C. Anyansi, “Computational methods for strain-level microbial detection in WGS data,” *Unpublished*, 2018.

[15] P. Ji, Y. Zhang, J. Wang, and F. Zhao, “MetaSort untangles metagenome assembly by reducing microbial community complexity,” *Nat. Commun.*, vol. 8, p. 14306, 2017.

[16] C. Hong *et al.*, “PathoScope 2.0: a complete computational framework for strain identification in environmental or clinical sequencing samples,” *Microbiome*, vol. 2, no. 1, p. 33, 2014.

[17] M. Roosaare *et al.*, “StrainSeeker: fast identification of bacterial strains from raw sequencing reads using user-provided guide trees,” *PeerJ*, vol. 5, p. e3353, May 2017.

[18] S. S. Mande, M. H. Mohammed, and T. S. Ghosh, “Classification of metagenomic sequences:

Methods and challenges," *Brief. Bioinform.*, vol. 13, no. 6, pp. 669–681, 2012.

[19] C. Luo, R. Knight, H. Siljander, M. Knip, R. J. Xavier, and D. Gevers, "ConStrains identifies microbial strains in metagenomic datasets," *Nat. Biotechnol.*, vol. 33, no. 10, pp. 1045–1052, 2015.

[20] S. Pulido-Tamayo *et al.*, "Frequency-based haplotype reconstruction from deep sequencing data of bacterial populations," *Nucleic Acids Res.*, vol. 43, no. 16, p. e105, 2015.

[21] P. Menzel, K. L. Ng, and A. Krogh, "Fast and sensitive taxonomic classification for metagenomics with Kaiju," *Nat. Commun.*, vol. 7, pp. 1–9, 2016.

[22] D. T. Truong, A. Tett, E. Pasolli, C. Huttenhower, and N. Segata, "Microbial strain-level population structure & genetic diversity from metagenomes," *Genome Res.*, vol. 27, no. 4, pp. 626–638, 2017.

[23] C. Huttenhower *et al.*, "Structure, function and diversity of the healthy human microbiome," *Nature*, vol. 486, no. 7402, pp. 207–214, 2012.

[24] M. Coscolla and S. Gagneux, "Consequences of genomic diversity in mycobacterium tuberculosis," *Semin. Immunol.*, vol. 26, no. 6, pp. 431–444, 2014.

[25] P. J. Turnbaugh, R. E. Ley, M. Hamady, C. M. Fraser-Liggett, R. Knight, and J. I. Gordon, "The Human Microbiome Project," *Nature*, vol. 449, no. 7164, pp. 804–810, 2007.

[26] K. A. Cohen *et al.*, "Evolution of Extensively Drug-Resistant Tuberculosis over Four Decades: Whole Genome Sequencing and Dating Analysis of Mycobacterium tuberculosis Isolates from KwaZulu-Natal," *PLoS Med.*, vol. 12, no. 9, pp. 1–22, 2015.

[27] NCBI, "Genomic trees," 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/genome/tree/808.

[28] A. R. Manges, "Escherichia coli and urinary tract infections: The role of poultry-meat," *Clin. Microbiol. Infect.*, vol. 22, no. 2, pp. 122–129, 2016.

[29] W. Huang, L. Li, J. R. Myers, and G. T. Marth, "ART: A next-generation sequencing read simulator," *Bioinformatics*, vol. 28, no. 4, pp. 593–594, 2012.

[30] D. Albanese and C. Donati, "Strain profiling and epidemiology of bacterial species from metagenomic sequencing," *Nat. Commun.*, vol. 8, no. 1, p. 2260, 2017.

Appendix A Project Description

The project description stated on BepSys is shown below;

Classification of diverse bacterial populations

Doctors make definitive diagnosis of what's infecting a patient using the output of a long stream of laboratory tests. For diseases like Tuberculosis, conducting and retrieving all the relevant laboratory test results can take months! A different approach to diagnosis can be found in analyzing sequencing data of patient samples. Sequencing makes it possible to rapidly identify which bacteria is infecting a patient by presence of the bacterial DNA. However, in the case where there is more than one subspecies of a bacteria infecting a patient, current technologies are severely limited. Such scenarios typically lead to inaccurate diagnosis, poor treatment and unwanted development of antibiotic resistance.

The majority of current methods to classify distinct subspecies in data are targeted towards viral genomes. The few methods claiming to be compatible with bacterial populations have yet to be comprehensively evaluated. This bachelor's project therefore focuses on the implementation and benchmarking of various tools & algorithms developed to classify & quantify the presence of distinct subspecies in a sequencing dataset. Tools to be evaluated include: Evorha, Shorah, Pathoscope, Gottcha, Constrains, TGS-TB, and others. These tools will be evaluated on both synthetic & real datasets used here.

Appendix B Project Plan

In this appendix we present the Project Plan which describes our planning for this project. We start by analyzing the problems that our final product and our benchmark should solve. Afterwards, we discuss why we chose Python as our programming language and why we chose to use Docker to setup our development environment. Lastly you can find a MoSCoW analysis of the final product and the original planning.

Problem definition and analysis

As metagenomic analysis becomes more widely spread, more and more tools are created in order to perform this analysis. These tools are often hard to install because they have many dependencies and the documentation that their authors provide often omits necessary information. Not only is it hard to use the tools, it is also hard to determine which tool one should use. Up to this point no one has done an independent benchmark to evaluate the performance of these tools. Our product should solve the first problem by making it easy to install and run all the commonly used tools for metagenomic analysis. Our research should solve the latter problem by providing the (potential) users of metagenomic analysis tools with an independent benchmark on the performance of these tools.

Final product

In order to use all these tools, one needs to spend weeks installing and studying the requirements of the tools. They often require advanced knowledge of computer science and biology in order to use them. However, most potential users of these tools do not have that knowledge. Our framework should make it possible for all potential users to do metagenomic analysis on their read sets independent of the knowledge that person may or may not have.

The user should not have to be an expert in computer science. He may not have the required knowledge of programming languages or the knowledge required to build the different tools from source and/or the advanced Linux knowledge that is required to install all the required dependencies. Without this knowledge, it is currently unlikely that someone would be able to use any of the currently available tools. Our product should solve this problem by making it easy to install and run all of the commonly used tools for metagenomic analysis.

The user also should not need to have specific bioinformatics knowledge in order to use the tools. For example, StrainSeeker requires a guide tree in order to build its database. However, creating a guide tree of a large number of whole genomes is not something that can be done easily. A single framework should be able to do all the preprocessing required for all the tools. The user should only be required to supply the reference genomes and the metagenomic samples they want to analyze.

Benchmark

Up to this point no one has done an independent comprehensive benchmark of the most common tools for metagenomic analysis of bacterial samples at the strain level. This means that users have to rely on the papers of the individual tools to determine which tool to use. These papers do not necessarily compare their tool against all other available tools and the experimental setups that they use are often not very comprehensive and not consistent with the experimental setups of the other tools. This makes it impossible to compare the results presented in these papers. We will test all the tools on the same inputs and thus provide an accurate representation of their performance.

Programming Language

We decided to use Python as our programming language as Python is widely available on personal computers and on computing clusters. Python is also dynamically typed and very flexible, which will likely reduce the development time and will make it easy to iterate on different versions of our product. Python also has a very intuitive and extensively documented subprocess library that can be used to run tools that use different languages than Python itself. Python is also used by many of the tools that we want to add to our pipeline like, for example, BIB, Pathoscope and ConStrains. This may make it easier for us to add these tools to our pipeline.

Docker

In order to ensure that our system can be run on any system, we will create a Docker image. We will ensure that we install the dependencies for the tools in the Dockerfile, such that we can guarantee that our development environment will be the same as our production environment. This should make testing our code a lot easier.

MoSCoW analysis

We present an analysis of what we believe to be the requirements for the pipeline. The Must haves are critical for the project to be a success. The Could haves on the other hand are not necessary for the product, but they would be important features of the product.

Must haves:

- The user can create synthetic read sets of genomes using a sequencing read simulator of a whole genome.
- The user can create synthetic metagenomic samples of a set of whole genomes.
- The outputs of the tools are parsed to standardized format, such that it is easily readable by both humans and computers.
- The user can run the most commonly used tools for metagenomic analysis on his own read sets.

Could haves

- The results that the tools output are visualized in graphs and charts.
- The user can determine the settings in a configuration which should be used by the pipeline to determine the input arguments for these tools.
- The user can easily run a benchmark of all the tools for certain experiment parameters that he can determine

Planning

This is the original planning that we presented to our supervisor and our coach. We have iterated on this planning many times during our project, but it gives a good overview of how we envisioned our project would be structured. You can find the original list of deadlines we set in order to structure our focus on delivering an excellent product in Table 7.

Week	Tasks
13 November – 19 November	- Write draft version of the introduction - Finish research for introduction - Investigate tools for benchmark - Investigate potential features of the pipeline
20 November – 26 November	- Investigate tools for benchmark - Investigate potential features of the pipeline - Generate input data for benchmark
27 November – 03 December	- Generate input data for benchmark - Create the skeleton for the pipeline - Add PathoScope2 to the pipeline
04 December – 10 December	- Add more tools to the pipeline
11 December – 17 December	- Finish the minimum requirements of the pipeline - Add tools to the pipeline - Run experiments for the benchmark - Start writing final report
18 December – 24 December	- Add tools to the pipeline - Run experiments for the benchmark - Analyze the results of finished experiments - Work on the final report
25 December – 01 January	- Add visualization to the pipeline
02 January – 07 January	- Slack time
08 January – 12 January	- Run experiments for the benchmark - Analyze the results of finished experiments - Work on the final report
15 January – 19 January	- Work on the final report - Compare the results of the tools
22 January – 26 January	- Prepare presentation - Create info sheet - Make final improvements to report

Table 6 Original planning from the Project Plan

Date	Deadline
13 November	Planning project and outline introduction
17 November	Draft introduction
27 November	Final version introduction
06 January	First draft final report
08 January	All features added to the pipeline
15 January	All tools added to the pipeline
15 January	Docker image with all dependencies
22 January	Final paper

Table 7 List of deadlines for the project

Appendix C Progress

In this appendix we discuss our process and the progress we made during the project. We first discuss the most important parts of our process and how we tested our system. Afterwards we discuss our progress per week, the problems that we encountered and how we solved them. This journal is followed by a discussion on the tools that we discarded from the benchmark. We also discuss the feedback that we received from SIG and how we took advantage of this. Lastly we reflect on how we experienced the project and what we have learned from it.

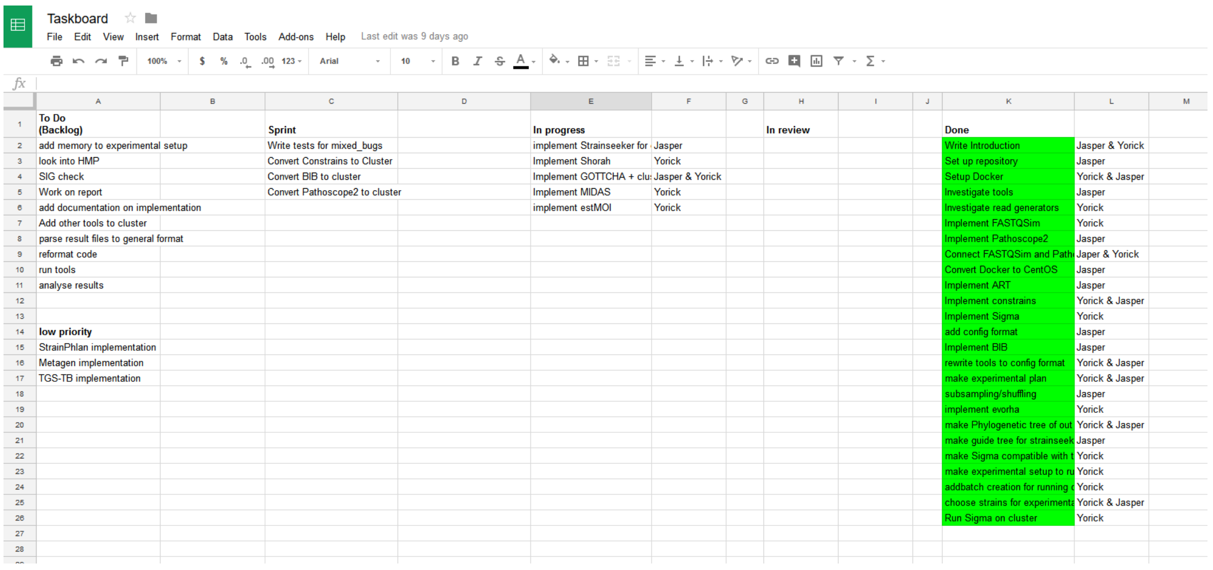
Process

In this section we discuss the most important parts of our process. We discuss why we worked in an agile fashion and why we opted to not use SCRUM. Lastly we discuss the important role that code review played in our process.

Agile

During our study we have learned that the requirements of software projects often change and that it is important that a team works in an agile fashion in order to anticipate these changes and to effectively react to the changed requirements. We knew we would be the first users of our product when we would use it to run the experiments required for the benchmark. We also knew that we were likely not able to completely finish the product before we needed to start running experiments. We thus worked in an agile fashion and had rolling releases. This made it possible to add tools to our experimental setup while we were already in the process of benchmarking other tools.

We used a SCRUM board, but this board was meant to get a high-level overview of what work still had to be done and thus it contained the high-level requirements of our project instead of small units of work as is more common in the SCRUM methodology. Because we worked together in a small group, we wanted to avoid the overhead that comes with using a strict framework to manage the development process. We had the advantage that we could work together in person every day, so our process relied on face-to-face communication. At the start of every day we both discussed our plans for the day and we made design decisions together. This ensured that we were constantly aware of what the other person was working on and that we could anticipate problems and resolve them fast if they did occur.



	A	C	E	F	H	K	L
1	**To Do (Backlog)**	**Sprint**	**In progress**		**In review**	**Done**	
2	add memory to experimental setup	Write tests for mixed_bugs	implement Strainseeker for	Jasper		Write Introduction	Jasper & Yorick
3	look into HMP	Convert Constrains to Cluster	Implement Shorah	Yorick		Set up repository	Jasper
4	SIG check	Convert BIB to cluster	Implement GOTTCHA + clu	Jasper & Yorick		Setup Docker	Yorick & Jasper
5	Work on report	Convert Pathoscope2 to cluster	Implement MIDAS	Yorick		Investigate tools	Jasper
6	add documentation on implementation		implement estMOI	Yorick		Investigate read generators	Yorick
7	Add other tools to cluster					Implement FASTQSim	Yorick
8	parse result files to general format					Implement Pathoscope2	Jasper
9	reformat code					Connect FASTQSim and Path	Japer & Yorick
10	run tools					Convert Docker to CentOS	Jasper
11	analyse results					Implement ART	Jasper
12						Implement constrains	Yorick & Jasper
13						Implement Sigma	Yorick
14	**low priority**					add config format	Jasper
15	StrainPhlan implementation					Implement BIB	Jasper
16	Metagen implementation					rewrite tools to config format	Yorick & Jasper
17	TGS-TB implementation					make experimental plan	Yorick & Jasper
18						subsampling/shuffling	Jasper
19						implement evorha	Yorick
20						make Phylogenetic tree of out	Yorick & Jasper
21						make guide tree for strainseek	Jasper
22						make Sigma compatible with t	Yorick
23						make experimental setup to ru	Yorick
24						addbatch creation for running	Yorick
25						choose strains for experiment	Yorick & Jasper
26						Run Sigma on cluster	Yorick
27							
28							

Figure 7 The SCRUM board that was used during our development process.

Code reviews

We had code reviews for all pull request to protect the quality of our code base. The author of the code explained how it worked and more importantly what the rationale behind the changes is. This made it easy for us to ask each other questions about the code and we could give feedback immediately. It also resulted in that bugs were found fast and that they could be fixed immediately. Another important benefit was that this helped with ensuring that different units of our code base were consistent with each other.

Testing

During our study, we have become well acquainted with the importance of software testing in computer science. While working on this project, we have taken this into account and implemented several automated tests with the Pytest framework. Several components of our framework could be tested in this way. However, we found out that large parts of our code are not easily testable as the code heavily relies on working with external tools. Additionally, the behavior of these external tools is not as predictable and running metagenomic tools can take quite some time, making it less attractive to make automatic tests. Furthermore, large parts of our methods are focused on manipulation of files and do not provide direct return values. We have discussed this issue with our coach and he agreed that for the majority of our code it is hard to write meaningful tests. He agreed that we should not put our focus a lot on automatic testing and that the reason for it lies in the nature of the project. Instead, we should make sure we ran the different tools lots of times throughout the development and be critical on the output that is returned.

Taking this advice, we have focused more on manual testing compared to automatic testing towards the end of the project. Due to the low automatic testability of our code, we ensured that we ran our code extensively. A critical view towards the output of the tools helped us with fixing several bugs. While implementing new tools in the framework, it helped a lot to just run the tool lots of times. By doing this, we were able to get a proper understanding of the tool which helped with the proper implementation of it in our framework.

Journal

In this section we discuss the progress we made each week and the difficulties that we faced during our project.

Research phase

In the first two weeks we focused on writing a project plan and an introduction for our benchmark paper. During the first week we mainly read the papers of the tools that we wanted to benchmark. In the second week we focused more on writing the introduction itself, while continuing to research the field as necessary to write the introduction and to find papers to support the claims made in our introduction.

Main development phase

From week three to the Christmas holidays we primarily focused on developing the pipeline that we would use for our benchmark. This meant that we primarily focused on adding tools to the pipeline and developing features to make our pipeline easy to use.

The next week we researched different sequencing read simulators and selected three simulators that we could potentially use: FastQSim, ART, and MetaSim. We discarded MetaSim because it is not able to generate paired end reads. We chose to use FastQSim because, unlike ART, it is able to generate metagenomic samples from the reference genomes. While we were implementing a wrapper for FastQSim, we also added our first tool to the pipeline: PathoScope2. While implementing the wrapper, we noticed that a module of PathoScope did not work with our genomes. PathoScope tried to download extra metadata for the genome that no longer existed from NCBI. We fixed this by using a script, which was posted in an issue on their GitHub page, to preprocess our genomes for PathoScope.

The fourth week we met with our coach and he informed us that our original plan to run our pipeline inside a Docker container was not possible on the INSY cluster. This meant that we had to ensure that our system would be portable, which required us to refactor parts of our code. During the same meeting our coach recommended that we do not use FastQSim for our benchmark as it is not widely used in the field. On his recommendation we replaced it with ART. We also added two more tools to the pipeline: ConStrains and Sigma. The implementation of a wrapper for Sigma was relatively straight forward. ConStrains on the other hand was a lot more problematic. It required older versions of dependencies, but this information cannot be found in the documentation.

In week five we refactored our code to use configuration files. We updated the tools that were already implemented to accept these configuration files. Simultaneously, we also added the subsampling and shuffling of the reads generated by ART to the pipeline. Lastly, we finished the implementation of the Evorha wrapper, but we ran into the problem that the reported abundances did not sum to one. We had to contact the author in order to understand the output.

In week six we added more tools to the pipeline. We added Shorah, BIB, GOTTCHA and EstMOI. We started working on the GOTTCHA tool, but ran into difficulties getting it directly to work and making a custom database for it.

During the Christmas holidays we worked on creating taxonomic trees and using them to pick the strains for our benchmark, creating a wrapper for StrainSeeker, and we started working on updating our implementation of the tools to make them portable to the cluster. We had quite a bit of trouble with implementing StrainSeeker properly as it did not accept the trees that we created for it. After contacting the authors of StrainSeeker, we were informed that StrainSeeker requires a bifurcating tree. As this was not possible with the tools that we used to create trees at the time, we implemented a different way to create trees using MASH and MEGA.

Benchmark phase

Once our pipeline was almost completely operational we started running experiments and analyzing the results. The remainder of the project was focused on writing this our scientific report.

We started week seven by designing our benchmark. Once we received the green light from our client to continue with the benchmark, we started running the experiments. While the experiments were running, we focused on improving the quality of our code. We reduced the amount of duplicate code and improved the documentation of our pipeline.

In week eight we started analyzing the results from the experiments and writing the materials and methods, and the results section of our paper. We also made it possible for the user to spike metagenomic samples with reads from the Human Microbiome Project.

In week nine our coach determined that we should expand our experimental setup with more experiments. So, we created more experiments and continued with analyzing the results. We also started working more extensively on writing the appendices of the final report. This work continued into the final week when we finished the project.

Discarded tools

We were not able to add all the tools that we initially considered for the benchmark to the pipeline. Some tools were not suitable for the benchmark and others were difficult to implement for different reasons and were therefore considered as low priority tools. We were not able to implement all of the low priority tools. In this section we discuss the specific reasons why each of the following tools was not added to benchmark.

GSM-er

We did not add GSMer to the benchmark as it outputs “Genome Specific Markers”. Although these genome specific markers can be used to calculate according to their paper, there is no documentation on how to do this. We decided that other tools were more important to research first, and we did not have enough time to add GSMer to the pipeline.

MIDAS

We investigated the applicability of the tool MIDAS (Metagenomic Intra-species Diversity Analysis System) for our benchmark. Judging on the paper where the tool is described, the tool seemed very promising. It aims at giving insight in the microbial composition of sequence reads on strain level.

After looking into the github page of MIDAS and doing some test runs with the code, we found out that the output of this tool consists of SNP- and gene profiles. The output generate by this tool is not comparable with the other tools and we therefore discarded the tool for use within this project.

WG-fast

WG-fast was also considered for implementation during the project. However, similarly to the MIDAS output, the WG-fast output was hard to put in line with the other tools. The output consisted of a genomic tree and SNP matrices. No strain identification data could be obtained.

Metasort

Metasort was considered as interesting tool as this tool claims to identify strains from metagenomic data. This tool however relies on their developed way of sequencing sorted bacterial cells. The input reads generated in our framework was not what the algorithm required to run and we considered the tool as not applicable for this research.

Shorah

The Shorah tool was an interesting tool for us as it had similar behavior and output as the EVORhA tool included in our research. Originally this tool was developed for viral genomes, but the potential for bacterial DNA had yet to be investigated. When trying to implement this tool and running it on our synthetic data, we found out that we were not able to run properly and get output from the tool. Upon contacting the author of the tool he confirmed that this tool is likely not applicable for our goal and we discarded the tool.

Eyre et al.

We looked in the tool developed by Eyre et al. for the identification of metagenomic samples. These researchers describe that their method can be applied to detect mixed infections. While analyzing the implementation we found out that their algorithm require input information about which haplotypes to expect in the sample. It was not clear how we could extract this info from our database to get the method to work. We contacted the author but he was not able to provide any help in time.

PanPhlan

PanPhlan was discarded because it can only identify the dominant strain. Although this is useful in many situations, it would not perform well in our benchmark which is focused on finding the tools that can detect as many strains as possible.

StrainEST

StrainEST was released while we were working on this project (December 2017). StrainEST would have been interesting to compare to the other tools, but we were not able to add it to our pipeline before the deadline.

SIG Feedback

In this section we present the feedback we have received from the Software Improvement Group (SIG) on the code quality of the product. As the feedback is in Dutch, we have summarized the feedback in English.

First feedback moment

De code van het systeem scoort 3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Size en Unit Complexity.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld run() in SubSample.py, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld 'change single quotes to double quotes so that it is valid json' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Daarnast zijn jullie een beetje wisselend in hoe groot methodes binnen een class moeten zijn. In sommige bestanden is dat vrij goed uitgesplitst, maar in bijvoorbeeld de constructor van ExperimentalSetup wordt een groot aantal verschillende acties uitgevoerd binnen één methode. Ook bij constructors is het voor de toekomstige onderhoudbaarheid de moeite waard om aparte stappen in het proces naar aparte methodes uit te splitsen.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt. Op dit moment blijft de hoeveelheid tests nog wat achter bij de hoeveelheid productiecode.

Over het algemeen scoort de code dus gemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

Key points:

- A score of three stars on SIGs maintainability model.
- Some of the units of code were longer than average and could be refactored in multiple smaller units.
- These units are also often more complicated than average.
- The number of tests is lower than expected given the size of the production code.

Our response:

We addressed the problems that SIG informed us about and significantly reduced the complexity of our code in these areas. We updated the parts of our code that could be considered as complex and by doing that also reduced the size of our longer methods.

We also added more tests for the system. It is however hard to test a system that relies on running code that is outside one’s control. We have opted to test our code that does not execute tools but that preprocess the data. We have done extensive manual tests to verify the correctness of the code that does execute the tools. We have discussed our testing approach more extensively in the testing section of appendix C.

We are glad that our efforts have been validated with a higher maintainability score for our second feedback.

Second feedback moment

In de tweede upload zien we dat de hoeveelheid code is toegenomen. De score voor onderhoudbaarheid is gestegen, jullie scoren nu ongeveer 3,5 ster. De stijging wordt met name veroorzaakt door een sterke verbetering op het gebied van Unit Complexity. Jullie hebben daar de door ons genoemde voorbeelden aangepast, en er daarnaast voor gezorgd dat deze problemen in de nieuwe code niet meer voorkomen. Ook op het gebied van Unit Size zien we een duidelijke verbetering.

Jullie hebben naast nieuwe productiecode ook nieuwe tests geschreven. Jullie hadden, zoals eerder aangegeven, een vrij grote achterstand qua hoeveelheid testcode, en het is niet gelukt om deze achterstand in te lopen. Dat was ook moeilijk geweest in zo'n korte tijd, maar probeer hier bij toekomstige projecten op te letten. Als je grote systemen gaat aanpassen zorgen tests voor een vangnet om zeker te zijn dat de aanpassingen geen fouten in andere delen van de code veroorzaken.

We kunnen hier uit concluderen dat de aanbevelingen van vorige keer tijdens de ontwikkeling zijn meegenomen.

Key points:

- A score of 3.5 stars on SIGs maintainability model.
- SIG observed a large improvement in Unit Complexity and Unit Size.
- SIGs recommendations have been implemented and the new code that was added did not result in a lower score for these measures.
- Even though more tests were added the ratio of test code to production code is still lower than expected.
- SIG concludes that their recommendations have been addressed in the final part of the development process.

Personal reflection

This project presented us with many challenges that we did not have to face in the other parts of our Bachelor curriculum. This project made it perfectly clear that we were spoiled during our Bachelor courses with extensively documented code and tools that just work. In this section we discuss how we experienced the project and the many things that we learned from it.

DevOps

We were often shielded from Linux during our bachelor courses. When we had to use Linux, we would often get a virtual machine where all the required dependencies were already installed. So, this was the first time that we got to experience Linux in all its glory and at times its wrath. We have become familiar with building programs from source and how to manage the many dependencies and prerequisites that programs may have. We have also become familiar with creating production and development environments and installing all the dependencies that are required to run a complex system like ours. Luckily, we have also learned how to automate this using Docker, which probably prevented us from a lot of headaches.

Creating the pipeline

We also experienced that in real projects you often do not have a clear pathway on how to proceed and that you may not be able to adhere to the planning that you have set out beforehand. We thought that we would be finished with building the pipeline in a few weeks and that we could then completely focus on our benchmark. We quickly realized that this would not be the case and that some tools required more time than we planned for. Getting the tools to work on our system turned out to be very difficult at times and we needed lots of coffee and take-away food to get us through the many late nights that we spend debugging the tools. As the tools were often outdated or lacked the documentation required in order to get it to work as advertised, we often had to find creative solutions. Thankfully, in the cases when we were not able to find a solution, the authors of the tools were very responsive to our inquiries on how to use their tools.

High-performance cluster

This project also gave us the opportunity to play with a high-performance cluster. This was the first time we worked on hundreds of processors simultaneously. We created an experimental setup manager that automatically creates jobs, which can be executed by the cluster, in order to run tens of experiments in parallel. This did not always go as planned. For example, we had to fix bugs that would only occur when a tool was run multiple times in parallel.

Meetings

Our weekly meetings with our helpful supervisor were very useful to keep us on track to do to benchmark in time. It also provided an opportunity for us to explain our plans and discuss our intermediary results with her. We had to update our experimental setup many times before we could convince both her and our coach that our experimental setup would be both interesting and a fair comparison of the tools.

Domain knowledge

This project did not just test our knowledge about computer science as we also had to study elements of biology in order to even understand what we were supposed to do. It was common to encounter unfamiliar terms or ideas in the papers that we had to research for our introduction. So, this often resulted in reading even more papers and searching for more information. Thankfully, we were able to start the project with a lecture on metagenomics in order to kick start us in our research. Thanks to this lecture we had an idea of what metagenomics was and how the different tools worked and how they could be used in practice.

Appendix D Design choices

One of the main challenges we faced during the design process was how we could ensure that the user did not have to install any tools and dependencies used by the tools. This affected our overall design and development process. It also affected the programming language that we used, and it is one of the reasons why we created a Docker image for development that has become part of the end product. In this appendix we highlight the key design choices made during implementation.

Portability

Our system needs to be usable on every Linux system and installing it should be as easy as unpacking a zip file. In order to achieve this, our system expects all tools to be located in a single resources directory and all dependencies that are used by more than one tool in a single dependencies directory. The dependencies in these directories may need to be added to the system path depending on the tools but we ship the product with a file that contains all the paths that the user needs to set in its Bash profile. Once this file is sourced he can execute all the tools using our system. Our product is delivered as a single tar file that once it is unzipped works on a Linux system.

Configuration files

Our system requires configuration files as input in order to run the tools with the correct input arguments. These configuration files can be automatically generated with the default values for each tool, such that the user does not have to change any settings. Below you can find an example of a configuration file for StrainSeeker.

```
[GENERAL]
threads = 8
resources_directory = /path/to/resources
dependencies_directory = /path/to/dependencies
temporary_directory = /path/to/temporary_storage_directory

[STRAINSEEKER]
name = exp_x
outputfolder = /path/to/output_folder
fastadirectory = /path/to/reference_genomes
analysereads = True
builddb = False
builddistancematrix = False
newicktree = /path/to/tree.nwk
database = /path/to/database
inputread_1 = /path/to/read1.fq
inputread_2 = /path/to/read2.fq
```

Read set metadata

Our system creates a metafile for readsets containing the information that our system needs to create a metagenomic sample. It is common to store readsets in the FastQ format, but this format lacks the required information to create artificial metagenomic samples. The biggest problem is that it does not have any information on the coverage depth of the sequenced genome. We create a meta file every time ART sequences a genome, which contains the coverage depth of the read set, the sequencer platform and the size of the reads. From this file our system can calculate the number of reads that are required to subsample enough reads from the read set to simulate any coverage level.

Standardized output

In order to make it easy to interpret the results of the metagenomic tools, the pipeline parses the results that the tools output to a standardized format. The user does not have to find the results that you want in the large web of data that the individual tools output, but he can use our simple format that only contains the essential information. This simple format only informs the user of which strains have been found and what the estimated abundances are of the strains. This standard format makes it easy to compare the results of different tools. We have chosen to use tab spaced values format as this is a format that is easily readable by both humans and machines. The user can analyze the data by hand, but the data can also be imported into, for example, Excel or R. You can find an example of this format for the output from Sigma below.

```
Tool:	sigma
Experiment:	enterococcus-soi8-even_incremental-tc1-hmp0

strain	abundance
DS5	0.13619307612505777
Aus0004	0.13511942204868566
79-3	0.13415152471617883
UAA702	0.13179119476312043
E1972	0.12053854958795196
T18	0.11715474315791295
7430821-4	0.11501375513362465
ATCC8459	0.10829683973144205
```

Standard setup

Our system can be used by first letting our system generate a standard configuration file containing the default settings for all the tools that one wants to run. When the user gives this configuration as input to the pipeline, it will run the tool with the arguments that the user provided in the file and it will move the output of the tool to the folder that the user prescribed. It is not required to create different configuration files for different tools. When a single configuration file with multiple tools is given as input the system will run all the tools sequentially.

Experimental setup

Our system also has an experimental setup manager, which makes it easy to run many experiments on many tools in parallel using the SLURM workload manager that is present on most computing clusters. It automatically creates a configuration file for each experiment for each tool and it creates a SLURM bash script that can be used to run that experiment. Once they system has created all the SLURM bash scripts the user can confirm that they are correct before running all the scripts. This makes it easy to run hundreds of experiments in parallel with only two commands.

Appendix E Framework manual

Version 08-02-2018

Repository location: https://gitlab.noshit.be/AbeelLab/bep_mixedbugs

Here, we present a benchmarking framework built in Python 3 which enables benchmarking the performance of tools aiming at unraveling the composition of metagenomic readsets. This framework is able to automatically generate batches of metagenomic readsets with custom defined properties. The tools can easily do their analysis on those reads in a streamlined fashion. The output of the tools are put in a standardized format to make the comparison of tools easier. The general workflow is depicted in Figure 8

Additionally it is possible to define an experimental setup which enables bulk creation of metagenomic reads with desired properties and the automatic running of the tools with these readsets.

Tools implemented in this framework are shown in Table 8.

Tool Name	Version	Type	Database	Reference
BIB	Oct 3, 2016	Reference	Genome files	[2]
Pathoscope	2.0	Reference	Genome files	[16]
Sigma	1.0.3	Reference	Genome files	[11]
StrainSeeker	10-feb-2016	Reference (k-mer)	k-mer database based on genome files and tree	[17]
StrainGR	Jan 2018	Reference (k-mer)	k-mer database based on genome files	Unpublished
GOTTCHA	1.0c	Reference	GOTTCHA database	[9]
Constrains	2016-04-20	SNP pattern	Constrains database (1 genome per species)	[19]
EstMOI	1.03	SNP pattern	One representative genome (.fna)	[6]
EVORhA	n/a	SNP pattern	One representative genome (.fna and .gff)	[20]

Table 8 Overview of the tools implemented in the framework

The main workflow of the framework consists of the following steps;

- Bulk read generation with the ART program [29]
- Metagenomic readset generation with multisample_fastq.py from the ART bulk reads
- Analyzing the metagenomic readsets created with one of the tools
- Combining the outputs of different tools for comparison

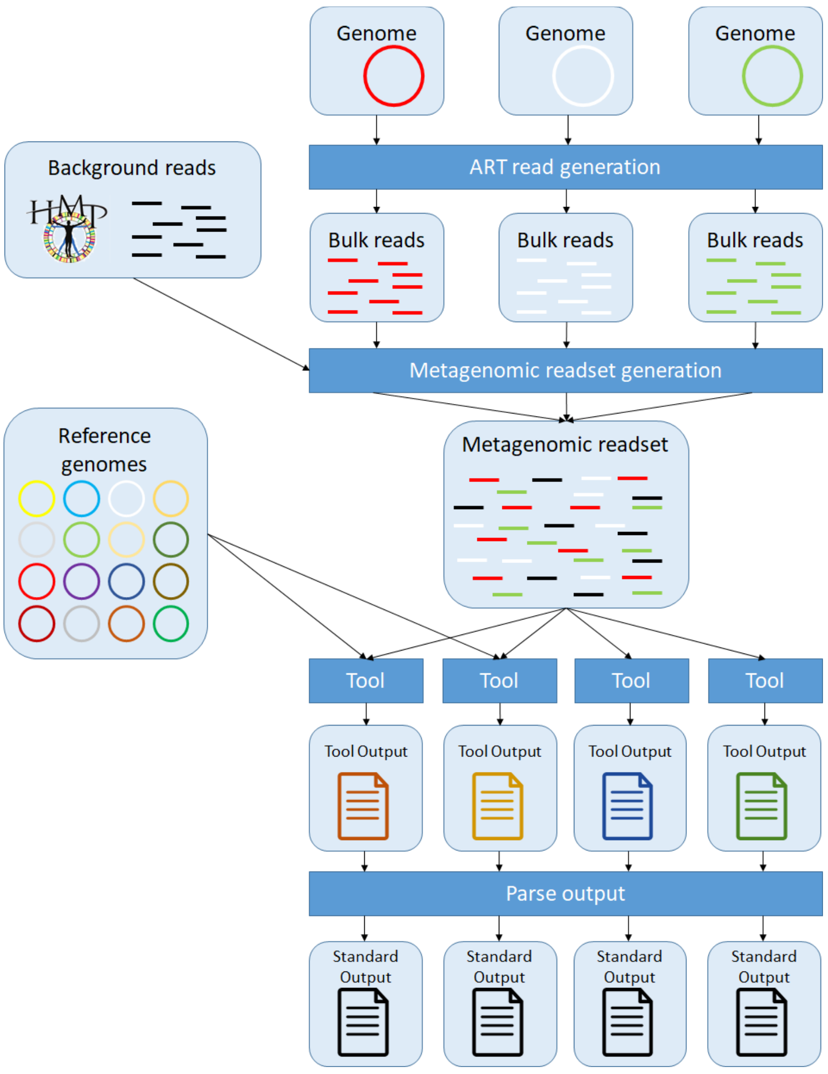


Figure 8 Workflow of the framework

Installation

The basic installation of the framework including importing the tools and their dependencies can be done in a Docker image. Additionally it is possible to export the installed files externally to a computer cluster like the INSY cluster (http://helix.ewi.tudelft.nl/servers/).

Docker Image

In the repository a Dockerfile is provided which automatically installs a CentOS image with the tools and dependencies needed. The Docker image can be installed by navigating to the location of the DockerFile in the repository folder and running the following command;

```
docker build -t bepimage .
```

Docker starts building the image and downloads the necessary files to run the tools. Once the docker image has successfully been built, you can start the image with;

```
docker run -v /path/to/bep_mixedbugs:/code -v /path/to/scratch:/scratch -it bepimage bash
```

In this example command two local folders are mounted;

- /path/to/bep_mixedbugs is the repository folder containing the framework python code
- /path/to/scratch is the folder where all input and output files used to run the framework are placed

INSY Cluster

The installation of the INSY cluster consists of moving the necessary folders to the cluster and adding the dependencies to the Linux profile. Additionally, when one wants to run the StrainGR tool, one should install miniconda3 as described below.

Folders

The installation in the Docker image can be exported to the INSY cluster. To be able to run the framework on the INSY cluster, one should both copy the following folders to the cluster;

- resources (includes the tools that are run from the framework)
- dependencies (external programs that are needed by the tools to be able to run)

To make things easier, the framework installed in the Docker image can automatically generate the packages resources.tar and dependencies.tar file with the following command;

```
python3.6 /code/main/scripts/make_tarfiles.py --outputFolder /scratch/tars
```

Subsequently, on the INSY cluster one can extract these files to the desired output location with the following command;

```
python3 /path/to/bep_mixedbugs/main/scripts/extract_tarfiles.py --outputFolder /path/to/output_folder --
dependenciesTar /path/to/dependencies.tar --resourcesTar /path/to/resources.tar
```

Dependencies

For the tools to be able to properly use the dependencies, the environmental variables should be added to the environment in the profile on the INSY cluster. This can be done by either using the .bash_profile file located in the repository or by adding the following lines to your existing .bash_profile ;

```
export DEPENDENCIES_FOLDER="/path/to/dependencies"
export RESOURCES_FOLDER="/path/to/resources"
#BOWTIE
export PATH="$DEPENDENCIES_FOLDER/bowtie2-2.3.3.1-linux-x86_64:$PATH"
#SAMTOOLS
export PATH="$DEPENDENCIES_FOLDER/samtools-1.6:$PATH"
#MASH
export PATH="$DEPENDENCIES_FOLDER/mash-Linux64-v2.0:$PATH"
#BITSEQ
export PATH="$DEPENDENCIES_FOLDER/BitSeq-0.7.5:$PATH"
#BWA
export PATH="$DEPENDENCIES_FOLDER/bwa-0.7.17:$PATH"
#BCFTOOLS
export PATH="$DEPENDENCIES_FOLDER/bcftools-1.6:$PATH"
```

StrainGR

Lastly, to be able to run the tool StrainGR on the cluster, a miniconda3 environment needs to be installed in your home directory.

Navigate to your home folder ($HOME) and download the miniconda installer with the following command;

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

This script is not directly executable after downloading, run the following command to enable this;

```
chmod +x ./Miniconda3-latest-Linux-x86_64.sh
```

Hereafter miniconda3 can be installed with;

```
./Miniconda3-latest-Linux-x86_64.sh -b
```

To install the StrainGR environment run;

```
./miniconda3/bin/conda env create -f /path/to/resources/StrainGR/environment.yml
```

Next, activate the environment;

```
source ./miniconda3/bin/activate strainge
```

Navigate to the StrainGR folder in resources;

```
cd /path/to/resources/StrainGR/
```

And install the following dependency;

```
pip install pybind11
```

Hereafter, run the following setup file located in the StrainGR tool;

```
python setup.py install
```

The StrainGR environment is now properly installed, you can exit the conda environment with;

```
source deactivate
```

Running single config files

The following instructions are for working with the bep_mixedbugs framework on the INSY cluster. When working in the Docker image, the process is similar. The main difference is that python needs to be called with python3.6 instead of python3 and SLURM sbatch jobs are not available in the Docker image.

When using the framework, config files are used for instructions on what to run. The framework can either be used to run single config files or by running complete batches of config files.

Within the framework it is possible to generate example config files. These files can be edited to the users taste before being run.

Making an example config

To make an example config file for art and subsample, the following command can be run with the framework;

```
python3 /path/to/bep_mixedbugs/main/config.py --tool art --tool subsample --outputFilename
/path/to/example.cfg
```

One or multiple tools can be provided as argument for making the example config with the --tool flag, in the example above both art and subsample entries are made in the example.cfg file. This contents of this example config is shown below;

```
[GENERAL]
threads = 2
resources_directory = /resources
dependencies_directory = /dependencies
scratch_directory = /scratchfolder

[ART]
inputfiles = /data/ARTinputfiles/
paired = True
coverage = 10
outputalignmentfiles = False
readlength = 150
outputdirectory = /data/ARTreadsets
standarddeviationfragmentsize = 10
sequencer = HS25
fragmentmeansize = 200
outputsam = False
platform = illumina

[SUBSAMPLE]
sources = [{'coverage': '1', 'metaFile': '/data/ARTreadsets/read_S_boydii_Sb227-fna.meta'},
{'coverage': '2', 'metaFile': '/data/ARTreadsets/read_GCA_001692795-1_ASM169279v1-fna.meta'},
{'coverage': '2', 'metaFile': '/data/ARTreadsets/read_S_boydii_CDC_3083-94-fna.meta'}]
hmp_read1 = /path/to/read_1
name = exp_x
outputfileprefix = metaread
outputdirectory = /data/MetagenomicSamples
```

```
hmp_read2 = /path/to/read_2
hmp = 0
```

The [GENERAL] section contains basic info needed for the tools to run like the amount of threads that should be used (if available in the tool) and the location of the resources and dependencies directories. The scratch_directory indicates the location where the tools are allowed to store their temporary data. The sections other than [GENERAL] are specific for certain tools.

Running a single config file

To run the framework with a config file, the following command can be run;

```
python3 /path/to/bep_mixedbugs/main/main.py --configFile /path/to/example.cfg
```

When multiple tools are indicated in this config file, the framework will run these sequentially. After the config has been run, the file example.cfg.log is created containing the start and endtime of running the config.

To be able to run StrainGR one should first activate the corresponding miniconda environment. This can be done with the following command taking into account that miniconda3 is installed in the $HOME directory during the installation;

```
source $HOME/miniconda3/bin/activate strainge
```

Config parameters

With the framework it is possible to generate an example config for every available tool with the --tool parameter;

Sample generation: art, subsample

Metagenomic tools: constrains, pathoscope, evorha, sigma, bib, strainseeker, gottcha, estmoi, straingr

General

At the start of the config file general parameters are defined. These consists of the general location of folders and the amount of threads that should be used in case the tool allows multithreading.

```
[GENERAL]
dependencies_directory = /dependencies       Location of the dependencies folder
resources_directory = /resources             Location of the resources folder
scratch_directory = /scratchfolder           Location of the temporary folder
threads = 2                                  Amount of threads that should be used if possible
```

ART Read generation

For making bulk readsets the ART program can be configured with the config below. The default setting is to use the HS25 Illumina profile with a readlength of 150, a fragment size of 200 and standard deviation of 10. Additionally, the amount of reads that need to be generated in terms of total coverage can be set (default 10).

The ART program makes readsets for every genome file (file with .fa or .fna extension) in the inputfiles folder. The location of the reads, along with a metafile with information about of the reads is put in the outputdirectory.

Lastly, it can be set whether the user wants .sam or .aln files of the reads (default False).

```
 [ART]
inputfiles = /data/ARTinputfiles/
outputdirectory = /data/ARTreadsets
sequencer = HS25
readlength = 150
outputsam = False
outputalignmentfiles = False
paired = True
fragmentmeansize = 200
standarddeviationfragmentsize = 10
coverage = 10
platform = illumina
```

Subsample

By running the subsample config, you combine the reads generated by the ART program together to make metagenomic reads. The metafiles of the reads to be used should be defined in the sources parameter, along with the desired coverage of that strain in the metagenomic sample.

Additionally, this tool is able to add background data from HMP reads. The amount used for hmp is the portion of background reads used in addition to reads defined in source.

The name of this metagenomic sample can be put in the name parameter. This value is used as the directory name in the output directory. After running subsample, the reads are all combined and shuffled (remaining parity) into two files in the outputdirectory.

```
[SUBSAMPLE]
outputfileprefix = metaread
name = exp_x
sources = [{'coverage': '1', 'metaFile': '/data/ARTreadsets/read_S_boydii_Sb227-fna.meta'},
{'coverage': '2', 'metaFile': '/data/ARTreadsets/read_GCA_001692795-1_ASM169279v1-fna.meta'},
{'coverage': '2', 'metaFile': '/data/ARTreadsets/read_S_boydii_CDC_3083-94-fna.meta'}]
hmp = 0
hmp_read1 = /path/to/read_1
hmp_read2 = /path/to/read_2
outputdirectory = /data/MetagenomicSamples
```

Metagenomic tools

The metagenomic tools all have a similar way their config file is set up. In the parameter name the experiment name can be defined. This name is used as folder name for the output of the tool. Additionally, the two reads of a read pair to be analyzed can be defined with inputread_1 and inputread_2. When paired is set to False, the path to a single read file should be defined under the term “inputread”. However, this is not supported for all tools. Lastly the output directory should be defined where the results can be put once finished analyzing. For BIB also it is possible to provide custom values for the resultsprefix, indexname, multifastaname when desired.

Some tools have additional parameters needed for defining or creating specific databases. The creation of these databases is covered in next section.

```
[PATHOSCOPE]
name = exp_x
inputread_1 = /data/MetagenomicSamples/metaread1.fq
inputread_2 = /data/MetagenomicSamples/metaread2.fq
outputdirectory = /data/output/pathoscope
referencefiles = /data/references

[SIGMA]
name = exp_x
inputread_1 = /data/MetagenomicSamples/metaread1.fq
inputread_2 = /data/MetagenomicSamples/metaread2.fq
outputdirectory = /data/output/sigma
referencefiles = /data/sigma-input/references
paired = True

[BIB]
name = exp_x
inputread_1 = /data/MetagenomicSamples/metaread1.fq
inputread_2 = /data/MetagenomicSamples/metaread2.fq
outputdirectory = /data/output/bib
referencefiles = /data/ARTinputfiles
resultsprefix = results
indexname = index
multifastaname = references
extractcoregenomes = False
mauvenumberofgenomes = 10

[STRAINGR]
name = exp_x
inputread_1 = /data/ecoli-soi1-even-tc1-hmp0/metaread1.fq
inputread_2 = /data/ecoli-soi1-even-tc1-hmp0/metaread2.fq
```

```
outputdirectory = /data/straingr/output
referencefiles = /data/references
create_database = True
database = /data/DB/straingr/ecoli/pan-genome-db.hdf5

[STRAINSEEKER]
name = exp_x
inputread_1 = /data/MetagenomicSamples/metaread1.fq
inputread_2 = /data/MetagenomicSamples/metaread2.fq
outputdirectory = /data/output/strainseeker
database = /data/ecoli_db_strainseeker
builddistancematrix = False
builddb = False
analysereads = True
newicktree = /data/tree/tree.nwk
fastadirectory = /data/references

[CONSTRAINS]
name = exp_x
inputread_1 = /data/MetagenomicSamples/metaread1.fq
inputread_2 = /data/MetagenomicSamples/metaread2.fq
outputdirectory = /data/output/constrains
paired = True


[GOTTCHA]
name = exp_x
inputread_1 = /data/MetagenomicSamples/metaread1.fq
inputread_2 = /data/MetagenomicSamples/metaread2.fq
outputdirectory = /data/output/gottcha
paired = True
gottchadatabase = /data/bep_mixedbugs/gottchaDB/database/GOTTCHA_BACTERIA_c4937_k24_u30_xHUMAN3x.strain

[EVORHA]
name = exp_x
inputread_1 = /data/MetagenomicSamples/final/metaread_fixed1.fq
inputread_2 = /data/MetagenomicSamples/final/metaread_fixed2.fq
outputdirectory = /data/output/evorha
paired = True
evorha_reference_fasta = /data/evorha-input/reference/EcoliK12MG1655_truncated.fasta
evorha_reference_gff = /data/evorha-input/reference/EcoliK12MG1655_truncated.fasta.gff

[ESTMOI]
name = exp_x
inputread_1 = /data/estmoi-input/testdata-fq/read1.fq
inputread_2 = /data/estmoi-input/testdata-fq/read2.fq
outputdirectory = /data/output/estmoi
paired = True
estmoi_reference_fasta = /data/estmoi-input/testdata-fq/c1.fasta
```

Tool databases

For the tools Pathoscope, Sigma and BIB the folder where the tools can find the reference genome files should be defined. These reference files can be in the form of .fa or .fna files. The framework is able to automatically construct databases from these genome files.

BIB

Optionally, it is possible for BIB to create the database with progressivemauve by setting “extractcoregenomes” to True. The “numberofcoregenomes” defines the group size on which the core genomes are determined.

It is however advised not to do this step of extracting the core genomes as this can take up to a few days. When the “numberofcoregenomes” is set at a low value like 5, the creation of a database can be done within a day however. It was observed that BIB is well able to run its algorithm without the extraction step of core genomes.

StrainGR

For StrainGR it is possible to either create a new database from the genome files or to use an existing one by setting the create_database variable.

Strainseeker

For Strainseeker a special database needs to be created before being able to analyze metagenomic reads. When running Strainseeker one can choose to either to make a database or analyze the reads with an existing database. The database creation of Strainseeker consists of several steps.

Firstly, one should make a distance matrix of the genomes used for the database. This is done when “builddistancematrix” is set to True. This option uses the genome files in “fastadirectory” as reference files.

Secondly, the distance matrix should be converted to a newick tree. This conversion is not possible to be automated in the pipeline. This distance matrix should be imported into the MEGA software and converted to a newick tree. The “construct UPGMA tree” option should be chosen within the program. The obtained newick tree can subsequently be used by to make a database for strainseeker, the “builddb” flag must be set to True to start this process.

Once a database has been created, strainseeker can be run with this database and the metagenomic reads. Therefore the flag “analysereads” should be set to True.

GOTTCHA

To run GOTTCHA you should download one of the databases provided by the authors of the tool. These databases can be found at; ftp://ftp.lanl.gov/public/genome/gottcha/

EVORhA and estMOI

The tools EVORhA and estMOI do both not use a database of genomes as reference, but just a single genome. The estMOI tool only needs a .fasta/.fna genome assembly file as reference while EVORhA also needs a .gff file with gene information. These files are obtainable from the NCBI website.

Experimental setup

Instead of manually creating single config files and running these with the help of the framework, it is possible to define a complete experimental setup and running this in an automated fashion. This can be done by either running lists of config files sequentially (advised for use in the Docker image) or by starting batches of SLURM jobs in parallel (advised when using the INSY cluster).

To do this the experimental_setup.py should be called. For the calls the parameter --workDirectory is required and needs to contain the bep_mixedbugs repository. This path is implemented in the files and used to call the framework from the sbatch files used for the running SLURM jobs.

Making a Setup file

An example setup file for such a run can be generated with the command;

```
python3 /path/to/bep_mixedbugs/main/experimental_setup.py --workDirectory
/path/to/workDirectory --make_setup_file /path/to/experimental_setup.cfg
```

You can edit this file to your personal needs. The example file generated with the command above looks as follows;

```
[EXPERIMENTAL_SETUP]
resourcedirectory = /scratch/bep_mixedbugs/resources
dependencydirectory = /scratch/bep_mixedbugs/dependencies
strainsofinterestdirectory = /scratch/bep_mixedbugs/soi
referencedirectory = /scratch/bep_mixedbugs/references
artreadsdirectory = /scratch/bep_mixedbugs/art_readsets
metagenomicdirectory = /scratch/bep_mixedbugs/metagenomic_samples
outputdirectory = /scratch/bep_mixedbugs/outputs
bulkdirectory = /scratch/bep_mixedbugs/bulk

strainseekerdatabases = /scratch/bep_mixedbugs/DB/strainseeker
straingrdatabases = /scratch/bep_mixedbugs/DB/straingr
gottchadatabase = /path/to/GOTTCHA_BACTERIA_c4937_k24_u30_xHUMAN3x.strain
estmoi_reference_database = /scratch/bep_mixedbugs/DB/estmoi/database
evorha_reference_database = /scratch/bep_mixedbugs/DB/evorha/database

qos = long
time = 24
threads = 2
memory = 16

species = enterococcus, ecoli, tuberculosis
strainsofinterest = [1, 2, 4, 8]
distribution = even, ascending
totalcoverages = [0.1, 1, 10, 100]
hmpbackground = [0]

hmp_read1 = /scratch/bep_mixedbugs/hmp_reads/SRS014613.fa_read_1.fq
hmp_read2 = /scratch/bep_mixedbugs/hmp_reads/SRS014613.fa_read_2.fq
```

These parameters should be adapted to what you want to run. In the table below an overview is given what the parameters indicate;

Parameter	Description
Folders	
resourcedirectory	Path to the resource directory, containing the tools being called by the framework
dependencydirectory	Path to the dependencies directory, containing software needed by the tools to be able to run
strainsofinterestdirectory	Path to the folder with genome assemblies that should be included when making metagenomic reads, for every species in the experimental setup a folder should exist with the name of the species containing the genome files
referencedirectory	Path to the folder with genome assemblies that should be used for making reference databases by the tools that use. For every species in the experimental setup a folder should exist with the name of the species (containing the genome files)
artreadsdirectory	Path where ART can store the generated bulk readsets
metagenomicdirectory	Path where metagenomic samples should be stored
outputdirectory	Path where the tools' output is placed after running
bulkdirectory	Path where the tools can put their temporary data while running
Tool-specific databases	
strainseekerdatabases	Path to the Strainseeker databases, for every species in the experimental setup a folder should exist with the same name (containing the db_binary database file)
straingrdatabases	Path to the StrainGR databases, for every species in the experimental setup a folder should exist with the same name (containing the pan-genome-db.hdf5 database file)
gottchadatabase	Path to the GOTTCHA database used, including the prefix of the database (for example GOTTCHA_BACTERIA_c4937_k24_u30_xHUMAN3x.strain)
estmoi_reference_database	Path to the estMOI reference genomes, for every species in the experimental setup a folder should exist with the same name. This folder should contain the genome as <species_name>.fna
evorha_reference_database	Path to the EVORhA reference genomes, for every species in the experimental setup a folder should exist with the same name. This folder should contain the genome as the files <species_name>.fna and <species_name>.gff
SLURM	
qos	QOS used for running sbatch files, (can be short, long or bigmem)
time	Time (hours) used for running sbatch files
threads	Threads used/ CPUs used for running sbatch files
memory	Memory (GB) used for running sbatch files
Metagenomic properties	
species	Different species used in experimental setup
strainsofinterest	Amount of strains that should be included per metagenomic sample
distribution	Distributions used in experimental setup (can be even, ascending, even_incremental, ascending_incremental)
totalcoverages	Coverages used in experimental setup
hmpbackground	Amount of HMP background used in experimental setup

hmp_read1	Path of HMP read 1 of the paired read used for making HMP background
hmp_read2	Path of HMP read 2 of the paired read used for making HMP background

Folders
For the experimental setup the absolute paths of the folders should be defined so the framework is able to point the tools to the right directories. For the output, the folders do not need to exist yet, these will be made automatically in the framework.

Tool-specific databases
The databases of BIB, Pathoscope and Sigma are automatically generated form the genome assemblies present in the reference directory. For the other tools these must be provided separately. For StrainGR and Strainseeker these can be generated as described in the previous section. For the GOTTCHA tool, one of the databases can be downloaded from the author's website. Both estMOI and EVORhA use a single genome as reference which can be obtained from NCBI.

SLURM
On the INSY cluster it is possible to submit several SLURM jobs in parallel. These jobs should be packed in .sbatch files to define properties like the quality of service (QOS), amount of CPUs, amount of memory needed and the time limit.

Metagenomic properties
The experimental setup defines the combination of properties that is tested for the different tools. Per parameter a list of values can be given and for every combination the corresponding config files and sbatch files will be made.

Species
A list of species is defined for which the experimental setup should be run. In the "strainsofinterestdirectory", a folder for every species should exist containing genome files to make the metagenomic samples from. Additionally, such folders should exist in the reference folder with reference genomes (BIB, Sigma, Pathoscope). The genome assembly files should have an .fna or .fa extension for the framework to be able to detect these. Lastly, folders per species also should be put in the database folders of straingr, strainseeker, estmoi and evorha, containing their respective databases (in case these tools are tested).

Strains of interest
The samples can contain several numbers of strains in the metagenomic sample. The framework takes the genomes present in the "strainsofinterest" folder in alphabetic order. So, in the case of 4 strains of interest, the first 4 files with an extension .fna or .fa are taken for the metagenomic sample. It is therefore advised to prepend a number before the genome names (like "01_ecoli_Santai.fna") to make sure the right genomes are picked for your sample

Distribution and total coverage
The distribution of the strains in the metagenomic samples can be done in four profiles; even, ascending, even_incremental and ascending_incremental.

In the even distribution the strains are present in equal amounts. For the ascending distribution the abundance of every subsequent strain is half as big as the previous strain. For these distributions the "totalcoverage" defines the coverage of all strains present added up together.

For the even_incremental and ascending_incremental distributions, the ratios of the different strains are the same as the non-incremental variants described above. However, here the “totalcoverage” is used as the coverage of the first strain (Strain-1). The coverages of the other strains are deduced from this initial amount. This results in that the sum of the coverages for even_incremental and ascending_incremental with a certain set total_coverage is different when there are more strains included. In case of the even and ascending distributions, this total coverage is equal to the sum of the individual coverages of the strains. In these cases the “totalcoverage” is not equal to the sum of the individual coverages of the strains. The term total coverage is however used due to legacy reasons.

An example of the distributions with a totalcoverage parameter of 2.0 can be found in the table below;

Strain #	even	ascending	even_incremental	ascending_incremental
Strain-1	0.5	1.067	**2.0**	**2.0**
Strain-2	0.5	0.533	2.0	1.0
Strain-3	0.5	0.267	2.0	0.5
Strain-4	0.5	0.133	2.0	0.25
Sum of coverages	**2.0**	**2.0**	8.0	3.75

Table 9 Example of the distribution of 4 strains with a total coverage set at 2.0 (bold)

HMP background

Within the experimental setup is it possible to spike the metagenomic samples with background reads. These can be paired end reads from the Human Microbiome Project (HMP). When used, the additional HMP reads will be added proportional to the amount of reads used for making the metagenomic strain samples. The amount of reads that will be added is proportional to the HMP factor; For example, when you use an HMP value of 5 and there are 10.000 read pairs made from the strains, 50.000 readpairs of the HMP samples will be added in addition. This results in a total of 60.000 read pairs.

Making config and sbatch files

Once you have defined your experimental setup, the appropriate config files and sbatch can be created with the experimental_setup.py script. These config files can be run in a regular way as described in the running single configs-section. The .sbatch files contain information of making SLURM jobs from these config files and can be run on the INSY cluster.

With the --tool parameter you should define one or multiple tools for which you want to run this experimental setup. These sbatch and config files are generated with the command (taking the tool sigma and straingr as example);

```
python3 /path/to/bep_mixedbugs/main/experimental_setup.py --workDirectory
/path/to/workDirectory --make_sbatch_files /path/to/experimental_setup.cfg --tool sigma --tool
straingr
```

The config files and sbatch files for making batch ART reads and the subsample configs for making the specific metagenomic samples will automatically be generated. When you do not wish to do so, it is possible to skip these steps with either the --no_art and/or --no_subsample flags (this could be the case if you already made these files).

The config files are created in the output directory specified in the experimental setup, in a separate folder per tool used. Additionally a folder named sbatch_configs is made in the workdirectory. In this folder you can find the sbatch files needed to create SLURM jobs on the cluster. Additionally, there is a file sbatch_file_list per tool which lists all the sbatch files. Such list is also present for all config files in the folder all_config_lists. Both of these list-files can be used to run the experimental setup per tool as described in the next section.

Running config and sbatch lists

You can choose to run the config files either sequentially in the command line or run as SLURM jobs on the INSY cluster.

To run a config list in the command line, the following command can be used;

```
python3 /path/to/bep_mixedbugs/main/experimental_setup.py --workDirectory
/path/to/workDirectory --run_config_files /path/to/all_config_lists/tool_config_list.cfg
```

This command runs all configs in the list file sequentially. This is useful when working in the Docker image. When working on the INSY cluster it is more efficient to run these jobs in parallel as SLURM jobs.

To start a list of sbatch files as SLURM jobs, the following command can be used;

```
python3 /path/to/bep_mixedbugs/main/experimental_setup.py --workDirectory
/path/to/workDirectory --run_sbatch_files
/path/to/sbatch_configs/tool_sbatch_files/sbatch_file_list
```

When starting jobs it should be taken into account that not all steps of the experimental setup can be started at the same time. First, all ART jobs should be finished before one can use the output of the ART jobs for the subsample step. The same holds when one wants to run the metagenomic tools on the readsets.

The output of the SLURM jobs are put into the same folder as the location of the .sbatch files. The job-id is put into the filename. Once a config is finished running, the start and endtime of the run is saved in a .cfg.log file at the same location as the .cfg file used. This file can be used to get insight into the runtime of the tools.

Additional scripts

In addition to the general framework for making synthetic reads and running these on the tools, additional scripts have been made for the creation of the desired input or for output analysis.

Extract results and timing

The extract_results.py script is used to make a summary of the outputs per tool. The --result_folder parameter should contain separate subfolders per experiment (which each contain a standardized_output.txt file). This scripts analyses the output of the individual experiments and compares it with the files in expected results. The output file contains a list of the experiments found in the results folder including information about true/false positives and f1 scores.

```
python3 /path/to/main/scripts/extract_results.py --result_folder /path/to/outdirectory/tool --
expected_folder /path/to/expected_strains --output_folder /path/to/summaryfolder
```

The expected strains folder should contain an <species>.txt file per species. Herein the expected strains in a tab delimited format are stated. An example is given below for ecoli.txt;

```
01	O157-H16_Santai
02	H7
03	FORC_028
04	O157-H7_FRIK2455
05	D8
06	S1
07	0127-H6_E2348-69
08	M10
```

Similarly the extract_timing.py script was used to analyze the runtime of the tools. The total runtime is saved in a .cfg.log file at the same place the .cfg file is located when running with the config file. The start and endtimes per tool can be extracted with this script and put in a list similar to the extract_results.py script.

Mutate

In our research we made mutated variants of out genomes to see whether this influenced the performance of the tools. This was done using the mutate.py script. This script takes a directory with genomes we want to mutate and makes a user defined amount of mutations in the genome. The new mutated genomes are saved under a new filename indicating the amount of mutations added.

Split HMP

The original HMP reads we used as background data were paired end fasta reads in one single file. To be able to use these reads for the framework we needed to split these reads over two files (one for every part of a pair) and add quality information. The split_hmp.py script can convert this file to the paired end read files needed.

Shear NCBI tree

For choosing strains uniformly from different clades of a certain species we extracted taxonomic trees in newick format of each species of interest from the NCBI website. However, these trees contain a lot more strains than we have in our database. With this script we can exclude the strains from the downloaded tree that are not of interest. The resulting tree only contained strains also present in our database.

Appendix F Ethical discussion of the project

The most important part of a benchmark is that each tool is compared fairly and that the results are an accurate representation of the relative performance of the different tools. We ran the tools with their default settings unless the tool was not able to analyze the data with these default settings. In cases we had doubts on how to run the programs we contacted the authors of the tools for advice. We discussed the situations that we were not able to run the tools with their optimal settings in the discussion of the results.

The Delft Bioinformatics Lab, our client, is a partner in the development of StrainGR. As this could potentially affect our ability to do an independent benchmark we ensured that certain safeguards were in place to guarantee the independence of our research.

Firstly, we were responsible for determining the settings with which every tool was run. At times we received feedback on how to prepare the data for the individual tools, but we would always run the tools in all ways that we discussed and that we thought could be potentially be appropriate for the tool. For the actual benchmark we used the procedures that resulted in the best results for each tool.

Secondly, we decided which experiments would be run on the tools. Our client and coach only provided us with feedback on how we could improve our experimental setup and how we could add control groups to our experiments.

Appendix G Info Sheet

Title: Classification of diverse bacterial populations

Name of the client organization: Delft Bioinformatics Lab, TU Delft

Data of the final presentation: February 16, 2017

Description:

When a patient suffers from an infection by multiple strains of a certain bacterial species, providing accurate treatment is a challenge. Metagenomic analysis techniques can provide a platform for mixed infection detection. We present an independent benchmark to compare the performance of the most common tools for metagenomic analysis at the strain level. We investigated the effect of properties like sample size and the coverage depth on the quality of the output. In order to do this benchmark, we developed a pipeline that can easily run the common tools for metagenomic analysis.

We used an agile approach to implement the pipeline in order to effectively respond to changing requirements and problems that we could potentially encounter in implementing the individual tools into our pipeline. The final product that we developed is a pipeline that is able to run the common tools for metagenomic analysis. The user does not have to install any dependencies himself. The system has been manually tested by running hundreds of experiments and validating the results.

Members of the project team

Name: Yorick de Vries
Interests: Biotechnology, Big Data
Contributions and role: Pipeline design/implementation, Tool analysis/implementation, Experimental setup, Data analysis

Name: Jasper Uljee
Interests: Machine learning, Natural Language processing
Contributions and role: Pipeline design/implementation, Tool analysis/implementation, Experimental setup, Data analysis

Coach:
Name: Dr. Thomas Abeel
Affiliation: Delft Bioinformatics Lab, TU Delft

Client
Name: Christine Anyansi
Affiliation: Delft Bioinformatics Lab, TU Delft

Contacts:
Yorick de Vries: yorickdevries@live.com
Jasper Uljee: uljee.jasper@gmail.com

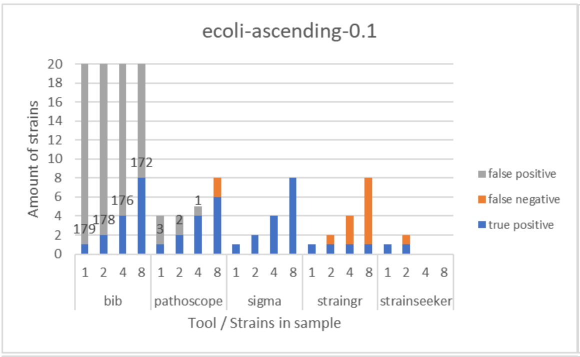
Appendix H Terminology

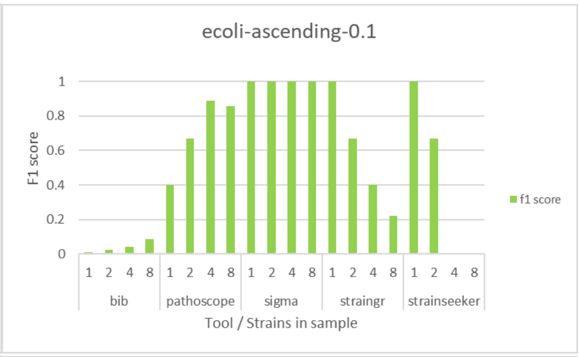
Term	Meaning
16S rRNA	Commonly used genetic marker for species identification
base	Basic building block of DNA, occurring in pairs, can be G, A, T or C
chromosome	DNA molecule with genetic material of an organism
clade	A group of closely related strains
contig	Continuous string of DNA
coverage	Degree to which a genome is on average covered by reads
fasta	De facto standard way of encoding genetic data
GC content	Fraction of G/C pairs in DNA
genome	Whole genetic information of a strain
genome assembly	All genetic information of an organism combined in one or several contigs
haplotype	A group of genes or SNPs which can be used to identify strains
HMP	Human microbiome project, contains large readsets from the human gut
Illumina	State of the art sequencing technique
k-mer	Fraction of DNA of “k” basepairs long
metagenomic analysis	Direct genetic analysis of genomes contained with an environmental sample
mutation	Change of the genetic content of a genome
paired end reads	Small DNA fragments read by sequencing technology. As DNA occurs in 2 strands, these can be made in pairs.
pathogen	Any biological entity that can cause a disease
readset	Batch of DNA fragments put out by a sequencing machine, generally in fasta format
sequencing	Extraction of DNA code from samples
SNP	Single nucleotide polymorphism, a single mutation of a base to another
species	Category under which a group of organisms can be classified
strain	Genetic variant of a bacterium
taxonomic tree	Tree which depicts the similarity between strains
whole genome sequencing	Process of obtaining the complete DNA content of a sample

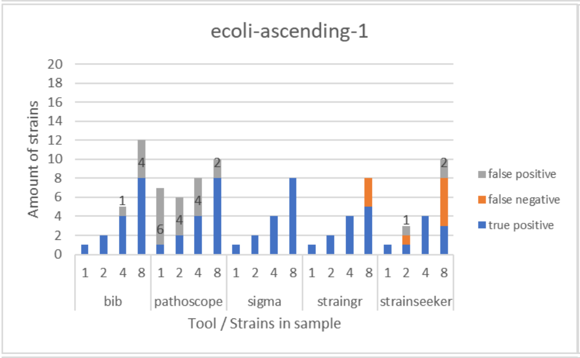
Appendix I Additional results: Mutated strains

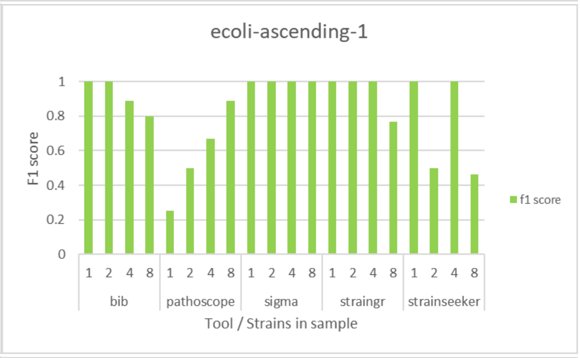
In this appendix you can find the additional results of the tools BIB, Pathoscope, Sigma, StrainGR and Strainseeker run with mutated variants of the strains. The species used are either *E. coli*, *Enterococcus* or *M . tuberculosis*. The samples were both evenly distributed or in an ascending way as indicated in the materials and methods section. The total coverages were fixed per experiment at either 0.1x, 1x, 10x or 100x total coverage.

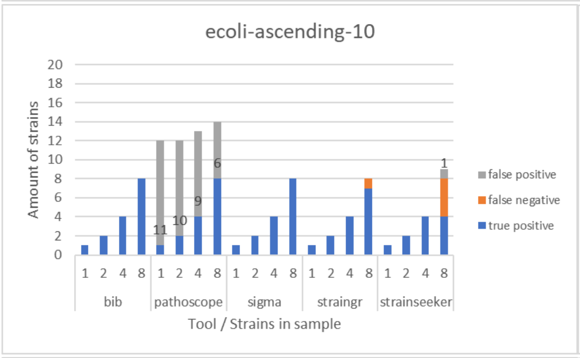
E. coli - ascending

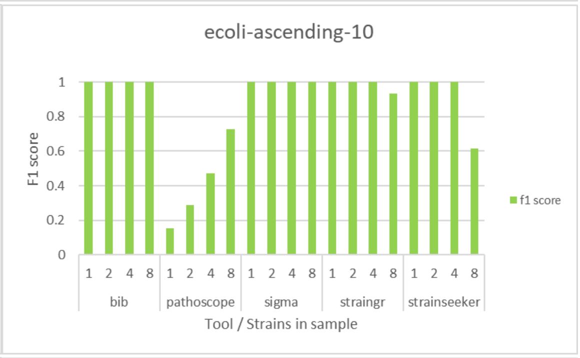


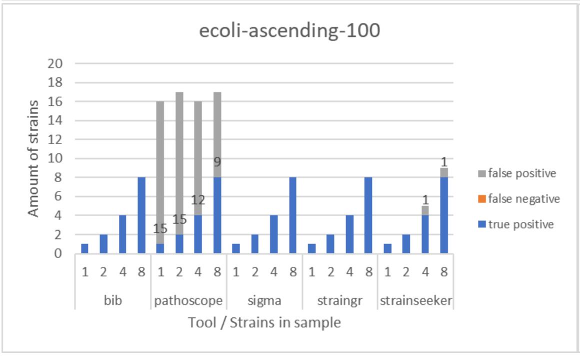


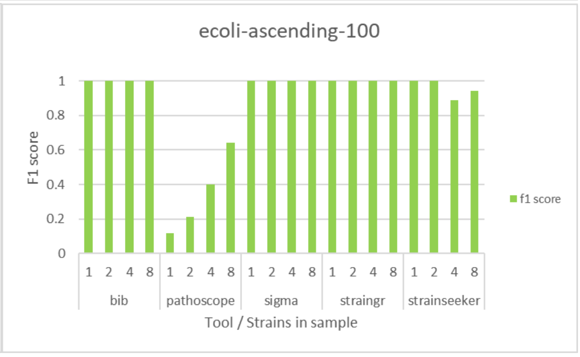




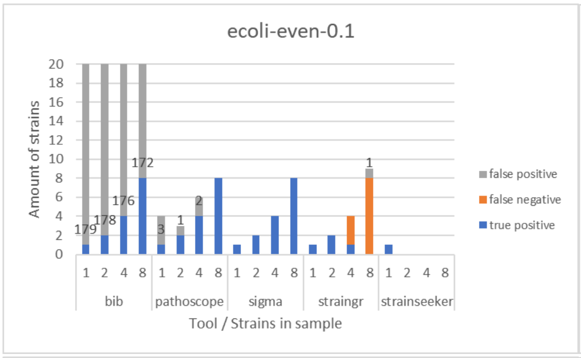


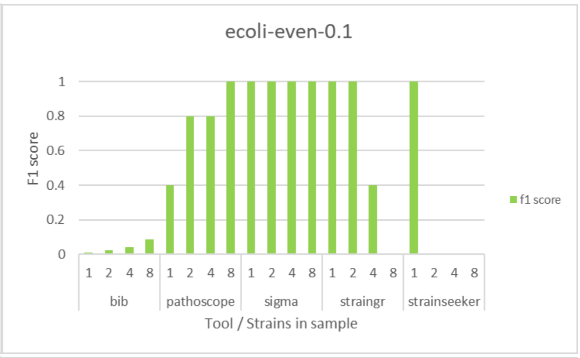


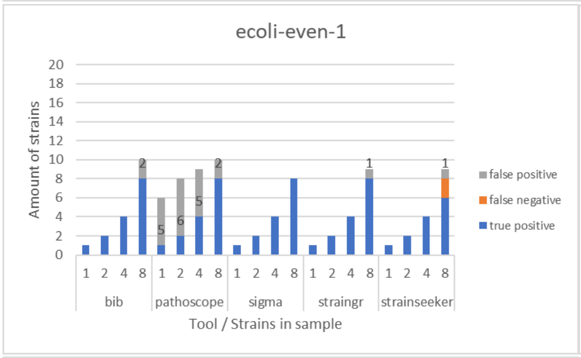


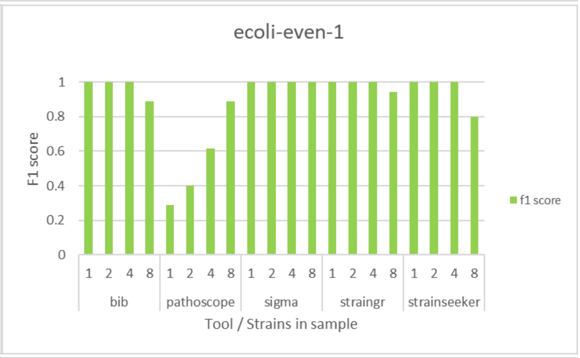


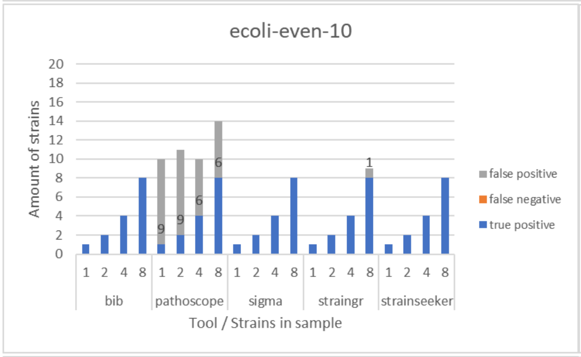
E. coli - even

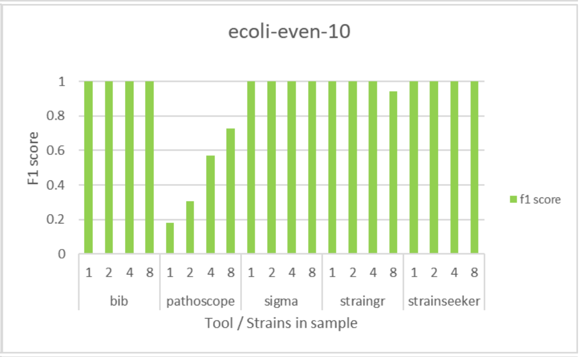

ecoli-even-0.1
Amount of strains
Tool / Strains in sample
bib
pathoscope
sigma
straingr
strainseeker
179
178
176
172
3
1
2
1
1
false positive
false negative
true positive

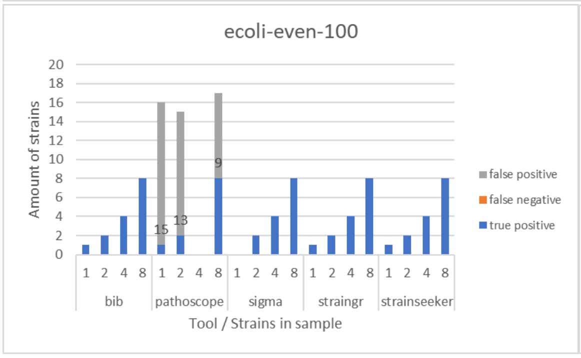

ecoli-even-0.1
F1 score
Tool / Strains in sample
bib
pathoscope
sigma
straingr
strainseeker
f1 score

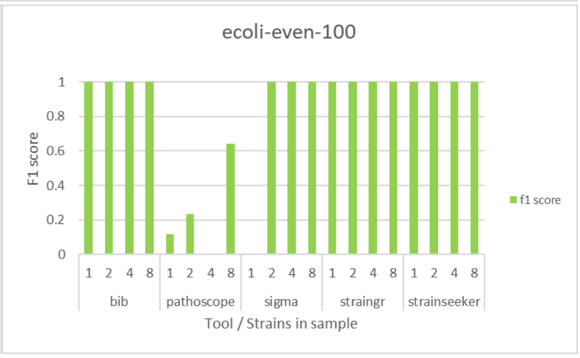

ecoli-even-1
Amount of strains
Tool / Strains in sample
bib
pathoscope
sigma
straingr
strainseeker
2
5
6
5
2
1
1
false positive
false negative
true positive


ecoli-even-1
F1 score
Tool / Strains in sample
bib
pathoscope
sigma
straingr
strainseeker
f1 score

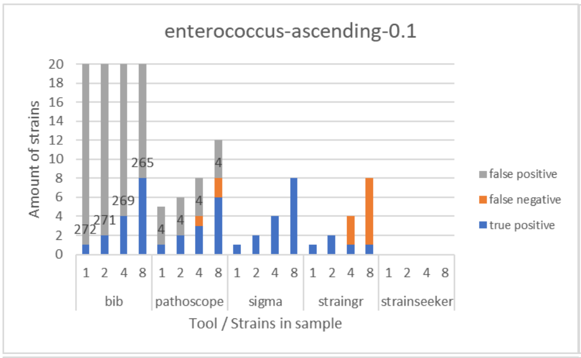

ecoli-even-10
Amount of strains
Tool / Strains in sample
bib
pathoscope
sigma
straingr
strainseeker
9
9
6
6
1
false positive
false negative
true positive

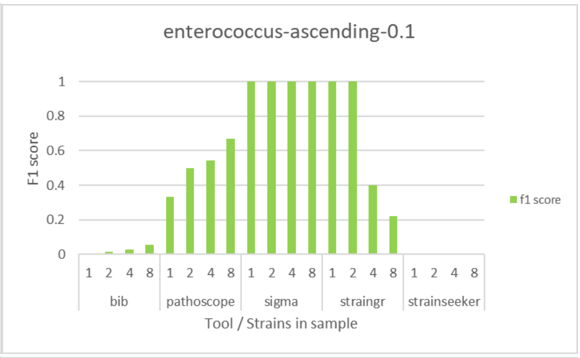

ecoli-even-10
F1 score
Tool / Strains in sample
bib
pathoscope
sigma
straingr
strainseeker
f1 score

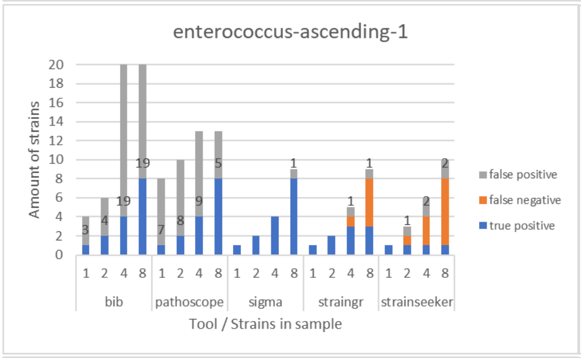

ecoli-even-100
Amount of strains
Tool / Strains in sample
bib
pathoscope
sigma
straingr
strainseeker
15
13
9
false positive
false negative
true positive

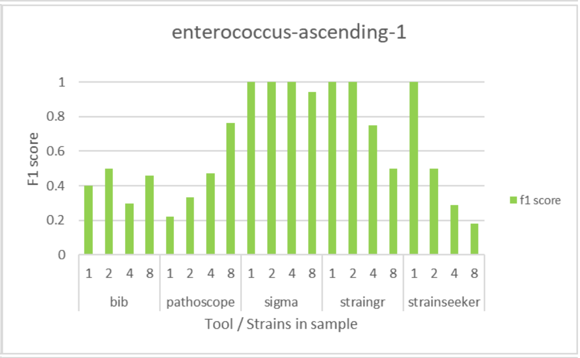

ecoli-even-100
F1 score
Tool / Strains in sample
bib
pathoscope
sigma
straingr
strainseeker
f1 score

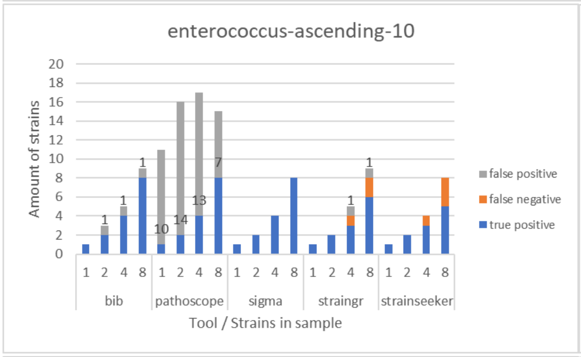
Enterococcus - ascending

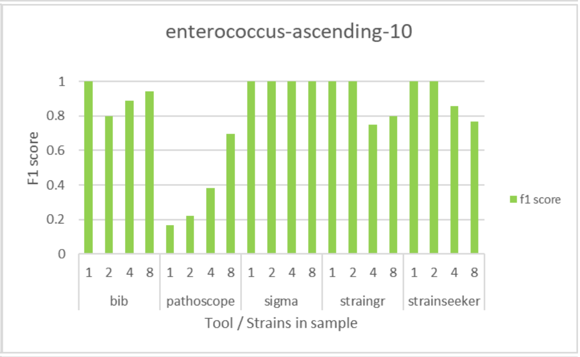


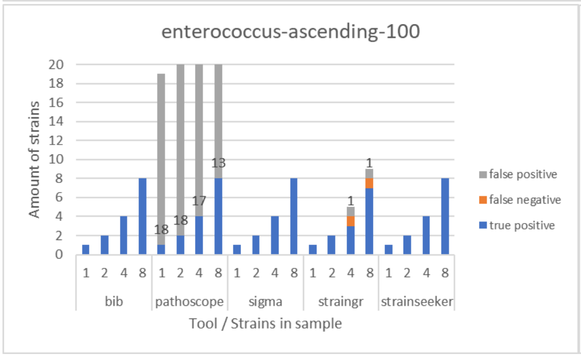


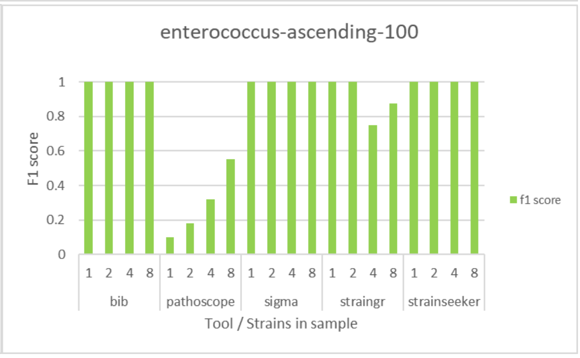




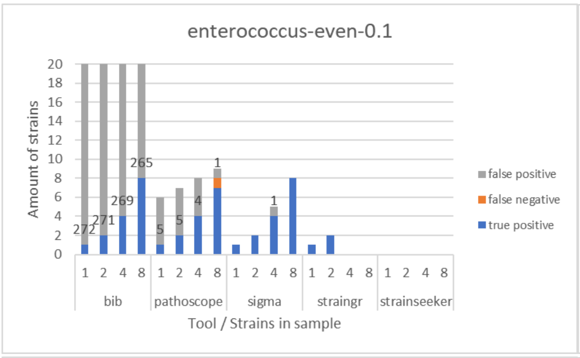


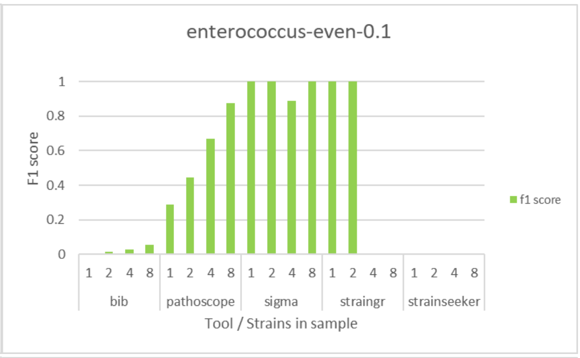


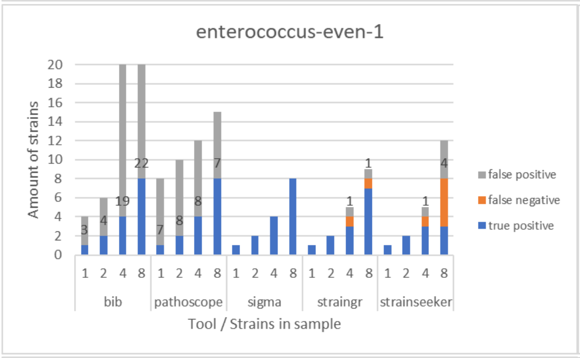


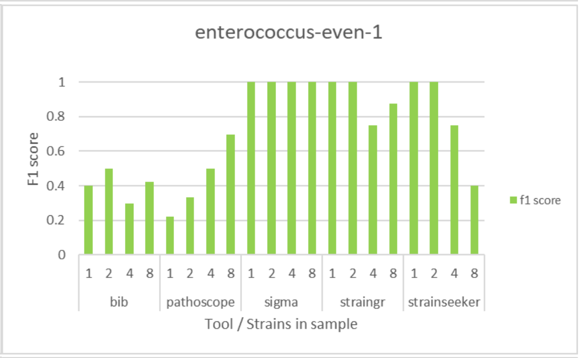


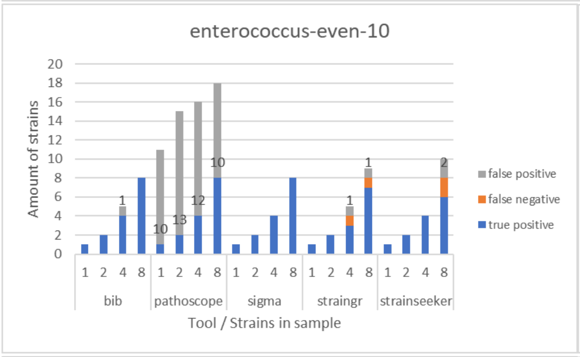
Enterococcus - even

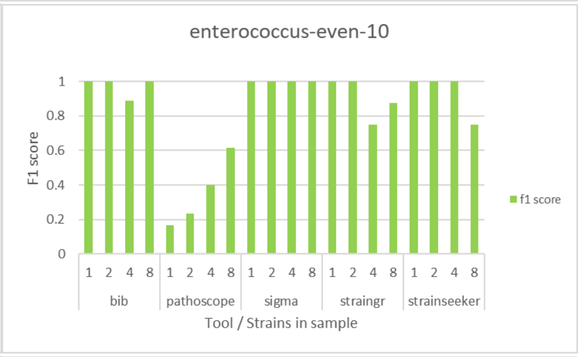


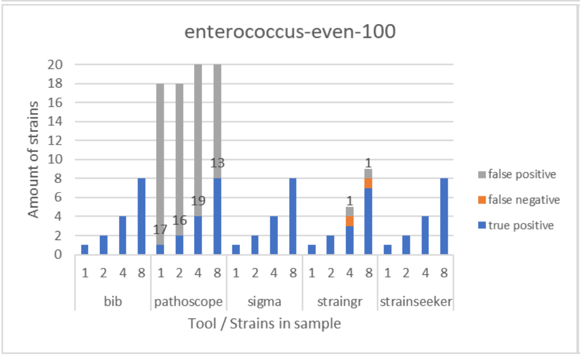


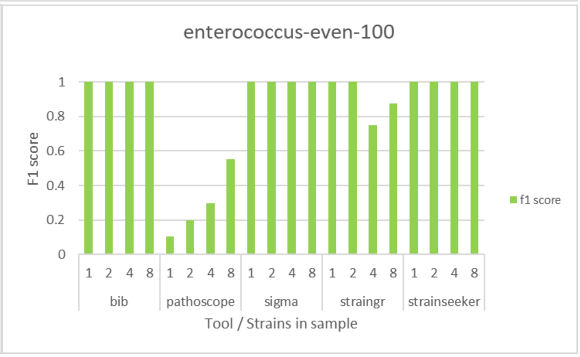




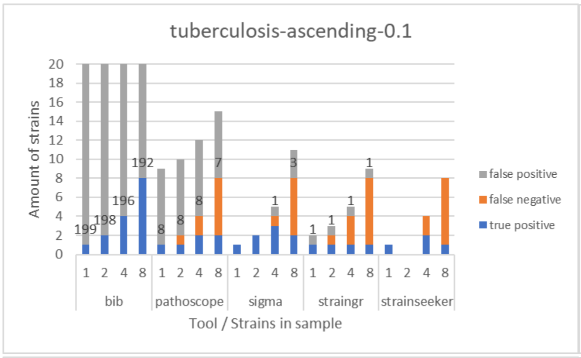


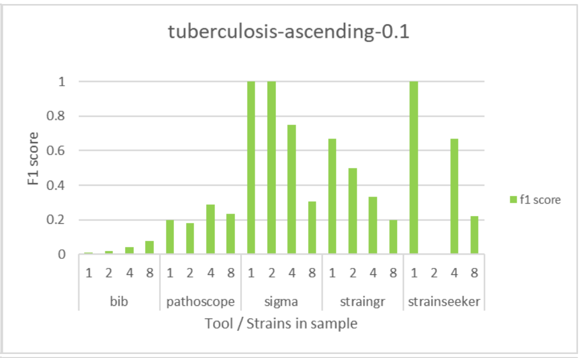


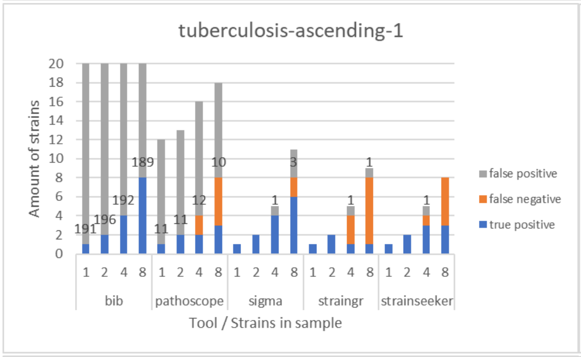


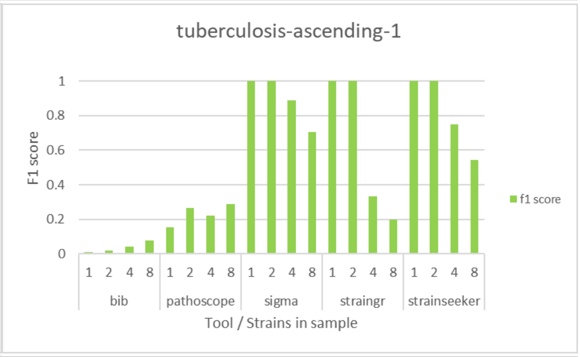


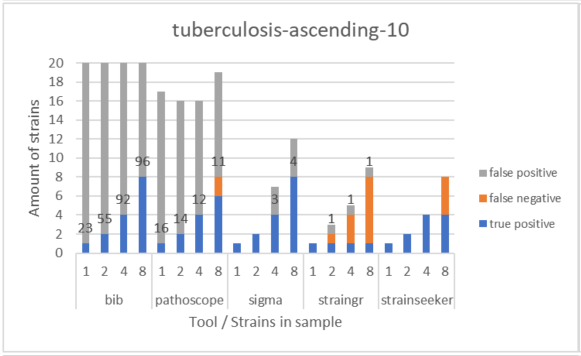
M. tuberculosis - ascending

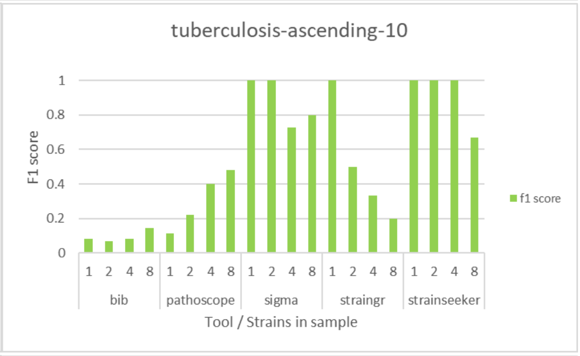


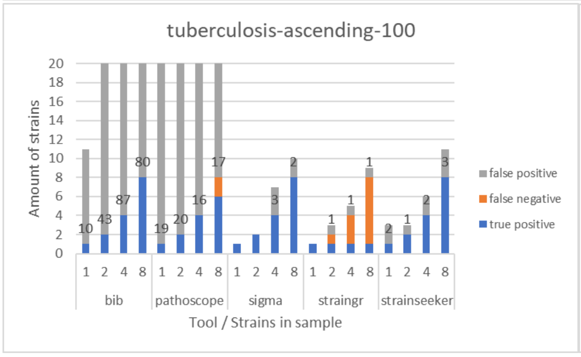


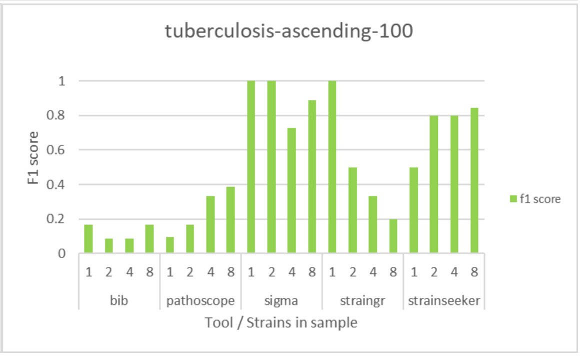




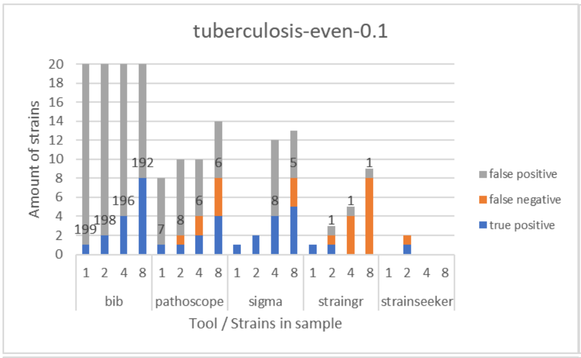


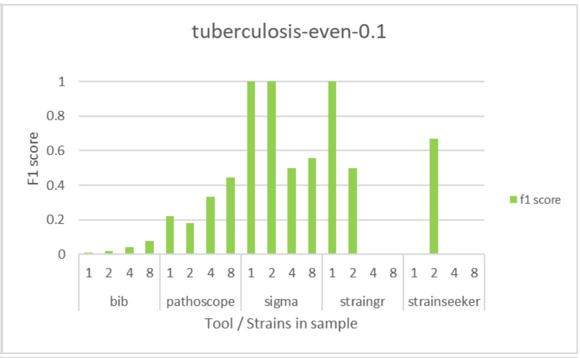


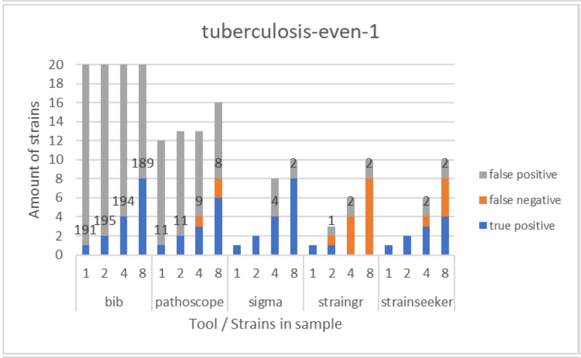


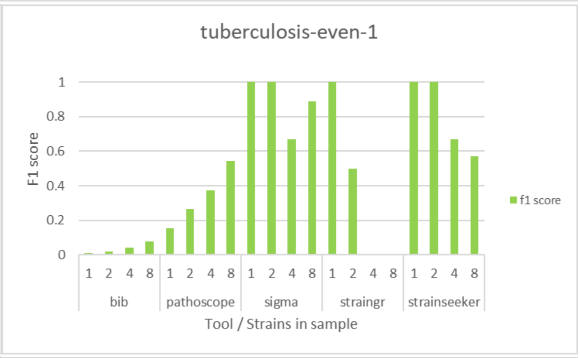


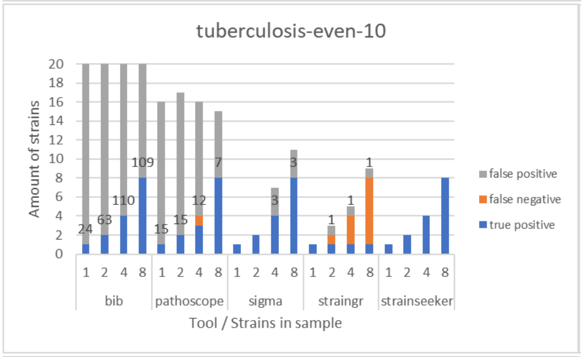
M. tuberculosis - even

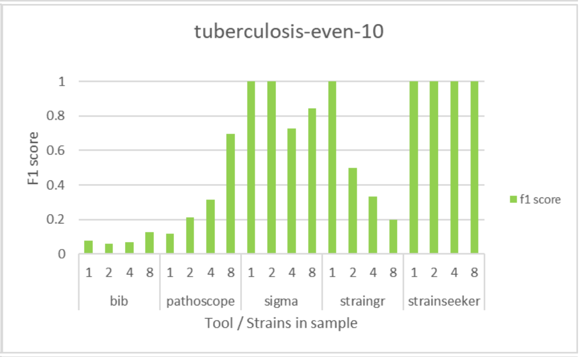


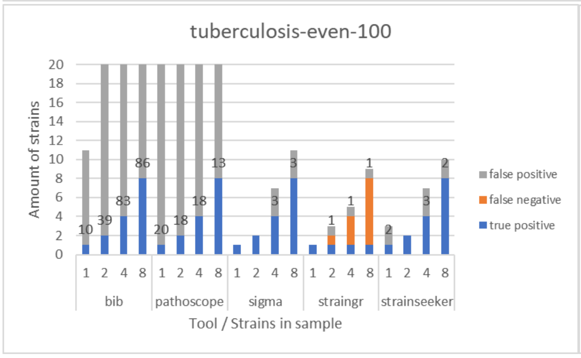


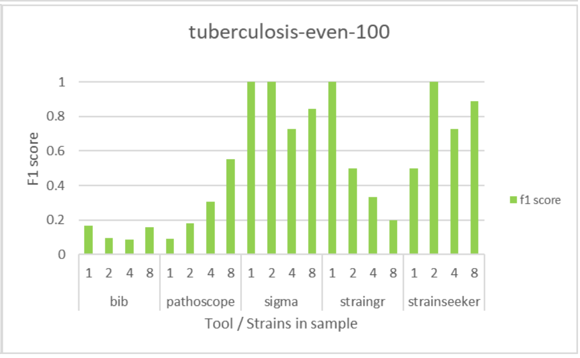












Appendix J Additional results: Total strains found per tool

In this appendix you can find the results of all the run with mutated variants of the strains. The species used are either *E. coli*, *Enterococcus* or *M. tuberculosis*. The samples were both incrementally evenly distributed or in an ascending way as indicated in the materials and methods. The base coverage used for the first strain is indicated in the tables.

BIB

Tool: bib **Coverage:** 0.1

	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	180	180	273	273	200	200
2	70	38	273	247	200	200
4	79	9	255	59	200	200
8	76	11	267	33	200	200

Tool: bib **Coverage:** 1

	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	6	4	191	192
2	2	2	6	4	185	172
4	4	4	17	10	193	129
8	8	8	22	10	185	125

Tool: bib **Coverage:** 10

	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	18	20
2	2	2	2	2	55	53
4	4	4	5	4	99	91
8	8	8	9	8	94	102

Constrains

Tool:	constrains	**Coverage:**	0.1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	0	1	1	1	1	2
2	1	1	1	2	2	2
4	1	1	2	2	1	1
8	1	1	2	2	2	1

Tool:	constrains	**Coverage:**	1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	1	1	2	2	1	1
4	1	1	2	2	1	1
8	1	1	2	2	1	1

Tool:	constrains	**Coverage:**	10			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	0	0	1	1
2	2	2	1	1	1	1
4	2	2	3	4	1	1
8	2	2	3	4	1	1

EstMOI

Tool:	estmoi	**Coverage:**	0.1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	0	0	0	0	0	0
2	0	0	0	0	0	0
4	0	0	0	0	0	0
8	0	0	0	0	0	0

Tool:	estmoi	**Coverage:**	1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	0	0	0	0	0	0
2	0	0	0	0	0	0
4	0	1	0	0	0	0
8	0	1	0	1	0	0

Tool:	estmoi	**Coverage:**	10			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	1	1	1	1	1	1
4	1	1	1	1	2	2
8	1	2	1	1	1	2

EVORhA

Tool:	evorha	**Coverage:**	0.1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	2	0	0	0	0	0
2	2	0	0	0	0	2
4	2	3	0	0	0	2
8	2	2	0	0	0	2

Tool:	evorha	**Coverage:**	1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	2	2	0	0	0	0
2	2	3	0	0	3	2
4	3	5	0	0	2	4
8	4	4	0	0	2	5

Tool:	evorha	**Coverage:**	10			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	3	4	0	0	4	4
2	4	6	0	0	4	4
4	5	7	0	0	5	5
8	5	8	0	0	5	4

GOTTCHA

Tool:	gottcha	**Coverage:**	0.1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	0	3	3	0	0
2	1	1	3	4	0	0
4	4	3	4	4	0	0
8	4	6	4	6	0	0

Tool:	gottcha	**Coverage:**	1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	5	5	6	5	0	0
2	8	11	5	5	0	0
4	11	20	8	7	0	0
8	13	26	6	8	0	0

Tool:	gottcha	**Coverage:**	10			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	12	12	6	6	0	0
2	22	22	6	6	0	0
4	27	33	7	8	0	0
8	30	45	7	11	0	1

Pathoscope

Tool:	pathoscope	**Coverage:**	0.1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	6	5	5	6	9	9
2	6	5	6	7	8	11
4	6	6	7	11	11	12
8	8	9	10	12	10	11

Tool:	pathoscope	**Coverage:**	1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	8	8	7	9	13	11
2	8	8	11	9	14	14
4	10	10	11	16	16	13
8	11	12	15	18	13	16

Tool:	pathoscope	**Coverage:**	10			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	11	10	11	12	17	16
2	13	12	16	17	15	17
4	15	15	17	18	16	18
8	14	20	18	21	18	20

Sigma

Tool:	sigma	**Coverage:**	0.1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	2	3	2	2	2	2
4	4	4	4	4	9	7
8	8	8	10	9	7	10

Tool:	sigma	**Coverage:**	1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	2	2	2	2	2	2
4	4	4	4	4	7	6
8	8	8	9	8	9	12

Tool:	sigma	**Coverage:**	10			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	2	2	2	2	2	2
4	4	4	4	4	7	6
8	8	8	8	8	10	11

StrainGR

Tool:	straingr	**Coverage:**	0.1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	2	2	2	2	2	2
4	3	4	2	4	2	2
8	2	9	2	8	2	2

Tool:	straingr	**Coverage:**	1			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	2	2	2	2	2	3
4	4	4	4	4	2	2
8	5	9	5	8	2	2

Tool:	straingr	**Coverage:**	10			
	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	2	2	2	2	2	2
4	4	4	4	4	2	2
8	7	8	7	8	2	2

Strainseeker

Tool: strainseeker **Coverage:** 0.1

	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	0	1	1	1
2	1	2	1	0	1	2
4	0	4	0	0	1	4
8	2	0	0	7	0	7

Tool: strainseeker **Coverage:** 1

	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	2	2	2	2	2	2
4	4	4	3	4	4	4
8	5	8	3	9	4	7

Tool: strainseeker **Coverage:** 10

	E. coli		*Enterococcus*		*M. tuberculosis*	
SOI	ascending	even	ascending	even	ascending	even
1	1	1	1	1	1	1
2	2	2	2	2	2	2
4	4	4	4	4	4	4
8	6	8	5	8	5	8

Appendix K Implementation

In this appendix you can find additional information on how the tools were implemented in our pipeline.

Dependencies

The tools use several dependencies to work properly. The specific versions used in this research were;

- samtools 1.6 (samtools 1.2 for Constrains)
- bowtie2-2.3.3.1 (bowtie2-2.3.2-legacy for Pathoscope)
- bwa-0.7.17
- bcftools-1.6
- MetaPhlAn version 2.7.0 (07 November 2017)

Bam files

The tools estMOI and Evorha require aligned bam files as input for their algorithm. These files were made using BWA, Samtools and BCFTools as follows;

- The reference genomes were indexed with BWA with: bwa index -a bwtsw
- A .sam file was made with: bwa mem
- This .sam file was converted to a .bam file by running: samtools view -b –T
- For Evorha the .bam file was sorted with: samtools sort

A .vcf file required for estMOI to run was created by running “bcftools mpileup -Ob" and subsequentely “bcftools call –vmO z” as described in (http://www.htslib.org/workflow/)

StrainSeeker

In order to create a database for StrainSeeker we first create a guide tree by generating a distance matrix for all genomes using Mash. Mash is a tool that can estimate the distance between two genomes. We do this using the dist function of mash, which returns the estimated distance between the two genomes that are given as arguments. We call dist on every genome pair, as such: `mash dist genome1.fa genome2.fa`. These distances are then combined in a lower triangular matrix. This matrix is then used as input for Mega, which creates taxonomic tree based on this matrix using an agglomerative (bottom-up) hierarchical clustering method (UPGMA). This tree is then given to StrainSeeker, which creates a k-mer database from all the reference genomes and this tree. This process only has to be done once. After a database has been created, one can compare a metagenomic read set to the database and determine whether the sample contains strains that are also in the database.

PathoScope

PathoScope consists of multiple individual modules that can be run individually. You first have to prepare a library using the LIB module. This module searches on NCBI for the taxonomic index based on the GI number of the genome. This module is, however, outdated as NCBI no longer uses GI numbers. This LIB module has been replaced by a script that was posted by one of the authors of the tool on their GitHub repository to find the taxonomic indexes. The end result of this module is a multifasta containing all the reference genomes with all the required metadata. Once a library has been prepared, the MAP module is used to the map the read set to the library. This module uses bowtie2 to align the reads to the genomes in the library and outputs a Sequence Alignment Map (SAM) file. This SAM contains the information of which parts of the read set can be aligned to one or more of the reference genomes. This SAM file is then given as input to the ID module, which ids strains from this alignment file. Lastly, one can call the REP module in order to generate a report of the strains that were identified.

StrainGR

StrainGR uses a k-mer database to analyze readsets. This database is generated by first kmerizing all the reference genome. In order to reduce the size of the database highly similar k-mers are clustered together. Once the reference k-mers are clustered together the database can be build. Once the user has a database, he can kmerize the read set using strainge kmerize again. The kmerized read set and the database are then used as input to the analysis tool strainGST.

Appendix L Strain databases

The strain genomes used to make the databases for the reference based tools are shown in the table below. The amount of contigs every strain consisted of is shown between brackets

Enterococcus		*E. coli*		*M. tuberculosis*	
12030 (4)	E1321 (22)	0127-H6_E2348-69 (1)	JJ1887 (1)	00-10219 (3)	I0003229-7 (1)
12107 (9)	E1392 (39)	042 (1)	K-12_substr_MDS42 (1)	00-R0434 (8)	I0003246-1 (3)
1448E03 (6)	E1552 (6)	06-00048 (1)	K-12_substr_MG1655 (1)	00-R1097 (2)	I0003398-0 (15)
14EA1 (4)	E1573 (9)	08-00022 (1)	K-15KW01 (1)	00-R1211 (15)	I0003461-6 (7)
16EA1 (22)	E1574 (18)	09-00049 (1)	KO11 (1)	01-R0165 (7)	I0003518-3 (18)
173EA1 (19)	E1575 (19)	1303 (1)	KTE135 (2)	01-R0186 (13)	I0003525-8 (10)
177EA1 (4)	E1576 (36)	1409160003 (4)	KTE147 (3)	01-R0446 (9)	I0003529-0 (5)
182970 (14)	E1578 (3)	2009C-3133 (1)	KTE154 (3)	01-R0712 (5)	I0003707-2 (15)
182EA1 (14)	E1590 (14)	2011C-3198 (4)	KTE161 (3)	01-R0774 (10)	I0003813-8 (9)
19116 (5)	E1604 (5)	2011C-3911 (1)	KTE165 (3)	01-R0908 (9)	I0003888-0 (15)
1_141_733 (25)	E1613 (11)	2012C-4227 (1)	KTE17 (3)	01-R0919 (16)	I0003889-8 (17)
1_230_933 (85)	E1620 (5)	2013C-4465 (1)	KTE196 (4)	01-R0956 (11)	I0004105-8 (18)
1_231_408 (79)	E1622 (6)	2014C-3250 (3)	KTE233 (3)	01-R1141 (2)	I0004188-4 (8)
1_231_410 (69)	E1623 (22)	2016C-3936C1 (1)	KTE29 (3)	01-R1275 (10)	I0004216-3 (14)
1_231_501 (25)	E1626 (43)	210205630 (1)	KTE45 (3)	01-R1278 (14)	I0004240-3 (12)
1_231_502 (61)	E1627 (22)	210221272 (1)	KTE79 (2)	01-R1302 (4)	I0004445-8 (12)
205EA1 (9)	E1630 (21)	536 (1)	KTE84 (3)	01-R1309 (3)	I0004509-1 (8)
209EA1 (8)	E1634 (6)	5363_plasmid_p53638_226	KTE9 (4)	01-R1379 (10)	I0004564-6 (17)
210AEA1 (13)	E1636 (223)	(4)	M1 (1)	01-R1554 (11)	I0004625-5 (3)
213EA1 (4)	E1644 (21)	55989 (1)	M10 (1)	01-R1568 (14)	I0004744-4 (7)
234EA1 (20)	E1679 (340)	6409 (1)	M18 (1)	01-R1591 (9)	I0005268-3 (9)
236EA1 (4)	E1731 (23)	789 (1)	M19 (1)	02-R0027 (5)	I0005322-8 (16)
248EA1 (6)	E1861 (8)	94-3024 (1)	M8 (1)	02-R0113 (17)	I0005324-4 (16)
250AEA1 (5)	E1904 (34)	AA86_plasmid_pAA86L_53_5	MGH108 (2)	02-R0220 (10)	I0005430-9 (3)
257EA1 (16)	E1972 (7)	(5)	MRE600 (1)	02-R0244 (10)	I0005546-2 (7)
261EA1 (8)	E1Sol (14)	ABU_83972 (1)	MRSN346355 (1)	02-R0358 (9)	I0005710-4 (17)
262EA1 (8)	E2039 (11)	ACN001 (1)	MRSN346647 (1)	02-R0459 (5)	I0005726-0 (4)
2630V05 (17)	E2071 (14)	ACN002 (1)	MS6198 (1)	02-R0755 (13)	INS_SEN (18)
263EA1 (13)	E2134 (6)	APEC_IMT5155 (1)	NGF1 (1)	02-R0924 (6)	KT-0017 (13)
26EA1 (20)	E2297 (19)	APEC_O1 (1)	Nissle_1917 (1)	02-R0951 (15)	KT-0063 (15)
270AEA1 (44)	E2369 (21)	APEC_O78 (1)	O103-H2_12009 (1)	02-R0987 (3)	KT-0078 (3)
277EA1 (3)	E2560 (27)	ATCC_25922 (1)	O111-H-_11128 (1)	02-R0990 (1)	KZN_R506 (1)
2924 (11)	E2620 (4)	ATCC_8739 (1)	O145-H28_RM12581 (1)	02-R1017 (8)	KZN_V2475 (1)
294EA1 (9)	E2883 (7)	B7A (5)	O145-H28_RM13516 (1)	02-R1063 (5)	M0000639-6 (6)
297EA1 (4)	E3083 (6)	BIDMC_59 (2)	O157-H16_Santai (1)	02-R1131 (6)	M0000956-4 (1)
29EA1 (2)	E3346 (16)	BL21-DE3- (1)	O157-H7_FRIK2455 (1)	02-R1154 (2)	M0003138-6 (10)
2EA1 (9)	E3548 (5)	C1 (1)	O177-H21 (1)	02-R1534 (13)	M0005676-3 (1)
300AEA1 (2)	E4215 (19)	C10 (1)	O26-H11_11368 (1)	02-R1625 (12)	M0008635-6 (6)
300BEA1 (18)	E4389 (16)	C2 (1)	O55-H7_CB9615 (1)	02-R1669 (3)	M0009182-8 (1)
302EA1 (7)	E4452 (268)	C3 (1)	O55-H7_RM12579 (1)	02-R1705 (15)	M0010396-1 (2)
304EA1 (14)	E4453 (374)	C4 (1)	O7-K1_CE10 (1)	02-R1752 (7)	M0012491-8 (3)
311EA1 (9)	E6012 (55)	C5 (1)	O83-H1_NRG_857C (1)	03-R0023 (10)	M0013032-9 (1)
312EA1 (7)	E6045 (33)	C8 (1)	P12b (1)	03-R0290 (4)	M0013793-6 (3)
315EA1 (7)	E980 (131)	C9 (1)	PCN033 (1)	03-R0839 (13)	M0013935-3 (12)
316EA1 (17)	E99 (14)	CFSAN004177 (1)	PCN061 (1)	04-R0766 (12)	M0014480-9 (3)
34EA1 (14)	EnGen0253 (38)	CFSAN029787 (1)	RM9387 (1)	06-05801 (6)	M0014577-2 (13)
35EA1 (19)	F1 (11)	CFT073 (1)	S1 (1)	06-14433 (16)	M0015869-2 (6)
39-5 (25)	FA2-2 (4)	CI5 (1)	S10 (1)	06-R0222 (13)	M0018310-4 (5)
39EA1 (1)	Fly1 (12)	clone_D_i2 (1)	S21 (1)	0FXR-23 (3)	M0020319-1 (3)
46EA1 (7)	Fly2 (8)	Co6114 (9)	S3 (1)	22103 (1)	M0020865-3 (4)
52EA1 (2)	HEF39 (10)	D1 (1)	S30 (1)	96075 (1)	M0021672-2 (1)
54EA1 (10)	HH22 (36)	D10 (1)	S40 (1)	99-02018 (11)	M0022470-0 (1)
55EA1 (8)	HIP11704 (38)	D2 (1)	S42 (1)	99-27860 (6)	M0022539-2 (10)
5952 (13)	JH1 (24)	D3 (1)	S43 (1)	99-R1224 (2)	MAL010109 (8)
599951 (8)	JH2-2 (2)	D5 (1)	S50 (1)	99-R537 (19)	MAL010133 (14)
60BEA1 (11)	LCT-EF90 (9)	D6 (1)	S51 (1)	BTB05-552 (1)	MAL020192 (4)
61EA1 (9)	Merz151 (20)	D7 (1)	S56 (1)	BTB07-283 (5)	MAL020200 (3)
67EA1 (7)	Merz192 (17)	D8 (1)	Sanji (1)	BTB07-325 (2)	PanR0202 (1)
68EA1 (8)	Merz204 (8)	D9 (1)	SE11 (1)	BTB09-230 (10)	PanR0203 (1)
72EA1 (9)	Merz89 (27)	E24377A (1)	SE15 (1)	BTB10-253 (6)	PanR0208 (1)
7330082-2 (10)	Merz96 (21)	EC590 (1)	SEC470 (1)	BTB11-343 (4)	PanR0301 (1)
7330112-3 (19)	MMH594 (25)	ECC-1470 (1)	SF-088 (1)	C (4)	PanR0304 (1)
7330245-2 (14)	Ned10 (18)	Eco889 (1)	SF-166 (1)	C0000604-0 (10)	PanR0305 (1)
7330257-1 (4)	OG1RF (1)	Ecol_743 (1)	SF-173 (1)	C0000689-1 (15)	PanR0308 (1)
7330259-5 (8)	Pan7 (4)	ECONIH1 (1)	SF-468 (1)	C0006290-2 (13)	PanR0316 (1)
7330948-5 (7)	RC73 (21)	ECONIH2 (1)	SLK172 (1)	CAS-NITR204 (1)	PanR0317 (1)
7430275-3 (6)	RM3817 (12)	ED1a (1)	SMS-3-5 (1)	CCDC5079 (1)	PanR0412 (1)
7430315-3 (4)	RM4679 (11)	FAP1 (5)	ST2747 (1)	CTRI-2 (1)	PanR0503 (1)
7430416-3 (6)	RMC1 (22)	FMU073332 (1)	ST540 (1)	EAI5-NITR206 (1)	PanR0601 (1)
7430821-4 (7)	RMC5 (17)	FORC_028 (1)	ST648 (1)	EAI5 (1)	PanR0604 (1)
76EA1 (7)	RMC65 (7)	FORC_031 (1)	S_boydii_CDC_3083-94	Erdman (1)	PanR0611 (1)
79-3 (17)	SF100 (16)	FORC_041 (1)	(1)	H37Rv (1)	PanR0708 (1)
87EA1 (14)	SF105 (35)	G749 (1)	S_boydii_Sb227 (1)	I0000038-5 (18)	PanR0805 (1)
8EA1 (5)	SF1592 (20)	GB089 (1)	S_dysenteriae_Sd197 (1)	I0000071-6 (7)	PanR0902 (1)
91EA1 (4)	SF19 (21)	H1 (1)	S_flexneri_2a_981 (1)	I0000187-0 (12)	PanR0906 (1)
A-2-1 (5)	SF21520 (32)	H10 (1)	S_flexneri_5_8401 (1)	I0000205-0 (10)	PanR1007 (1)
A-3-1 (8)	SF21521 (53)	H15 (1)	S_flexneri_FDAARGOS_74	I0000236-5 (13)	PanR1101 (1)
AR01_DG (13)	SF24396 (3)	H1827-12 (1)	(2)	I0000517-8 (11)	PR08 (1)
ATCC4200 (14)	SF24397 (29)	H2 (1)	S_sonnei_53G (1)	I0000673-9 (9)	R1207 (1)
ATCC8459 (10)	SF24413 (27)	H3 (1)	S_sonnei_FORC_011 (1)	I0000808-1 (7)	TB_RSA130 (3)
ATCC_10100 (6)	SF26630 (13)	H6 (1)	UCD_JA65_pb (3)	I0000841-2 (17)	TB_RSA96 (6)
ATCC_19433 (3)	SF28073 (45)	H7 (1)	UMEA_3144-1 (3)	I0001262-0 (2)	TKK-01-0005 (5)
ATCC_27275 (8)	SF339 (13)	H8 (1)	UMEA_3682-1 (2)	I0001324-8 (9)	TKK-01-0016 (7)
ATCC_27959 (2)	SF350 (28)	HS (1)	UMN026 (1)	I0001355-2 (15)	TKK-01-0024 (9)
ATCC_29200 (15)	SF370 (19)	HUSEC2011 (1)	UMNF18 (6)	I0001406-3 (2)	TKK-01-0091 (3)

ATCC_35038 (10)	SF5039 (18)	HVH_147_-4-5893887- (3)	UMNK88 (1)	I0001498-0 (11)	TKK_02_0037 (5)
ATCC_6055 (31)	SF6375 (28)	HVH_195_-3-7155360- (2)	UPEC_26-1 (1)	I0001560-7 (8)	TKK_03_0044 (4)
Aus0004 (4)	SS-6 (6)	HVH_24_-4-5985145 (4)	VR50 (1)	I0001711-6 (15)	TKK_04_0023 (6)
B-4-111 (7)	SS-7 (9)	IAI1 (1)	WCHEC1613 (1)	I0001905-4 (7)	TKK_04_0034 (2)
B15725 (33)	T1 (16)	IAI39 (1)	Y5 (1)	I0002066-4 (10)	TKK_04_0125 (9)
B16457 (3)	T10 (4)	IHE3034 (1)	YD786 (1)	I0002107-6 (13)	UT0055 (8)
B56765 (33)	T11 (13)	isolate_NCTC86EC (1)	ZH063 (1)	I0002353-6 (1)	UT0070 (9)
B653 (5)	T12 (15)			I0002458-3 (8)	UT0088 (16)
B69486 (14)	T13 (3)			I0002531-7 (3)	UT0094 (6)
B84847 (4)	T14 (8)			I0002615-8 (9)	UT0100 (15)
BM4539 (5)	T15 (3)			I0002793-3 (11)	XTB13-086 (5)
BM4654 (23)	T16 (11)			I0002935-0 (7)	XTB13-089 (8)
C68 (64)	T17 (5)			I0003020-0 (4)	XTB13-115 (9)
CH116 (6)	T18 (5)			I0003165-3 (9)	XTB13-240 (10)
CH136 (20)	T19 (9)			I0003179-4 (16)	XTB13-247 (12)
CH188 (27)	T2 (22)				
CH19 (10)	T20 (4)				
CH570 (16)	T21 (4)				
Com12 (19)	T3 (10)				
Com15 (20)	T4 (5)				
Com_1 (8)	T5 (3)				
Com_2 (6)	T6 (13)				
Com_6 (3)	T7 (13)				
Com_7 (7)	T8 (24)				
C_19315_led_1A_WT (10)	T9 (3)				
C_19315_led_1b_pp_SCV (11)	TC6 (125)				
D1 (5)	TR161 (31)				
D173 (4)	TR197 (12)				
D3 (15)	U0317 (227)				
D344SRF (215)	UAA1014 (24)				
D6 (11)	UAA1180 (19)				
DIV0555 (38)	UAA1489 (33)				
DS16 (10)	UAA409pIP819 (11)				
DS5 (43)	UAA702 (10)				
E0045 (13)	UAA769 (10)				
E0120 (20)	UAA823 (26)				
E0164 (14)	UAA902 (9)				
E0269 (13)	UAA903 (9)				
E0333 (27)	UAA904 (8)				
E0679 (8)	UAA905 (9)				
E0680 (18)	UAA906 (8)				
E0688 (7)	UAA907 (21)				
E1 (16)	UAA943 (10)				
E1007 (6)	UAA948 (40)				
E1039 (124)	V583_GB1 (4)				
E1050 (9)	V583_V2 (9)				
E1071 (96)	V587 (25)				
E1133 (25)	WH245 (26)				
E1162 (139)	WH257 (34)				
E1185 (11)	WH571 (30)				
E1258 (8)	X98 (13)				
	YI6-1 (5)				

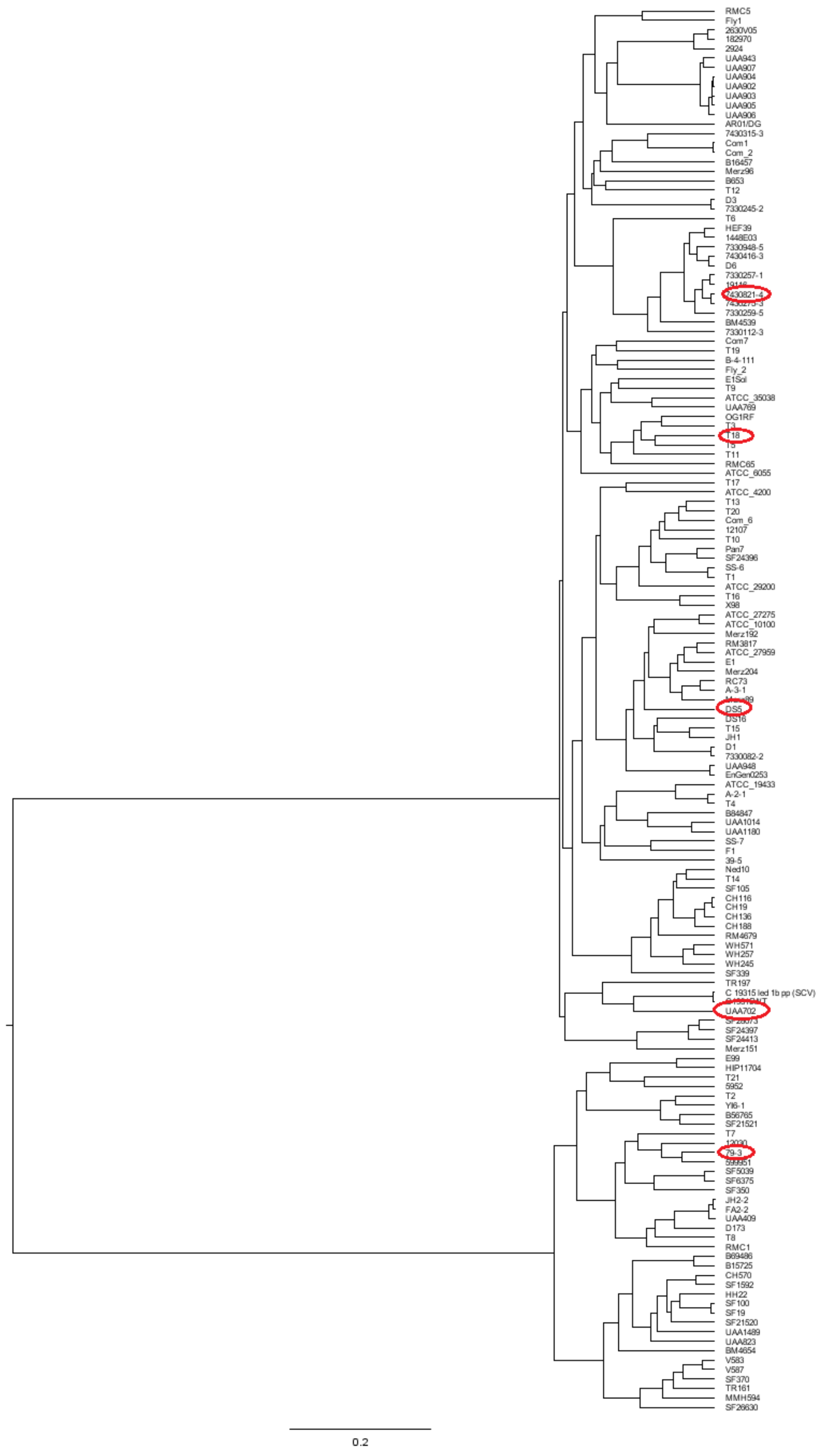
Table 10 Strains for which the genomes are used in this study, the amount of contigs for every genome are depicted between brackets

Appendix M Taxonomic trees

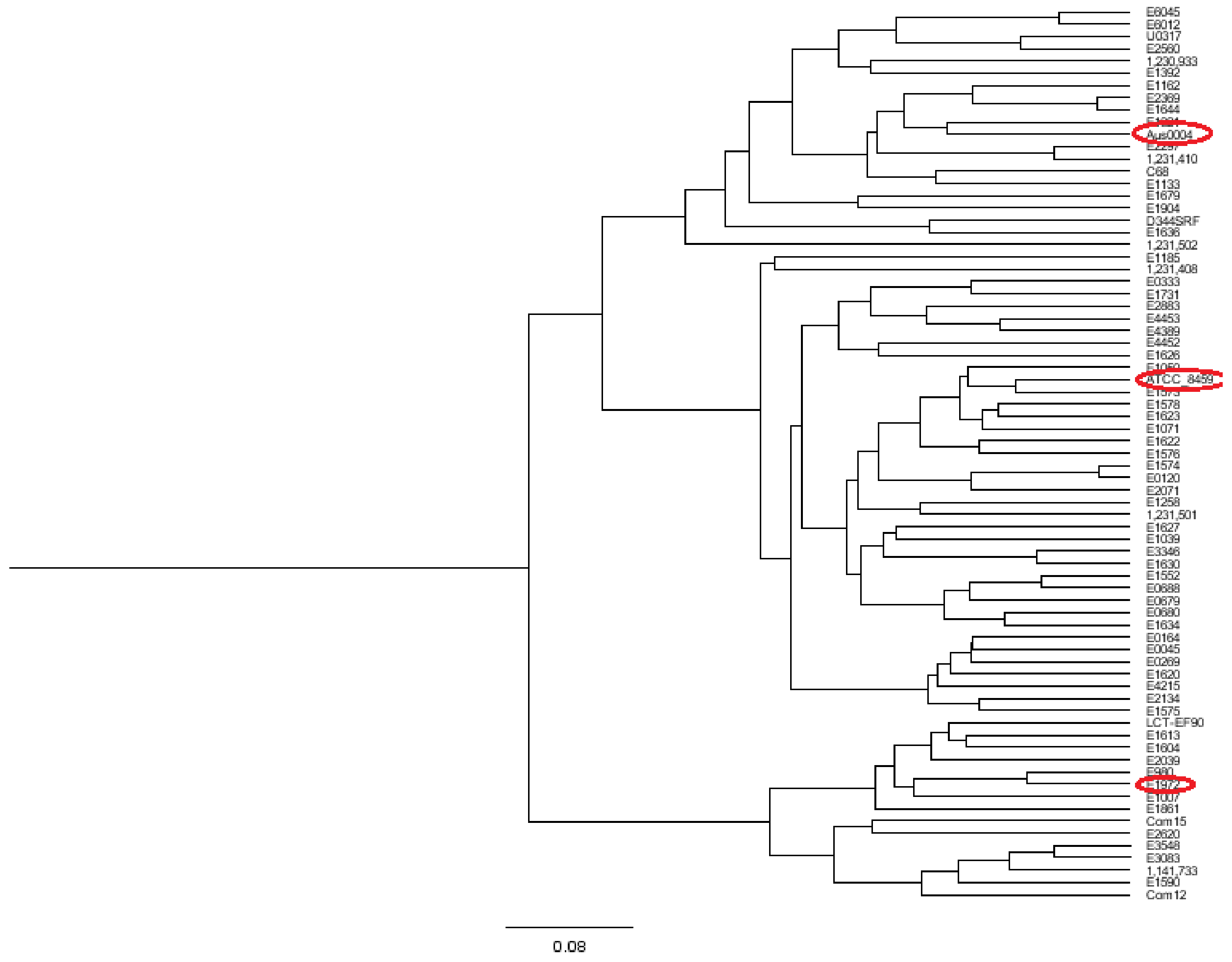
The taxonomic trees used for strain selection for use in synthetic metagenomic samples are shown below. The strains used for the experiments are marked with a red circle. These trees were extracted from the NCBI database and trimmed to so it solely contains strains present in our database.

Enterococcus

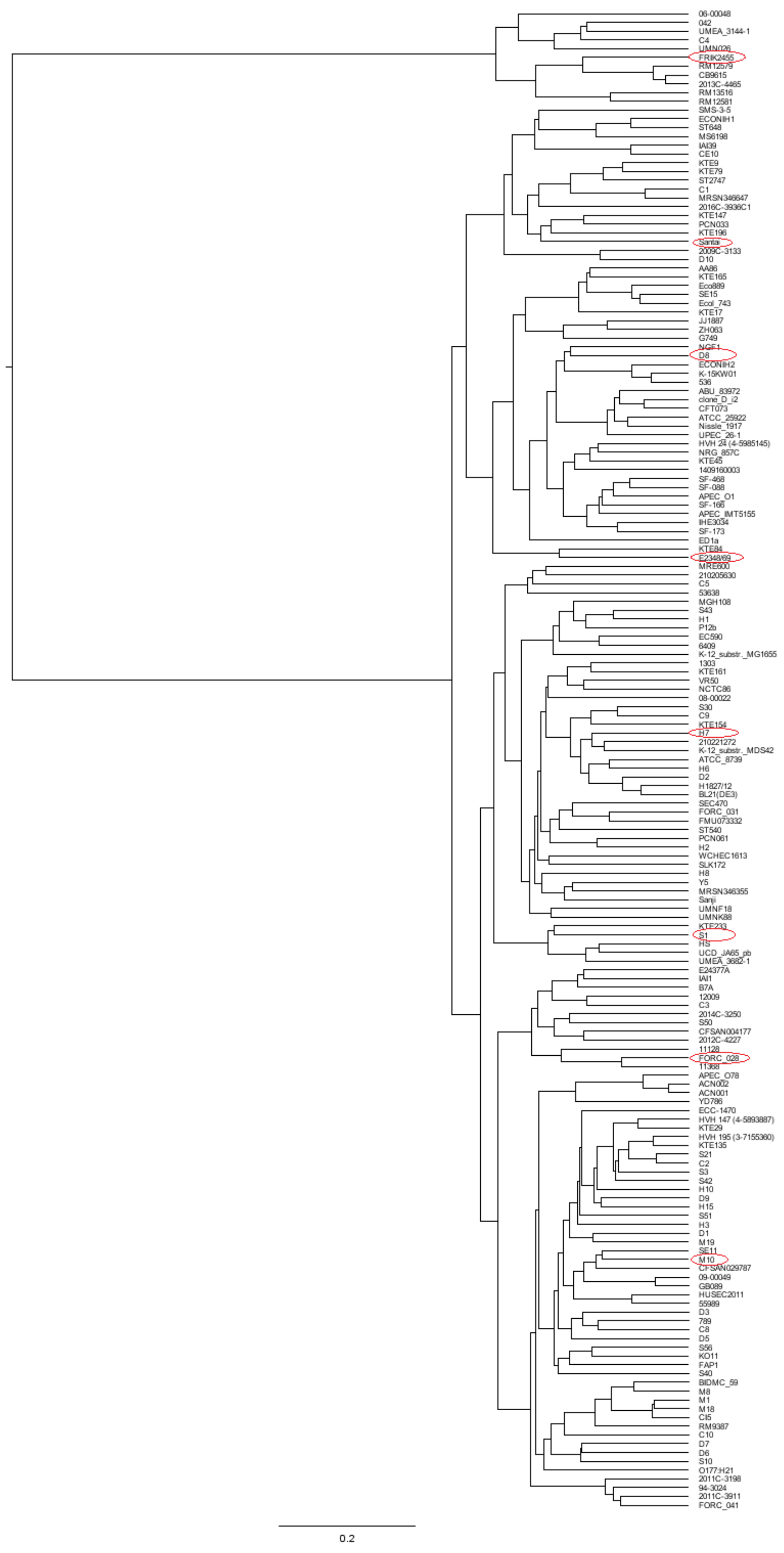
Enterococcus faecalis



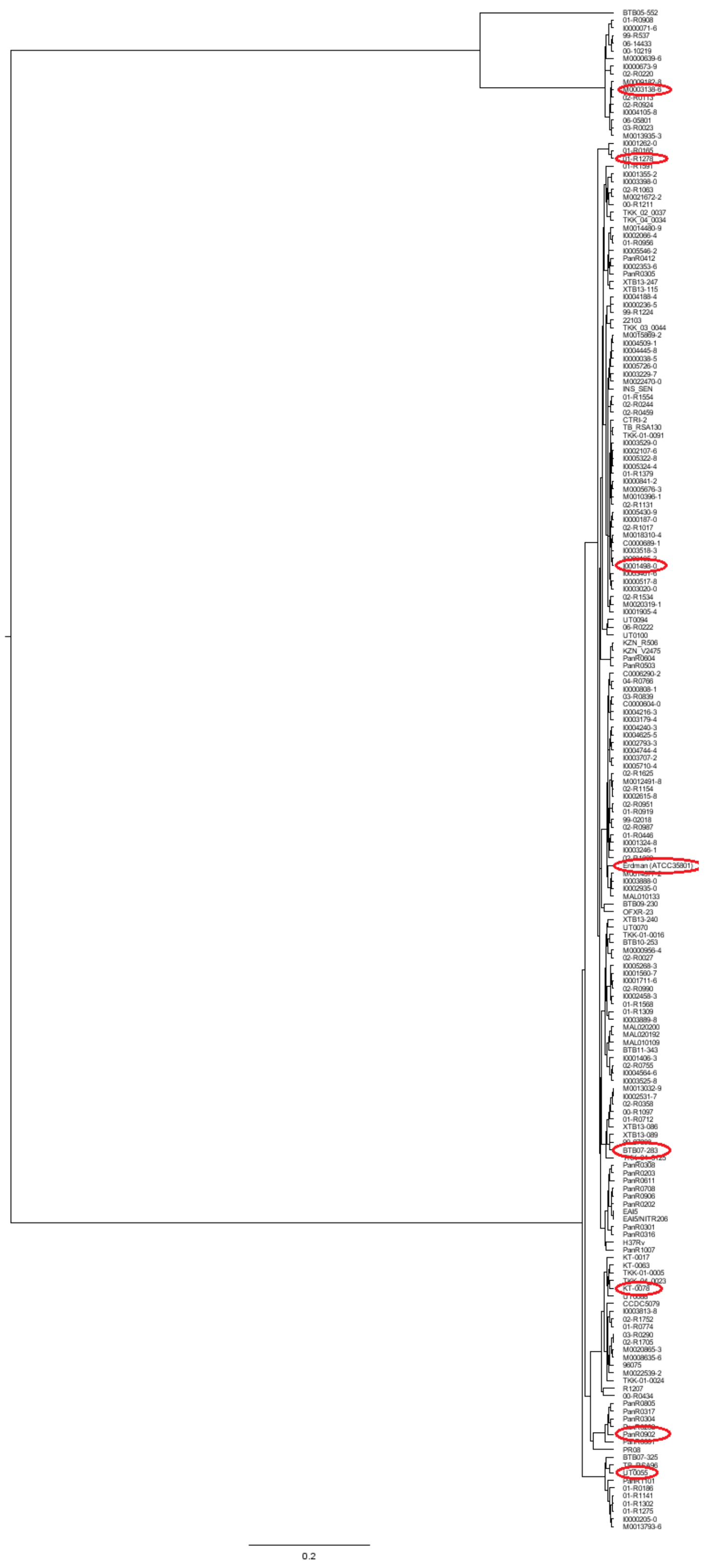
Enterococcus faecium



Escherichia coli



Mycobacterium tuberculosis


BTB05-552
01-R0908
I0000071-6
99-R537
06-14433
00-10219
M0000639-6
I0000673-9
02-R0220
M0009182-8
M0003138-6
02-R0113
02-R0924
I0004105-8
06-05801
03-R0023
M0013935-3
I0001262-0
01-R0165
01-R1278
01-R1591
I0001355-2
I0003398-0
02-R1063
M0021672-2
00-R1211
TKK_02_0037
TKK_04_0034
M0014480-9
I0002066-4
01-R0956
I0005546-2
PanR0412
I0002353-6
PanR0305
XTB13-247
XTB13-115
I0004188-4
I0000236-5
99-R1224
22103
TKK_03_0044
M0015869-2
I0004509-1
I0004445-8
I0000038-5
I0005726-0
I0003229-7
M0022470-0
INS_SEN
01-R1554
02-R0244
02-R0459
CTRI-2
TB_RSA130
TKK-01-0091
I0003529-0
I0002107-6
I0005322-8
I0005324-4
01-R1379
I0000841-2
M0005676-3
M0010396-1
02-R1131
I0005430-9
I0000187-0
02-R1017
M0018310-4
C0000689-1
I0003518-3
I0001498-0
I0000517-8
I0003020-0
02-R1534
M0020319-1
I0001905-4
UT0094
06-R0222
UT0100
KZN_R506
KZN_V2475
PanR0604
PanR0503
C0006290-2
04-R0766
I0000808-1
03-R0839
C0000604-0
I0004216-3
I0003179-4
I0004240-3
I0004625-5
I0002793-3
I0004744-4
I0003707-2
I0005710-4
02-R1625
M0012491-8
02-R1154
I0002615-8
02-R0951
01-R0919
99-02018
02-R0987
01-R0446
I0001324-8
I0003246-1
Erdman (ATCC35801)
I0003888-0
I0002935-0
MAL010133
BTB09-230
OFXR-23
XTB13-240
UT0070
TKK-01-0016
BTB10-253
M0000956-4
02-R0027
I0005268-3
I0001560-7
I0001711-6
02-R0990
I0002458-3
01-R1568
01-R1309
I0003889-8
MAL020200
MAL020192
MAL010109
BTB11-343
I0001406-3
02-R0755
I0004564-6
I0003525-8
M0013032-9
I0002531-7
02-R0358
00-R1097
01-R0712
XTB13-086
XTB13-089
BTB07-283
PanR0308
PanR0203
PanR0611
PanR0708
PanR0906
PanR0202
EAI5
EAI5/NITR206
PanR0301
PanR0316
H37Rv
PanR1007
KT-0017
KT-0063
TKK-01-0005
TKK-04-0023
KT-0078
CCDC5079
I0003813-8
02-R1752
01-R0774
03-R0290
02-R1705
M0020865-3
M0008635-6
96075
M0022539-2
TKK-01-0024
R1207
00-R0434
PanR0805
PanR0317
PanR0304
PanR0902
PR08
BTB07-325
TB_RSA96
UT0055
PanR1101
01-R0186
01-R1141
01-R1302
01-R1275
I0000205-0
M0013793-6
0.2