UNIVERSITY OF CALIFORNIA SAN DIEGO

Visualization and assembly methods for microbiome sequencing data

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Marcus William Fedarko

Committee in charge:

Professor Pavel A. Pevzner, Chair
Professor Nuno Bandeira
Professor Melissa Gymrek
Professor Siavash Mirarab

2025

The Dissertation of Marcus William Fedarko is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2025

DEDICATION

To my family and friends, who inspire me to be better; and also to the other weird people who download other people's theses to read the front matter and then don't read anything else in the document. But mostly the first thing though

EPIGRAPH

You were sick, but now you're well again, and there's work to do.

— Kurt Vonnegut

"It wasn't the second helping [of crab], it was the third that hurt,"
said Joe Lee, CEO of parent company Darden Restaurants, on a call
with investors. "And the fourth," said Red Lobster president Dick Rivera.

— Scott Meslow

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

There are a lot of people I need to thank, so let's get this show on the road.

First: thank you to my advisor, Pavel Pevzner. I am aware that taking on a PhD student out of the blue is a massive commitment, and I will always be thankful to Prof. Pevzner for his thorough mentorship, support, and patience. I hope that the past few years' work comes at least somewhere close to repaying his kindness.

Thank you to the other members of my dissertation committee: Melissa Gymrek, Nuno Bandeira, and Siavash Mirarab. (Starting in this paragraph, I'm going to be listing people in alphabetical order by first name.) I am sincerely grateful for their feedback on my work, and for tolerating the attendant deluge of scheduling emails.

Thank you to the past and present members of the Pevzner Lab I've had the chance to work with (Andrey Bzikadze, Anton Bankevich, Ishaan Gupta, Mikhail Kolmogorov, Rohan Vanheusden, Vikram Sirupurapu, and Zhenmiao Zhang), as well as the members of the Bafna and Bandeira Labs, for many helpful discussions and for making our Tuesday lab meetings enjoyable.

In the first two years of grad school (2018–2019 and 2019–2020) I worked in the Knight Lab; from 2019–2020, I also worked with the Center for Microbiome Innovation and IBM AI Horizons Network. I am thankful to the many colleagues I met during these years whom I had the chance to learn from, befriend, and harangue with naïve questions, including Antonio González Peña, Anupriya Tripathi, Austin Swafford, Bryn Taylor, Cameron Martino, Celeste Allaband, Charles Cowart, Dan Hakim, Daniel McDonald, Erfan Sayyari, Farhana Ali, Franck Lejzerowicz, Gail Ackermann, George Armstrong, Gibraan Rahman, Greg Humphrey, Greg Sepich-Poore, Ho-Cheol Kim, Jake Minich, Jamie Morton, Jeff DeReus, Jerry Kennedy, Jon Sanders, Julia Gauglitz, Justin Shaffer, Kalen Cantrell, Kristen Beck, Larry Smarr, Lisa Marotz, Mehrbod Estaki, Michiko Souza, Niina Haiminen, Pedro Belda-Ferre, Qiyun Zhu, Rob Knight, Robert Mills, Rodolfo Salido, Sarah Adams, Shi Huang, Tomasz Kościółek, Victor Cantu, Yimeng Yang, Yna Villanueva, and

Yoshiki Vázquez-Baeza. These years feel like a dream now, but I will always remember the surreal comfort of spending 2020 sequestered in a tiny studio apartment in grad housing working remotely on EMPress with Kalen and Yoshiki.

I decided to come to grad school largely because of the positive experiences I had as an undergraduate researcher in Mihai Pop's lab at the University of Maryland from 2016–2018. I am grateful to Prof. Pop and the wonderful members of his lab I was able to meet (Brian Brubach, Brook Stacy, Dan Nasko, Harihara Muralidharan, Jacquelyn Michaelis, Jay Ghurye, Jeremy Selengut, Kiran Javkar, Matt Myers, Nathan Olson, Nidhi Shah, Seth Commichaux, and Todd Treangen) for their support and kindness. I'm a bit jealous that UMD set up a fancy new computer science building just as I left for San Diego, but the Biomolecular Sciences Building (and the network of forested paths across from it, and the inimitable vibes of College Park, and the frankly unnecessary amount of pizza places College Park had) will always have a special place in my heart.

Thank you to Stephen Checkoway, the author of the LaTeX template I'm using to typeset this dissertation (https://github.com/stevecheckoway/ucsddissertation). Thank you also to the various former students who have updated this template over the years to maintain its compliance with UCSD's requirements.

Thank you to Andrey, Ishaan, Jennifer (and Finnigan), Tara, Vikram, and Zhenmiao for making CSE 4250 such a fun office to work in, and for many hours of unproductive gossiping; thank you to everybody I went running and/or climbing with; thank you to all my family and friends (from California, from Maryland, and from everywhere else) for your love and support. Sorry for taking too long to respond to your messages and/or for inundating you with low-quality internet memes.

Chapter 1.1, in full, is a reprint of the material as it appears in "Visualizing 'omic feature rankings and log-ratios using Qurro." **Fedarko MW**, Martino C, Morton JT, González A, Rahman G, Marotz CA, Minich JJ, Allen EA, and Knight R. *NAR Genomics and Bioinformatics 2*(2), 2020. The dissertation author was the primary investigator and

first author of this paper.

Chapter 1.2, in full, is a reprint of the material as it appears in "EMPress Enables Tree-Guided, Interactive, and Exploratory Analyses of Multi-omic Data Sets." Cantrell K, **Fedarko MW**, Rahman G, McDonald D, Yang Y, Zaw T, Gonzalez A, Janssen S, Estaki M, Haiminen N, Beck KL, Zhu Q, Sayyari E, Morton JT, Armstrong G, Tripathi A, Gauglitz JM, Marotz C, Matteson NL, Martino C, Sanders JG, Carrieri AP, Song SJ, Swafford AD, Dorrestein PC, Andersen KG, Parida L, Kim H-C, Vázquez-Baeza Y, and Knight R. *mSystems 6*(2), 2021. The dissertation author was a primary investigator and co-first author of this paper.

Chapter 2, in full, is a reprint of the material as it appears in "Metagenome assembly of long and accurate reads using iterative downsampling." **Fedarko MW**, Zhang Z, Bankevich A, and Pevzner PA. *In preparation.* The dissertation author was the primary investigator and first author of this paper.

Chapter 3, in full, is a reprint of the material as it appears in "Analyzing rare mutations in metagenomes assembled using long and accurate reads." **Fedarko MW**, Kolmogorov M, and Pevzner PA. *Genome Research 32*(11-12), 2022. The dissertation author was the primary investigator and first author of this paper.

Chapter 4, in full, is a reprint of the material as it appears in "Efficient creation and visualization of exact dot plot matrices." **Fedarko MW**. *In preparation.* The dissertation author was the primary investigator and sole author of this paper.

VITA

2018	B.S. with High Honors in Computer Science, University of Maryland
2022	M.S. in Computer Science, University of California San Diego
2023	C. Phil. in Computer Science, University of California San Diego
2025	Ph.D. in Computer Science, University of California San Diego

PUBLICATIONS

Fedarko MW, Kolmogorov M, and Pevzner PA (2022). "Analyzing rare mutations in metagenomes assembled using long and accurate reads." *Genome Research, 32*(11-12):2119–2133.

Cantrell K*, **Fedarko MW***, Rahman G, McDonald D, Yang Y, Zaw T, Gonzalez A, Janssen S, Estaki M, Haiminen N, Beck KL, Zhu Q, Sayyari E, Morton JT, Armstrong G, Tripathi A, Gauglitz JM, Marotz C, Matteson NL, Martino C, Sanders JG, Carrieri AP, Song SJ, Swafford AD, Dorrestein PC, Andersen KG, Parida L, Kim H-C, Vázquez-Baeza Y, and Knight R (2021). "EMPress Enables Tree-Guided, Interactive, and Exploratory Analyses of Multi-omic Data Sets." *mSystems, 6*(2):e01216-20. (* = contributed equally)

Huey SL, Jiang L, **Fedarko MW**, McDonald D, Martino C, Ali F, Russell DG, Udipi SA, Thorat A, Thakker V, Ghugre P, Potdar RD, Chopra H, Rajagopalan K, Haas JD, Finkelstein JL, Knight R, and Mehta S (2020). "Nutrition and the Gut Microbiota in 10- to 18-Month-Old Children Living in Urban Slums of Mumbai, India." *mSphere, 5*(5):e00731-20.

Fedarko MW, Martino C, Morton JT, González A, Rahman G, Marotz CA, Minich JJ, Allen EA, and Knight R (2020). "Visualizing 'omic feature rankings and log-ratios using Qurro." *NAR Genomics and Bioinformatics, 2*(2):lqaa023.

Sanders JG, Nurk S, Salido RA, Minich J, Xu ZZ, Martino C, **Fedarko M**, Arthur TD, Chen F, Boland BS, Humphrey GC, Brennan C, Sanders K, Gaffney J, Jepsen K, Khosroheidari M, Green C, Liyange M, Dang JW, Phelan VV, Quinn RA, Bankevich A, Chang JT, Rana TM, Conrad DJ, Sandborn WJ, Smarr L, Dorrestein PC, Pevzner PA, and Knight R (2019). "Optimizing sequencing protocols for leaderboard metagenomics by combining long and short reads." *Genome Biology, 20*(1):226.

Ghurye J, Treangen T, **Fedarko M**, Hervey WJ, and Pop M (2019). "MetaCarvel: linking assembly graph motifs to biological variants." *Genome Biology, 20*(1):174.

Meisel JS, Nasko DJ, Brubach B, Cepeda-Espinoza V, Chopyk J, Corrada-Bravo H, **Fedarko M**, Ghurye J, Javkar K, Olson ND, Shah N, Allard SM, Bazinet AL, Bergman NH, Brown A, Caporaso JG, Conlan S, DiRuggiero J, Forry SP, Hasan NA, Kralj J, Luethy PM, Milton DK, Ondov BD, Preheim S, Ratnayake S, Rogers SM, Rosovitz MJ, Sakowski EG, Schliebs NO, Sommer DD, Ternus KL, Uritskiy G, Zhang SX, Pop M, and Treangen TJ (2018). "Current progress and future opportunities in applications of bioinformatics for biodefense and pathogen detection: Report from the Winter Mid-Atlantic Microbiome Meet-up, College Park, MD January 10^{th}, 2018." *Microbiome, 6*(1):197.

ABSTRACT OF THE DISSERTATION

Visualization and assembly methods for microbiome sequencing data

by

Marcus William Fedarko

Doctor of Philosophy in Computer Science

University of California San Diego, 2025

Professor Pavel A. Pevzner, Chair

Microorganisms make homes of our bodies, our food and water, and the world among us. Recent decades have seen the development of marker gene and metagenome sequencing—methods that attempt to read the DNA of the microorganisms within a sample—as tools to study the precise "microbial ecology" of these environments.

The widespread adoption of these methods and accompanying deluge of sequencing data have led to a need for reliable, user-friendly software that can automate and simplify common analyses of these data. To this end Chapter 1.1 presents Qurro, a tool for visualizing feature rankings and log-ratios in the context of differential abundance analyses; and Chapter 1.2 presents EMPress, a tool for visualizing phylogenetic trees and associated

information. These methods lower the barriers to performing exploratory analyses of microbiome sequencing data, and are applicable to data produced by other high-throughput assays such as untargeted mass spectrometry.

The rise of microbiome sequencing has been soured by the reproducibility crisis. For example, the literature is awash with dramatic, irreproducible claims that the human gut microbiome is connected to sundry health conditions. These troubles are attributable in part to the limited resolution available from current sequencing methods: although small genotypic differences between two strains can be indicative of important phenotypic differences, current studies often fail to profile genomes to even the species level.

Determining the full genomes of the microorganisms in a sample typically requires at minimum assembling the reads produced by metagenome sequencing. Chapter 2 describes metaLJA, a metagenome assembler that uses iterative downsampling and de Bruijn graph simplification steps to attempt to reconstruct genomes from long and accurate reads.

The remainder of this dissertation describes methods for the high-resolution analysis of metagenome assemblies like those produced by metaLJA. Chapter 3 presents the strainFlye pipeline for the identification and analysis of rare mutations in metagenome assemblies of long and accurate reads, and shows that a wealth of diversity can remain hidden within even a single metagenome-assembled genome. Finally, Chapter 4 presents wotplot, a library for the efficient creation and visualization of exact dot plot matrices.

Chapter 1

Visualization methods for general 'omic data

The outputs of many high-throughput "'omic" assays—including marker gene and metagenome sequencing, RNA-seq, and untargeted mass spectrometry—are often transformed into a *feature table*: a matrix where one axis corresponds to samples, the other axis corresponds to features, and each entry indicates the observed abundance of a given feature in a given sample [120]. Though the meanings of "feature" and "abundance" are dependent upon the assay and the ways in which its raw data were transformed into a feature table, these tables usually share certain statistical properties and can thus be analyzed in similar ways [49].

There is now widespread agreement, for example, that feature tables produced from current marker gene and metagenome sequencing projects are *compositional*: that is, the sum of all feature abundances for a given sample is in general an arbitrary uninformative quantity imposed by the sequencing machine [49, 60]. Ignoring this property and analyzing these data as if they were absolute abundances can lead to incorrect conclusions [60, 132].

Given a feature table and further information about this table's samples (in a microbiome sequencing study, for example, some samples might be labelled as being obtained from "healthy" or "sick" patients), we are often interested in performing a *differential abundance* (also known as differential expression) analysis: that is, asking

the question of which features are associated with which types of samples [60, 109]. To aid researchers in answering this question while respecting the compositional nature of microbiome sequencing data Chapter 1.1 presents Qurro, a visualization tool created to simplify differential abundance analyses by relating feature rankings to feature log-ratios (as proposed in [132] and [118]).

The features in a feature table can often be related to each other in a tree—for example, a phylogenetic tree describing the estimated evolutionary history of a group of sequences [110, 128, 84], or a hierarchy of molecules [194]. To help researchers analyze these structures Chapter 1.2 presents EMPress, a tree visualization tool that facilitates the visual integration of these structures with additional sample and feature information. EMPress contains novel functionality, such as integration with ordination plots using the EMPeror tool [200], and uses some optimizations in order to scale to the visualization of relatively large trees while running in a web browser. EMPress' barplot functionality also makes it easily applicable to the visualization of differential abundance information alongside a tree, as shown for example in Figure 1.4C and in [117].

Qurro and EMPress are both available as plugins through the QIIME 2 framework [18] or as standalone Python packages. By expediting certain analyses of 'omic data, these tools have helped advance the pace of bioinformatics research.

1.1 Visualizing 'omic feature rankings and log-ratios using Qurro

1.1.1 Abstract

Many tools for dealing with compositional "'omics" data produce feature-wise values that can be ranked in order to describe features' associations with some sort of variation. These values include differentials (which describe features' associations with specified covariates) and feature loadings (which describe features' associations with variation along

a given axis in a biplot). Although prior work has discussed the use of these "rankings" as a starting point for exploring the log-ratios of particularly high- or low-ranked features, such exploratory analyses have previously been done using custom code to visualize feature rankings and the log-ratios of interest. This approach is laborious, prone to errors and raises questions about reproducibility. To address these problems we introduce Qurro, a tool that interactively visualizes a plot of feature rankings (a "rank plot") alongside a plot of selected features' log-ratios within samples (a "sample plot"). Qurro's interface includes various controls that allow users to select features from along the rank plot to compute a log-ratio; this action updates both the rank plot (through highlighting selected features) and the sample plot (through displaying the current log-ratios of samples). Here, we demonstrate how this unique interface helps users explore feature rankings and log-ratios simply and effectively.

1.1.2 Introduction

High-throughput sequencing and metabolomics data detailing the organisms, genes or molecules identified within a microbial sample are inherently compositional [60, 132]: that is, absolute abundances are often inaccessible and only relative information can be obtained from the data. These data must be interpreted accordingly. Performing a differential abundance analysis in a dataset generally requires selecting a "reference frame" (denominator) for log-ratio analysis, then relating the resulting log-ratios to sample metadata [60, 132]. Critically, how to best select such a "reference frame" is an open question. The implicit use of different references across different studies can be a cause of irreproducible findings [132].

Various tools for differential abundance analyses including but not limited to ALDEx2 [49] and Songbird [132] can produce *differentials*, which describe the (estimated) log-fold change in relative abundance for features in a dataset with respect to certain covariate(s) [132]. Similarly, tools like DEICODE [118] can produce *feature loadings* that

characterize features' impacts in a compositional biplot [2]. Differentials and feature loadings alike can be sorted numerically and used as *feature rankings*, and this representation provides relative information about features' associations with some sort of variation in a dataset [132, 118]. The natural next step is to use these rankings as a guide for log-ratio analyses (e.g. by examining the log-ratios of high- to low-ranked features). However, modern studies commonly describe hundreds or thousands of observed features: manually exploring feature rankings, whether as a tabular representation or as visualized using one-off scripts, is inconvenient.

Here we present Qurro (pronounced "churro"), a visualization tool that supports the analysis of feature log-ratios in the context of feature rankings and sample metadata. Qurro uses a two-plot interface: a "rank plot" shows how features are differentially ranked for a selected differential or feature loading (as shown in Figures 1.1A and 1.2A), and a "sample plot" shows log-ratios of the selected features across samples relative to selected sample metadata field(s) (as shown in Figures 1.1B and C, 1.2B and C). These plots are linked [11]: selecting features for a log-ratio highlights these features in the rank plot and updates the y-axis values of samples (corresponding to the value of the currently selected log-ratio for each sample) in the sample plot. This interface is intended to make it easy for researchers to explore log-ratios in a dataset, using feature rankings as a starting point.

Due to its unique display, and the availability of multiple controls for feature selection and plot customization, Qurro simplifies compositional data analyses of 'omic data.

1.1.3 Implementation

Qurro's source code is released under the BSD 3-clause license and is available at https://github.com/biocore/qurro.

Qurro's codebase includes a Python 3 program that generates a visualization and the HTML/JavaScript/CSS code that manages this visualization. Qurro can be used as a

standalone program or as a QIIME 2 plugin [18].

Both plots in a Qurro visualization are embedded as Vega-Lite JSON specifications [167], which are generated by Altair [197] in Qurro's Python code. An advantage of Qurro's use of the Vega infrastructure is that both plots in a Qurro visualization can be customized to the user's liking in the Vega-Lite or Vega grammars. As an example of this customizability, the Vega-Lite specifications defining Figures 1.1A–C and 1.2A–C of this paper were edited programmatically in order to increase font sizes, change the number of ticks shown, etc. (Our Python script that makes these modifications is available online; please see section 1.1.6, "Data availability".)

Code dependencies

In addition to Altair, Qurro's Python code directly relies on the BIOM format [120], Click (https://palletsprojects.com/p/click), NumPy [196], pandas [211] and scikit-bio (http://scikit-bio.org) libraries. Qurro's web code relies on Vega [168], Vega-Lite [167], Vega-Embed (https://github.com/vega/vega-embed), RequireJS (https://requirejs.org), jQuery (https://jquery.com), DataTables (https://datatables.net), Bootstrap (https://getbootstrap.com), Bootstrap Icons (https://icons.getbootstrap.com), and Popper.js (https://popper.js.org).

1.1.4 Case study: the gills of *Scomber japonicus*

To demonstrate the utility of Qurro on a dataset with clear "signals," we applied it to an extant dataset of V4-region 16S rRNA sequencing data from Pacific chub mackerel (*Scomber japonicus*) and environmental samples [126]. This dataset, currently described in a preprint, includes samples taken from five *S. japonicus* body sites (digesta, GI, gill, pyloric caeca and skin) from 229 fish captured across 38 time points in 2017, along with many seawater, marine sediment, positive/negative control and non-*S. japonicus* fish samples. A Jupyter Notebook [95] showing how we processed this dataset computationally

is available online; see section 1.1.6, "Data availability".

Sample processing and analysis

When these samples were initially sequenced, the KatharoSeq protocol [127] was followed. This led us to exclude samples with less than 1370 total counts from our analysis of this dataset.

Sequencing data (already processed using QIIME 1.9.1 [22] and Deblur [3] on Qiita [61]) were further processed and analyzed using QIIME 2 [18]. Our use of Deblur outputs as the starting point in our analysis means that "features" in our analysis of this dataset correspond to "sub-operational-taxonomic-units" (sOTUs), although Qurro is interoperable with compositional datasets including arbitrary types of "features."

These sOTUs were assigned taxonomic classifications using q2-feature-classifier [17]. Specifically, we extracted sequences from the SILVA 132 99% database [154] using the same forward [144] and reverse [6] primer sequences as were used for sample processing, trained a Naïve Bayes classifier on these extracted sequences, and then used this classifier (through q2-feature-classifier's `classify-sklearn` method [147]) to classify sOTUs in our dataset based on their sequences.

Due to upstream filtering steps taken in our analysis (a combination of filtering out non-*S. japonicus* and non-seawater samples, applying the aforementioned KatharoSeq sample exclusion criterion, Songbird's default `--min-feature-count` of each feature needing to be present in at least 10 samples, and Qurro's behavior of filtering out empty samples and features), 639 samples and 985 features were included in the Qurro visualization produced for this case study.

Computing "body site" differentials

One basic question about this dataset we investigated using Qurro was of which features were associated with which *S. japonicus* body sites. To produce feature rankings

accordingly, we used Songbird [132] to compute differentials detailing features' associations with samples from each of the five studied body sites, using the seawater samples in the dataset as a reference category for Songbird's internal construction of a design matrix representing the sample categories being analyzed (Appendix A.1.1, "Computing feature differentials using Songbird").

In general, highly ranked features for a differential—the (estimated) log-fold change in relative abundance for a feature with respect to some covariate(s)—are positively associated with samples from these covariate(s), while lowly ranked features are negatively associated with these covariate(s). These differentials can therefore be thought of as a starting point for investigating differentially abundant features for particular fish body sites in this dataset.

Using Qurro to analyze differentials and log-ratios

Qurro simplifies the process of analyzing features' log-ratios in the context of these differentials. The "rank plot" of a Qurro visualization is a bar plot where each bar corresponds to a single differentially ranked feature. The y-axis values of each feature's bar are either the estimated logfold change values for that feature if the feature rankings are differentials (as is the case in our analysis here, and therefore in Figures 1.1A and 1.2A), or the loadings of each feature along a selected biplot axis if the feature rankings are feature loadings in a biplot. In either case, features are sorted in ascending order by these values along the rank plot's x-axis. The exact differential or feature loading used is configurable, so Qurro users can quickly toggle between these; for the case study Qurro visualization, this means that users can—for example—quickly switch between differentials computed based on association with skin samples to differentials computed based on association with gill samples.

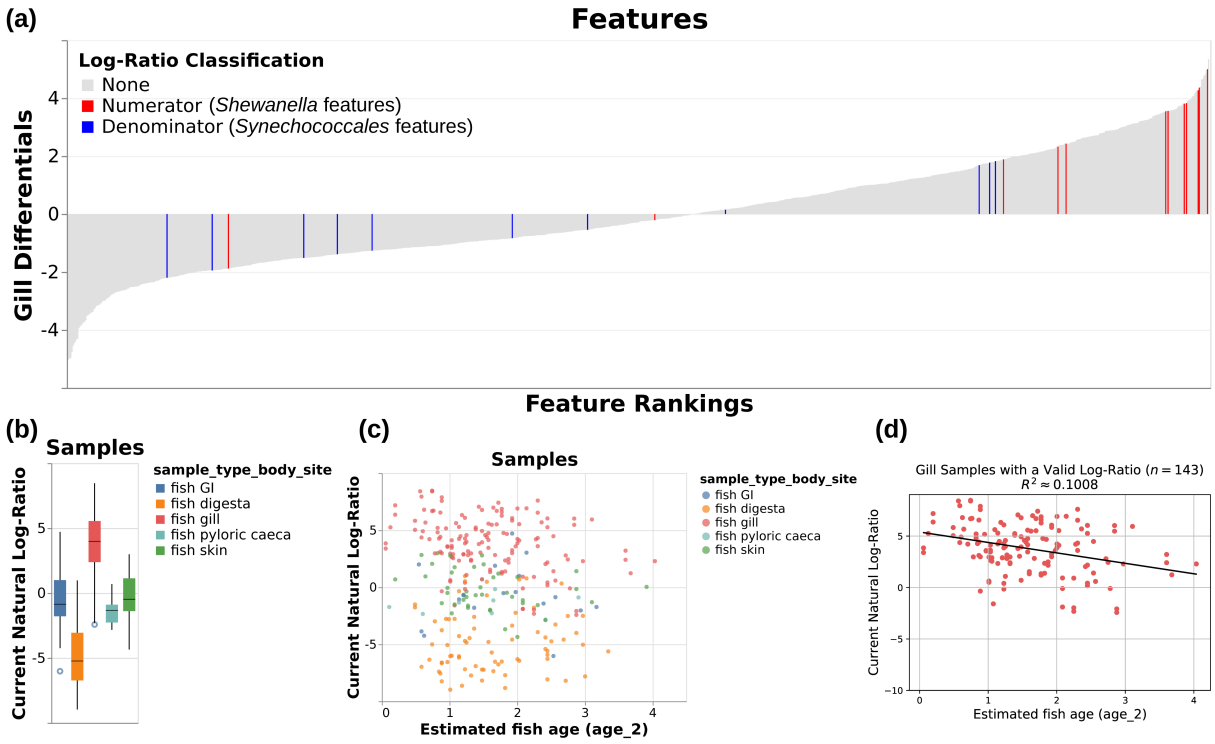


Figure 1.1. Various outputs from the case study showing the log-ratio of classified *Shewanella* features to classified *Synechococcales* features. **(a)** "Rank plot" showing differentials computed based on association with gill samples, using seawater samples as a reference category in the regression. The term "Log-Ratio Classification" only refers to the currently selected log-ratio in the Qurro visualization: in this case, this is the log-ratio of classified *Shewanella* features to classified *Synechococcales* features. To show the rankings of these "selected" features relative to the remaining features in the dataset, these features are colored in the rank plot: *Shewanella* features are colored in red, and *Synechococcales* features are colored in blue. The remaining features, colored gray, have a "Log-Ratio Classification" of `None` because they are not involved in the selected *Shewanella*-to-*Synechococcales* log-ratio. **(b)** "Sample plot" in boxplot mode, showing samples' *Shewanella*-to-*Synechococcales* log-ratios by sample body site. Note that only 285 samples are represented in this plot; other samples were either filtered out upstream in the analysis or contained zeroes on at least one side of their log-ratio. **(c)** "Sample plot," showing a scatterplot of samples' selected log-ratios versus estimated fish age. Individual samples are colored by body site. As in panel B, only 285 samples are present. **(d)** Ordinary-least-squares linear regression ($R^2 \approx 0.1008$) between estimated fish age and the selected log-ratio for just the 143 gill samples shown in B and C, computed outside of Qurro using scikit-learn [147] and pandas [211] and plotted using matplotlib [78].

Highlighting features on the rank plot.

The initial study of this dataset [126] agreed with prior work [152] on the frequency of *Shewanella* spp. in the fish gill microbiome. Qurro supports searching for features using arbitrary feature metadata (e.g. taxonomic annotations), and using this functionality to highlight *Shewanella* spp. on the rank plot of gill differentials (Appendix A.1.2, "Qurro log-ratio-selection controls used") corroborates these findings: as Figure 1.1A shows, the majority of identified *Shewanella* spp. are highly ranked for the gill differentials relative to the other features in this dataset.

Particularly high- or low-ranked features like *Shewanella* spp. can merit further examination via a log-ratio analysis [132]; in particular, one question we might be interested in asking at this point is if *Shewanella* spp. are similarly abundant across other fish body sites. The remainder of this case study discusses a simple exploratory investigation in pursuit of an answer to this question, as well as to a few other questions that came up along the way.

Choosing a suitable "reference frame".

The compositional nature of marker gene sequencing data means that we cannot simply compare the abundances of *Shewanella* across samples in this dataset alone; however, we can instead compare the log-ratio of *Shewanella* and other features in this dataset across samples [132].

For demonstrative purposes, we chose the taxonomic order *Synechococcales* as the denominator ("reference frame") for the first log-ratio shown here. Features in this dataset belonging to this order included sOTUs classified in the genera *Cyanobium*, *Prochlorococcus* and *Synechococcus*. These are common genera of planktonic picocyanobacteria found ubiquitously in marine surface waters [169]. The expected stability of this group of features across samples in this dataset supports its use as a denominator here [132]. Furthermore, as shown in Qurro's rank plot in Figure 1.1A, many *Synechococcales* features are relatively

lowly ranked for the gill differentials; this gives additional reason to expect a comparative difference among gill samples for the *Shewanella*-to-*Synechococcales* log-ratio.

Qurro's computation of log-ratios.

Currently, Qurro computes log-ratios between between arbitrary groups of N selected numerator features and D selected denominator features by, for each sample S, computing the log-ratio of the sums of the raw abundances of the numerator and denominator features:

$$\text{LogRatio}\big(S, N, D\big) = \ln\left(\sum_{n \in N} S_n\right) - \ln\left(\sum_{d \in D} S_d\right)$$

Computing log-ratios by summing feature abundances in this way, as opposed to taking the geometric mean of these abundances (e.g. as described in [162]) has benefits and downsides alike, as discussed in a recent preprint [157]. One benefit is that this approach is relatively robust to highly sparse datasets like those commonly encountered in microbiome studies, since the presence of a zero-abundance feature in a group on one side of a sample's log-ratio does not necessarily force this sample to have an invalid log-ratio. There are likely better alternatives to amalgamating feature abundances in this way, but this approach is useful for exploratory analysis nonetheless (and it is modifiable in Qurro's source code, should another method of amalgamation be desired in the future).

Relating log-ratios to sample metadata.

Upon selecting a numerator and a denominator for a log-ratio (in this case, by searching through taxonomic annotations), Qurro updates the sample plot so that all samples' y-axis ("Current Natural Log-Ratio") values are equal to the value of the selected log-ratio for that sample. The x-axis field, color field, and scale types of these fields—along with other options—can be adjusted by the user interactively to examine the selected log-ratio from new perspectives.

Once the log-ratio of *Shewanella*-to-*Synechococcales* was selected, Figure 1.1B was produced by setting the sample plot x-axis to the categorical `sample_type_body_site` field and checking the "Use boxplots for categorical data?" checkbox. The resulting boxplot shows that the *Shewanella*-to-*Synechococcales* log-ratio is relatively high in gill samples, compared with other body sites' samples (Figure 1.1B). This observation corroborates the initial study of this dataset on the frequency of *Shewanella* particular to the fish gill microbiome [126].

Qurro can visualize quantitative sample metadata, as well. Using this functionality, we can add additional perspectives to our previously-reached observation. Age has been discussed as a factor impacting the microbiota of fish gills in this and other datasets [126, 152], and we use it here as an illustrative example of visualizing a quantitative metadata field alongside a log-ratio. By setting the x-axis field to the `age_2` metadata field (estimated fish age), changing the x-axis field scale type to "Quantitative," and setting the color field to `sample_type_body_site`, we get Figure 1.1C—a scatterplot showing the selected log-ratio viewed across samples by the estimated age of their host fish.

One trend that stood out to us in this scatterplot, and one of the reasons we chose age for this example, is that this plot contains an apparent negative correlation between the selected log-ratio and estimated fish age for gill samples. To support further investigation of patterns like this, Qurro can export the data backing the sample plot to a standard tab-separated file format—this file can then be loaded and analyzed in essentially any modern statistics software or programming language. This functionality was used to generate Figure 1.1D, in which we quantify and visualize this correlation for gill samples using ordinary-least-squares linear regression ($R^2 \approx 0.1008$). Although obviously not evidence of a causal relationship, this result opens the door for further investigation of this trend. One of many possible explanations for this observed trend is that the gills of younger fish are differentially colonized by *Shewanella* spp. and/or by *Synechococcales*; this may, in turn, be reflective of factors like vertical habitat use, immune development, or

food choice.

Interrogating the "multiverse" of reference frames.

Prior literature has shown the impact that choices in data processing can have on a study's results, and on the corresponding "multiverse" of datasets generated during this process [182]. We submit that the choice of reference frame (denominator) in log-ratio analyses introduces a similar "multiverse": for a set of n features, there are $O(2^n)$ possible subsets [132], so manually checking all possible reference frames for a given numerator is an intractable effort for the vast majority of datasets (although various heuristic methods have been proposed to address this sort of problem, e.g. [162]). In spite of this, the interactive nature of Qurro simplifies the task of validating results across reference frames.

Revisiting our analysis of *Shewanella* spp. in the gills of *S. japonicus*, there are multiple reasonable choices for reference frames. We chose *Synechococcales* mostly due to its expected ubiquity and stability across the marine samples in this dataset, but many other plausible choices exist.

In Figure 1.2, we repeat the exact same analysis as in Figure 1.1: but instead of using *Synechococcales* as the denominator of our log-ratio, we instead select the bottom $\sim 10\%$ (98/985) of features as ranked by gill differentials as the denominator (Figure 1.2A; Appendix A.1.2, "Qurro log-ratio-selection controls used"). Refreshingly, this log-ratio also shows clear "separation" of gill samples from other body sites' samples in the dataset (Figure 1.2B), as well as a similar negative correlation between estimated fish age and this log-ratio for gill samples (Figure 1.2C and D) ($R^2 \approx 0.1350$). This serves as further evidence for our previous claims: although we still can't say for sure, we can now more confidently state that *Shewanella* spp. seem to be dominant in the gills of *S. japonicus*, and that *Shewanella* abundance in these fishes' gills seems to be negatively correlated with (estimated) fish age—since the trends shown in Figures 1.1B–D and 1.2B–D have held up across multiple log-ratios with *Shewanella* as the numerator.

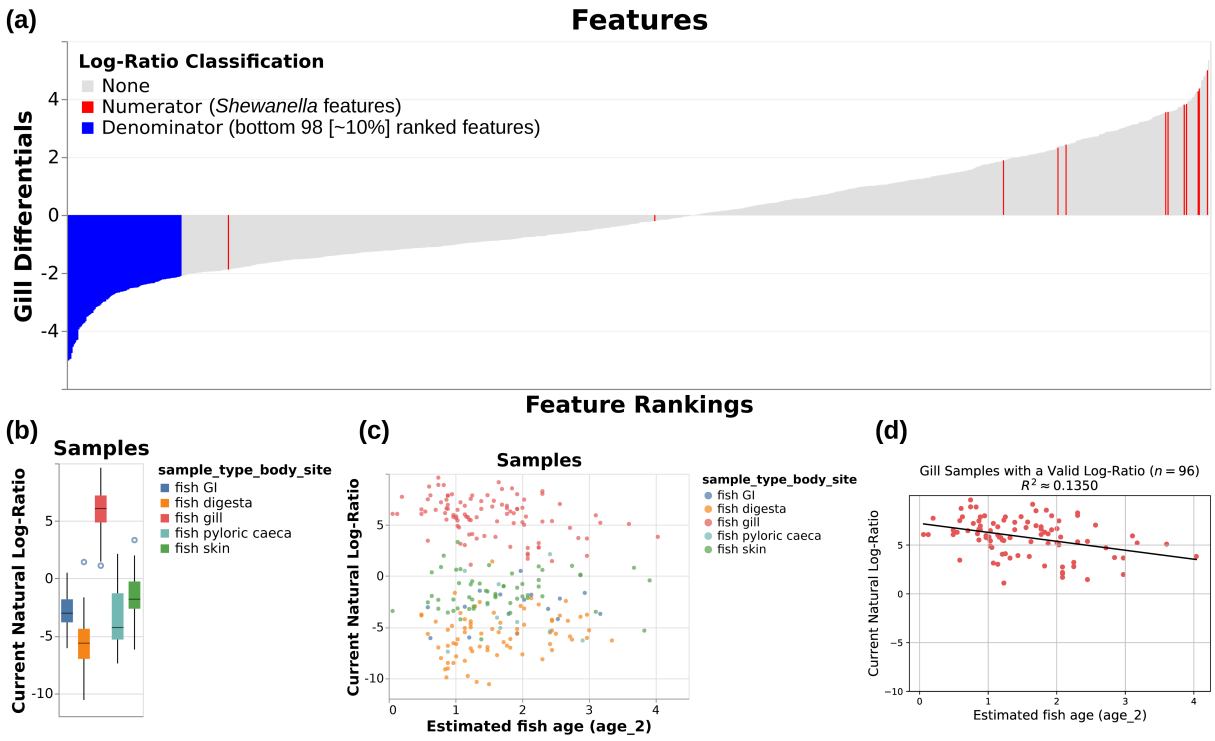


Figure 1.2. Various outputs from the case study (analogous to those in Figure 1.1) showing the log-ratio of classified *Shewanella* features to the bottom 98 ranked features for the gill differentials. **(a)** "Rank plot" analogous to that shown in Figure 1.1A, with the selected numerator features (those classified as *Shewanella* spp.) colored in red and the selected denominator features (the bottom 98 ranked features for the gill differentials) colored in blue. **(b)** "Sample plot" in boxplot mode, showing the selected log-ratios of samples by body site. 252 samples are represented in this plot; as in Figure 1.1B, other samples were either filtered out upstream in the analysis or contained zeroes on at least one side of their log-ratio. **(c)** "Sample plot," showing a scatterplot of samples' selected log-ratios versus estimated fish age. Individual samples are colored by body site. As in panel B, only 252 samples are present. **(d)** Ordinary-least-squares linear regression ($R^2 \approx 0.1350$) between estimated fish age and the selected log-ratio for just the 96 gill samples shown in B and C, computed outside of Qurro as specified for Figure 1.1D.

Handling "invalid" samples.

It is worth noting that many samples—including all of the seawater samples in the Qurro visualization (Appendix A.1.3, "Details on Qurro (and Songbird) input data filtering")—are not present in Figures 1.1B–D or 1.2B–D. If a given sample in Qurro cannot be displayed for some reason—for example, the sample has a zero in the numerator and/or denominator of the currently selected log-ratio—Qurro will drop that particular sample from the sample plot. Furthermore, to make sure the user understands the situation, Qurro will update a text display below the plot that includes the number and percentage of samples excluded for each "reason." This behavior helps users avoid spurious results caused by visualizing only a small proportion of a dataset's samples.

Using Qurro in practice

Since Qurro visualizations are essentially just web pages it is trivial to host them online, thus making them viewable by anyone using a compatible web browser. As an example of this we have made Qurro visualizations of various datasets, including the case study's, publicly available at https://biocore.github.io/qurro. We encourage users of Qurro to share their visualizations in this way, whenever possible, in order to encourage reproducibility and facilitate public validation of the conclusions drawn. Furthermore, we encourage readers of this paper to reconstruct Figures 1.1 and 1.2 and verify that this paper's claims are accurate.

1.1.5 Conclusion

Qurro serves as a natural "first step" for users of modern differential abundance tools to consult in order to analyze feature rankings, simplifying the work needed to go from hypothesis to testable result. We have already found it useful in a variety of contexts, and it is our hope that others find similar value.

As more techniques for differentially ranking features become available, we believe

that Qurro will fit in as a useful piece within the puzzles represented by modern 'omic studies.

1.1.6 Data availability

All data used was obtained from study ID 11721 on Qiita. Deblur output artifact ID 56427 was used, in particular. Sequencing data is also available at the ENA (study accession PRJEB27458). Various Jupyter Notebooks and files used in the creation of this paper are available at https://github.com/knightlab-analyses/qurro-mackerel-analysis.

1.1.7 Acknowledgements

The authors thank the anonymous reviewers for their thoughtful feedback and suggestions on the manuscript. The authors thank Julia Gauglitz, Shi Huang, Franck Lejzerowicz, Robert Mills, Justin Shaffer, Seth Steichen, Bryn Taylor and Yoshiki Vázquez-Baeza for helpful comments on the tool's functionality and design. The authors thank Gail Ackermann for assistance with study metadata. The authors thank Jerry Kennedy, Yoshiki Vázquez-Baeza, and Austin Swafford for assistance with funding information. The authors thank Jake VanderPlas, Dominik Moritz and the rest of the Altair/Vega development teams for answering questions regarding their software. Finally, the authors thank Sarah Allard, Tomasz Kościółek, Franck Lejzerowicz, Anupriya Tripathi and Yoshiki Vázquez-Baeza for help naming the tool.

Chapter 1.1, in full, is a reprint of the material as it appears in "Visualizing 'omic feature rankings and log-ratios using Qurro." **Fedarko MW**, Martino C, Morton JT, González A, Rahman G, Marotz CA, Minich JJ, Allen EA, and Knight R. *NAR Genomics and Bioinformatics 2*(2), 2020. The dissertation author was the primary investigator and first author of this paper.

1.1.8 Funding

This work is partly supported by IBM Research AI through the AI Horizons Network and the UC San Diego Center for Microbiome Innovation (to R.K. and M.W.F.); Joint University Microelectronics Program (JUMP)'s Center for Research on Intelligent Storage and Processing-in-memory (CRISP) [GI18518 to R.K. and M.W.F.]; University of California San Diego Computer Science and Engineering Department (to M.W.F.); University of California San Diego Frontiers of Innovation Scholars Program (to C.M.); National Science Foundation [GRFP DGE1144086 to J.T.M.]; National Institute of Dental and Craniofacial Research through F31 Fellowship [1F31DE028478 to C.A.M.]; University of California San Diego Center for Microbiome Innovation through Microbial Sciences Graduate Research Fellowship (to J.J.M.).

1.1.9 Conflict of interest statement

None declared.

1.2 EMPress enables tree-guided, interactive, and exploratory analyses of multi-omic data sets

1.2.1 Abstract

Standard workflows for analyzing microbiomes often include the creation and curation of phylogenetic trees. Here we present EMPress, an interactive web tool for visualizing trees in the context of microbiome, metabolome, and other community data scalable to trees with well over 500,000 nodes. EMPress provides novel functionality—including ordination integration and animations—alongside many standard tree visualization features and thus simplifies exploratory analyses of many forms of 'omic data.

1.2.2 Importance

Phylogenetic trees are integral data structures for the analysis of microbial communities. Recent work has also shown the utility of trees constructed from certain metabolomic data sets, further highlighting their importance in microbiome research. The ever-growing scale of modern microbiome surveys has led to numerous challenges in visualizing these data. In this paper we used five diverse data sets to showcase the versatility and scalability of EMPress, an interactive web visualization tool. EMPress addresses the growing need for exploratory analysis tools that can accommodate large, complex multi-omic data sets.

1.2.3 Introduction

The increased availability of sequencing technologies and automation of molecular methods have enabled studies of unprecedented scale [189], prompting the creation of tools better suited to store, analyze [18], and visualize [200] studies of this magnitude. Many of these tools, including for example UniFrac [110], phylofactor [207], PhILR [178], and Gneiss [133], make use of tree structures in some way: often these structures are phylogenetic trees detailing the evolutionary relationships among features in a data set, although this category also includes general dendrograms that organize features in a hierarchical structure (e.g., clustering of mass spectra) [194]. The challenge of enabling fully interactive analyses stems from the disconnect between tools that focus on features (for example, microbial relative abundances) and tools that focus on samples (for example, alpha diversity distributions). In addition, few tools can interactively integrate multiple representations of the data side-by-side [136] while scaling to display large data sets. We view this as a key unresolved challenge for the field: to contextualize community-level patterns (groupings of samples) together with feature-level structure, i.e., which features lead to the groupings explained in a given sample set.

Although many useful phylogenetic visualization and analysis tools are available,

few focus on community analysis tasks. The current state of the art includes specialized tools like Anvi'o [44], which consolidates a large collection of methods for sequence-based analysis and visualization of metagenomic assembled-genomes, pangenomes, and proteins (among many other data types). The state of the art also includes more general-purpose tree visualization tools like PHYLOViZ [136], SigTree [183], and iTOL [104] (among many others). Although tree structures are usually stored in standard file formats like Newick, the data accompanying these trees—for example, tip-level taxonomic classifications or other metadata values—are less standardized and sometimes require the onerous creation of configuration files. Furthermore, some types of exploratory analyses are not easily possible: for example, ordination plots computed from UniFrac [110] distances (or other phylogenetically informed distances) are often used to visualize sample clustering patterns in microbiome studies. However, interpreting the patterns in these plots—and determining which features influence the separation of certain groups of samples—is not always straightforward. While biplots can improve legibility by showing information about influential features alongside samples, the phylogenetic relationships of these features are not always obvious.

To address these and other outstanding gaps in the state of the art, we introduce EMPress (https://github.com/biocore/empress), an open-source (BSD 3-clause), interactive and scalable phylogenetic tree viewer accessible as a QIIME 2 [18] plugin or as a standalone Python program. EMPress is built around the high-performance balanced parentheses tree data structure [31] and uses a hardware-accelerated WebGL-based rendering engine that allows EMPress to visualize trees with hundreds of thousands of nodes from within a laptop's web browser (see section 1.2.6, "Materials and Methods"). EMPress visualizations can be created solely from a tree, or users can optionally provide additional metadata files and a feature table to augment the visualization. Additionally, through integration with the widely-used EMPeror software [200], EMPress can simultaneously visualize a study's phylogenetic tree alongside an ordination plot of the samples in the data

set (in what we colloquially term an "EMPire plot"). User actions in one visualization, such as selecting a group of samples in the ordination, update the other (in this case, highlighting the portions of the tree corresponding to these samples), providing context that would not be easily accessible with independent visualizations. This tight integration between displays streamlines several use-cases elaborated below that previously required manual investigation or writing custom scripts.

1.2.4 Results

Rather than providing a programmatic interface for the procedural generation of styled phylogenetic trees [216, 77, 158], EMPress provides an interactive environment to support exploratory feature- and sample-level tree-based analyses. One of the ways EMPress stands out is in its scalability in comparison to other web-based tree viewers: iTOL [104] claims trees with more than 10,000 tips to be "very large" (https://itol.embl.de/help.cgi), while EMPress readily supports trees with over hundreds of thousands of tips, as shown in Figure 1.3. Many visualization customization options available in EMPeror [200], iTOL [104], and Anvi'o [44] are immediately accessible in EMPress' interface. Continuous metadata associated with the tips of the tree can be visualized as barplots with a color gradient and/or by mapping each value to the height of each bar. Similarly, categorical sample metadata information can be visualized using a stacked barplot showing—for each tip—the proportion of samples containing that tip stratified by category. These options are available in EMPress' user interface, based on the data provided by the user to EMPress when creating a visualization, and—providing data files are stored in an accepted format—do not require programming or the creation of configuration files.

EMPress also aids interpretation of ordination plots by optionally providing a unified interface where the tree and ordination visualizations are displayed side-by-side and "linked" through sample and feature identifiers [12]. This combination of EMPress and EMPeror [200] allows for many novel exploratory data analysis tasks. For example,

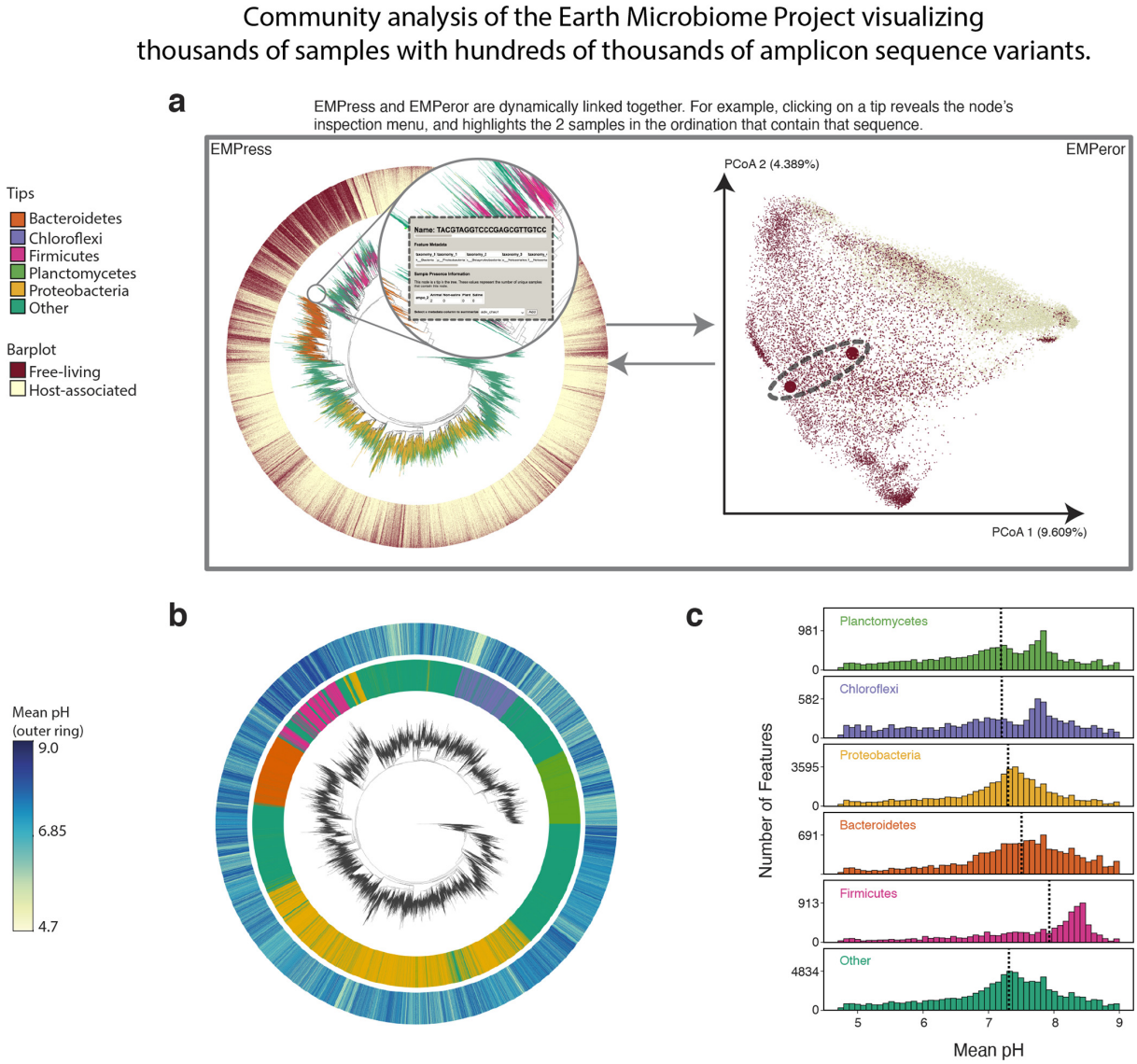


Figure 1.3. Earth Microbiome Project paired phylogenetic tree (including 756,377 nodes) and unweighted UniFrac ordination (including 26,035 samples). **(A)** Graphical depiction of EMPress' unified interface with fragment insertion tree (left) and unweighted UniFrac sample ordination (right). Tips are colored by their phylum-level taxonomic assignment; the barplot layer is a stacked barplot describing the proportions of samples containing each tip summarized by level 1 of the EMP ontology. Inset shows summarized sample information for a selected feature. The ordination highlights the two samples containing the tip selected in the tree enlarged to show their location. **(B)** Subset of EMP samples with pH information: the inner barplot ring shows the phylum-level taxonomic assignment, and the outer barplot ring represents the mean pH of all the samples where each tip was observed. **(C)** pH distributions summarized by phylum-level assignment with median pH indicated by dotted lines. Interactive figures can be accessed at https://github.com/knightlab-analyses/empress-analyses.

selecting a group of samples in the ordination highlights nodes in the tree present in those samples and vice versa (see section 1.2.6, "Materials and Methods"). This integration extends to biplots: clicking feature arrows in the ordination highlights the corresponding node in the tree. Lastly, EMPress supports the visualization of longitudinal studies by simultaneously showing tree nodes unique to groups of samples at each individual time point during an EMPeror animation (see section 1.2.6, "Materials and Methods").

Scalability: visualizing data from the Earth Microbiome Project.

Using the first data release of the Earth Microbiome Project (EMP), we demonstrate EMPress' scalability by rendering a 26,035-sample ordination and a 756,377-node tree (Figure 1.3A). We also demonstrate EMPress' ability to annotate large tree visualizations with categorical "feature" (i.e., corresponding to nodes in the tree) metadata: to visualize the relative proportions of taxonomic groups at the phylum level, we use EMPress' feature metadata coloring to color tips in the tree by their phylum-level taxonomic classifications (see section 1.2.6, "Materials and Methods"). We extend this further by demonstrating EMPress' ability to visualize sample-level categorical metadata: we add a barplot layer showing, for each tip in the tree, the proportions of samples containing each tip summarized by level 1 of the EMP ontology ("Free-living" and "Host-associated").

The paired "EMPire plot" visualizations supported by EMPress' integration with EMPeror allow users to click on a tip in the tree (in EMPress) and view the samples that contain that feature in the ordination (in EMPeror). Clicking on an internal node in the tree functions similarly, showing all samples that contain any of the descendant tips of this node. These actions, and other functionality unique to these paired visualizations, are especially useful when analyzing data sets with outliers or mislabeled metadata. Figure 1.3A shows an example of this functionality in practice, in which the samples in which a tip in the tree is present are highlighted dynamically in an ordination.

EMPress' barplots can also be used to summarize environmental metadata: as a

demonstration of this, Figure 1.3B shows the subset of samples (4,002) with recorded pH information and a barplot layer with the mean pH where each feature was found. The barplot reveals a relatively dark section near many tips classified in the phylum *Firmicutes*; in concert with histograms showing mean pH for each phylum (Figure 1.3C), we can confirm that *Firmicutes*-classified sequences are more commonly found in higher-pH environments. These and the other observations highlighted here indicate the utility of EMPress for exploratory analyses of large, complex data sets.

Versatility: visualizing diverse types of data.

EMPress—both in the visualization of tree structures and in the visualization of various types of metadata alongside these structures—can be applied to many types of "'omic" data sets. To illustrate this versatility, we reanalyzed a COVID-19 metatranscriptome sequencing data set [177], a liquid chromatography-mass spectrometry (LC-MS) untargeted metabolomic food-associated data set [194], and a 16S rRNA gene sequencing oral microbiome data set [132]. Despite the vastly different natures of these data sets, EMPress provides meaningful functionality for their analysis and visualization. Movie S1 (Appendix A.2.2, "Animated analysis of SARS-CoV-2") shows a longitudinal exploratory analysis using EMPress and EMPeror representing a subset of SARS-CoV-2 genome data from GISAID. This paired visualization emphasizes the relationships in time and space among "community samples" and the convergence of locales in the United States with the outbreak in Italy (see section 1.2.6, "Materials and Methods"). The interactive nature of EMPress allows rapid visualization of strains observed in a collection of samples from different geographical locations.

Figure 1.4A showcases EMPress' ability to identify feature clusters that are differentially abundant in COVID-19 patients compared to community-acquired pneumonia patients and healthy controls [177]. Clades showing KEGG enzyme code (EC) [87] annotations are collapsed at level two except for lyases, highlighting feature 4.1.1.20 (carboxy-lyase

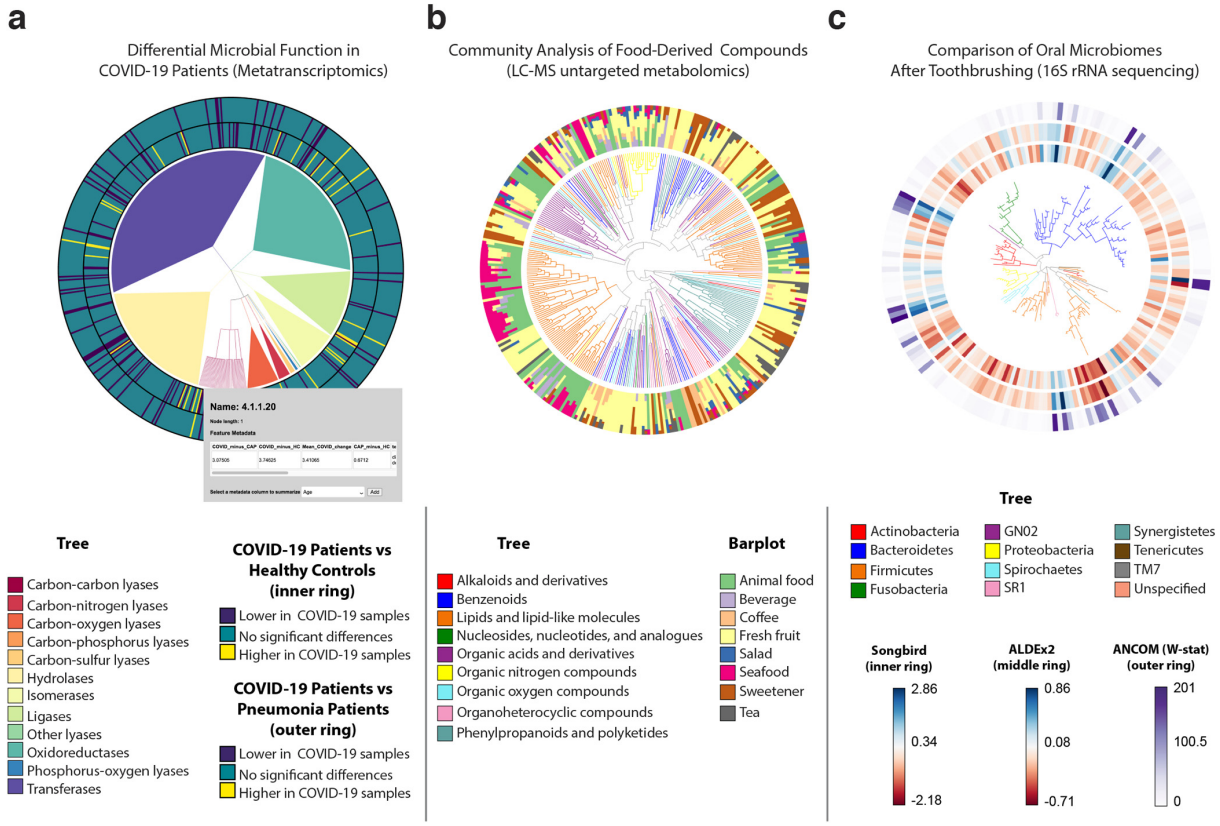


Figure 1.4. EMPress is a versatile exploratory analysis tool adaptable to various 'omics data types. **(A)** RoDEO differential abundance of microbial functions from metatranscriptomic sequencing of COVID-19 patients ($n = 8$) versus community-acquired pneumonia patients ($n = 25$) and versus healthy control subjects ($n = 20$). The tree represents the four-level hierarchy of the KEGG enzyme codes. The barplot depicts significantly differentially abundant features ($P < 0.05$) in COVID-19 patients. Clicking on a tip produces a pop-up insert tabulating the name of the feature, its hierarchical ranks, and any feature annotations. **(B)** Global FoodOmics Project LC-MS data. Stacked barplots indicate the proportions of samples ($n = 70$) (stratified by food) containing the tips in an LC-MS Qemistree of food-associated compounds, with tip nodes colored by their chemical superclass. **(C)** *De novo* tree constructed from 16S rRNA sequencing data from 32 oral microbiome samples. Samples were taken before ($n = 16$) and after ($n = 16$) subjects ($n = 10$) brushed their teeth; each barplot layer represents a different differential abundance method's measure of change between before- and after-brushing samples. The innermost layer shows estimated log-fold changes produced by Songbird, the middle layer shows effect sizes produced by ALDEx2, and the outermost layer shows the W-statistic values produced by ANCOM (see section 1.2.6, "Materials and Methods"). The tree is colored by tip nodes' phylum-level taxonomic classifications. Interactive figures can be accessed at https://github.com/knightlab-analyses/empress-analyses.

diaminopimelate decarboxylase) that was more abundant in COVID-19 here and in an independent metaproteomic analysis of COVID-19 respiratory microbiomes [116].

Recent developments in cheminformatics have enabled the analysis and visualization of small molecules in the context of a cladogram [194]. Using a tree that links molecules by their structural relatedness, we analyzed untargeted LC-MS/MS data from 70 food samples (see section 1.2.6, "Materials and Methods"). With EMPress' sample metadata barplots, we can inspect the relationship between chemical annotations and food types. Figure 1.4B shows a tree where each tip is colored by its chemical superclass and where barplots show the proportion of samples in the study containing each compound by food type. This representation reveals a clade of lipids and lipid-like molecules that are well represented in animal food types and seafoods. In contrast, salads and fruits are broadly spread throughout the cladogram.

Lastly, in Figure 1.4C, we compare three differential abundance methods in an oral microbiome data set [132] as separate barplot layers on a tree. This data set includes samples ($n = 32$) taken before and after subjects brushed their teeth (see section 1.2.6, "Materials and Methods"). As observed across the three differential abundance tools' outputs, all methods agree broadly on which features are particularly "differential" (for example, a group of sequences classified in the phylum *Firmicutes* in the bottom right of the tree; see section 1.2.6, "Materials and Methods"), although there are discrepancies due to different methods' assumptions and biases.

1.2.5 Discussion

Future work on visualization integration.

By providing an intuitive interface supporting both categorically new and established functionality, EMPress complements and extends the available range of tree visualization software. EMPress can perform community analyses across distinct "'omics" types, as demonstrated here. Moving forward, facilitating the integration of multiple orthogonal

views of a data set at a more generalized framework level (for example, using QIIME 2's [18] visualization API) will be important as data sets continue to grow in complexity, size, and heterogeneity.

Validating visual observations and aiding reproducibility.

It is important to note that making visual observations using EMPress does not eliminate the need for providing statistical support. For example, various layout and branch length options available in EMPress can drastically affect the perceived size of a clade or taxonomic group. We therefore recommend that users remain careful of the need to validate their claims, in order to ensure that conclusions drawn are not solely visual artifacts. In the context of microbial ecology studies, tools like Phylofactor, Gneiss, and PhILR can complement observations made with EMPress well.

Similarly, although the various exploratory (*post hoc*) analyses shown in this paper are simplified by EMPress, they are not a substitute for sound hypothesis-driven science and should not be presented as such [74]. Exploratory analyses, when documented transparently, can be useful to the scientific community [74]; our hope is that, by providing a tool that simplifies these analyses of complex data, EMPress can fulfill a legitimate scientific need. One way we believe EMPress helps fulfill this need is through the inherent shareability of its outputs (see section 1.2.6, "Materials and Methods"). This shareability simplifies the process of reproducing a visualization in EMPress, as well as the interactive exploration of alternative representations of a data set. As an example of this, we have provided QZV files for Figures 1.3 and 1.4 on GitHub (see section 1.2.6, "Materials and Methods"), and we encourage readers to reproduce these figures for themselves. We acknowledge that "reproducibility" of this kind is limited to the files the user provides when creating an EMPress visualization—and since the same data and same methods are used in the reproduction as in the original visualization, this therefore does not necessarily add evidence that conclusions derived from the visualization are completely correct [170].

However, we contend that it can still be beneficial for readers and authors alike, and we hope that by simplifying the sharing of EMPress visualizations we can encourage researchers to share their visualizations and make clear the exploratory nature of their work.

Tree shearing in the context of community data.

In order to aid in the analysis of community data, EMPress—when a feature table is provided—by default shears the tree to include only tips that are found in the feature table. This preprocessing step visually emphasizes samples spanning only a small number of tips within larger reference trees. Since this option may not always be desired—if, for example, the focus of an analysis is to compare novel diversity to a reference database—this option can be disabled from the command line.

1.2.6 Materials and Methods

All EMPress plots in this paper were visualized and stylized in Safari (14.0) or Google Chrome (85.0.4183.121) using a MacBook Pro (15-inch, 2017) with a 2.9-GHz Quad-Core Intel Core i7 7820QM, 16 GB of RAM, a Radeon Pro 560 4 GB, and Intel HD Graphics 630 1,536-MB graphics processor. Data, analyses, and steps to reproduce the figures in this paper can be found at https://github.com/knightlab-analyses/empress-analyses. Due to file size restrictions on GitHub, some files are downloaded by executing the Jupyter notebook associated with each figure.

Earth Microbiome Project.

The EMP release 1 [189] table, tree, and metadata were used to generate the visualization (ftp://ftp.microbio.me/emp/release1). The original feature table was subset to remove sterile water blanks and mock community samples. The table contains the taxonomic assignments used to annotate tips. For ease of visualization, only the top 5 most abundant phylum annotations in the data set were kept while microbial features

annotated with any other phylum (or unspecified) were categorized as "Other." A distance matrix was generated by computing the unweighted UniFrac distances between samples [110, 122] that was then used to generate the principal-coordinate plot.

A subset of the feature table was generated by extracting all samples in the middle 90% of the pH range (4.7 to 9) to remove outliers. Samples without a valid pH value were removed. For each remaining feature, the number of samples ($\log_{10}$) in which this feature occurred and the mean pH of those samples were calculated and saved as a feature metadata file passed into EMPress. Calculations were performed using NumPy v1.18.1 [70] and Pandas v0.25.3 [211, 159]. Distributions of pH were plotted using matplotlib v3.1.3 [24] and seaborn v0.10.0 [208].

COVID-19 metatranscriptome data set.

The COVID-19 bronchoalveolar lavage fluid metatranscriptome sequencing data [177] consists of COVID-19 ($n = 8$), community-acquired pneumonia ($n = 25$), and healthy control ($n = 20$) samples. The tree for this data set corresponds to the KEGG enzyme code (EC) [87] hierarchy. Sequencing reads were processed and annotated with EC feature labels using PRROMenade [195, 67] with a database of bacterial and viral protein domains from the IBM Functional Genomics Platform [172], as previously reported [67]. Differential abundance per feature was determined by performing a Kolmogorov-Smirnov test on average RoDEO-processed [67, 66] values per sample. A cutoff ($P < 0.05$) was applied to focus the visualization on the features that are significantly more abundant or less abundant in COVID-19 patients than in healthy controls and/or community-acquired pneumonia samples.

Global FoodOmics data set.

The untargeted metabolomics data set was generated using a quadrupole time of flight (QTOF) mass spectrometer in positive ionization mode (Bruker). The samples

presented in this data set were processed using Qemistree version 2020.1.1114.g1b4edb4 running in QIIME 2 version 2019.7. For ease of interpretation, the data set was subset to keep features with a superclass assignment and keep samples with a *common meal type* classification. The tip barplots show the proportion of samples where each small molecule is present summarized by *meal type*.

Differential abundance comparison of oral microbiomes.

The oral microbiome 16S rRNA sequencing data used in reference [132] was revisualized for Figure 1.4C. This data set comprises $n = 32$ samples total, taken before and after subjects brushed their teeth. Some paired samples were taken more than once from the same subject; in total, these 32 samples were contributed by 10 unique subjects.

The sequences in this data set were processed (in July 2018) using Deblur v1.0.4 [3] through q2-deblur in QIIME 2 2018.6 [18]. Taxonomic classifications were assigned (in August 2018) using q2-feature-classifier's [17] `classify-sklearn` method [147], using the Greengenes reference database [121], also in QIIME 2 2018.6. The table provided lacks provenance information due to not being stored as a QIIME 2 artifact, but since its features are a subset of those in the sequences file—and since the lowest number of samples that a feature within it is present in is 6—it was likely filtered at some point. To construct a rooted tree from the sequences in this data set (in September 2020), we used QIIME 2 2019.10's `qiime phylogeny align-to-tree-mafft-fasttree` pipeline [89, 153, 102].

In reference [132], three differential abundance tools were run on this data set, using the "brushing_event" metadata field (indicating before/after toothbrushing status) as the sole field across which to identify differentially abundant features. The three differential abundance tools used in reference [132] and visualized in Figure 1.4C are Songbird [132], ALDEx2 [49], and ANCOM [113]. Songbird's column of feature differentials (describing the estimated log-fold changes of each feature between the "after" and "before" brushing states) is shown as the innermost barplot layer in Figure 1.4C; ALDEx2's per-feature effect

size is shown as the middle layer; and ANCOM's per-feature W-statistic is shown as the outermost layer. For both Songbird and ALDEx2 results, higher values indicate association with before-brushing samples (i.e., features that decreased most from toothbrushing, for example, secondary metabolizers present on the outer layers of dental plaque biofilms such as *Haemophilus*) while lower values indicate association with after-brushing samples (i.e., features that decreased least from toothbrushing, for example, primary metabolizers such as *Actinomyces* that are rooted at the base of the biofilm) [132]. ANCOM's W-statistic corresponds to the number of log-ratio hypothesis tests in which a given feature was found to be differentially abundant between before- and after-brushing samples [113] (https://forum.qiime2.org/t/1844/10). Since Songbird and ALDEx2's results include directionality between before and after brushing, they are shown in Figure 1.4C with a "diverging" color map; ANCOM's W-statistic does not include this information and is therefore shown with a "sequential" color map (see Figure A.3 in Appendix A.2.1, "Differential abundance comparison of oral microbiomes").

We note that the Songbird results from reference [132] did not include differentials for nine features in the data set; this may have been due to software bugs or other unknown factors in the data analysis, since (although Songbird does filter out features present in less than a given number of samples) these absent features are present in the same number of samples as other features which were included in the Songbird differentials. For the sake of simplicity here, and since the purpose of this subfigure is primarily to demonstrate the utility of EMPress in the context of existing data, we simply reused the data from reference [132], filtering these nine features out of the data set before constructing and visualizing the tree.

Animated analysis of SARS-CoV-2.

The GISAID [39] SARS-CoV-2 genome alignment and genome metadata were obtained on 21 September 2020. Sequences were converted to DNA and subset to the set

of sequences associated with Italy, Madrid, King County, San Diego, Brooklyn, Queens, and Manhattan. Highly gapped and high-entropy positions in the alignment were filtered using q2-alignment (2020.6; default parameters). A tree was estimated using FastTree [153] (v2.1.10 compiled with double precision support; default options except `-fastest`) and subsequently rooted using midpoint rooting as implemented by q2-phylogeny (2020.6; default parameters).

Separately, a sliding window procedure was developed to assess the observed SARS-CoV-2 genomes within a given time period within a geographic location. To do so, the metadata were partitioned into the respective locations (note that the three New York boroughs were treated as New York) and ordered by the genome date information. A sliding window width of 7 days was used, and a sample was retained only if five or more strains were observed within a window. These windows were then aggregated into a BIOM table [120] with the GISAID strain identifier on one axis and a "community sample" identifier on the other. Unweighted UniFrac (q2-diversity 2020.6 [18, 122]) was then computed over these samples followed by a principal-coordinate analysis. The tree and ordination were visualized with a development version of EMPress (version 0.3.0-dev), and therefore, the visualization shown in the video may look slightly different from more recent versions of EMPress.

Implementation details.

EMPress is implemented as a QIIME 2 plugin (or standalone Python program, usable outside QIIME 2) capable of generating HTML documents with a self-contained visualization user interface. The code-base is composed of a Python component and a JavaScript component. The Python code-base is responsible for data validation, preprocessing, filtering, and formatting. User interaction, rendering, and figure generation are all handled by the JavaScript code-base. In both cases, we rely on the balanced parentheses data structure [31] to rapidly operate on the tree structures.

EMPress' Python code-base currently uses NumPy [70], SciPy [203], Pandas [211, 159], Click (https://palletsprojects.com/p/click/), Jinja2 (https://jinja.palletsprojects.com/), scikit-bio (http://scikit-bio.org), the BIOM format [120], iow (https://github.com/wasade/improved-octo-waddle) [31], and EMPeror [200]. The JavaScript code-base uses Chroma.js (https://gka.github.io/chroma.js/), FileSaver.js (https://github.com/eligrey/FileSaver.js/), glMatrix (http://glmatrix.net/), jQuery (https://jquery.com/), Require.js (https://requirejs.org/), Spectrum (https://bgrins.github.io/spectrum/), and Underscore.js (https://underscorejs.org/). For testing and linting, EMPress' Python code-base uses flake8 (https://flake8.pycqa.org/en/latest/) and nose (https://nose.readthedocs.io/), and EMPress' JavaScript code-base uses QUnit (https://qunitjs.com/), qunit-puppeteer (https://github.com/davidtaylorhq/qunit-puppeteer/), jshint (https://jshint.com/about/), and Prettier (https://prettier.io/).

As of writing, EMPress supports drawing trees using three standard layout algorithms ("rectangular," "circular," and "unrooted"), coloring the tree using sample and feature metadata, collapsing clades based on common metadata values, adding tip-aligned barplots for sample and feature metadata, and summarizing tips' presence within sample groups using interactive node selections. This is in addition to integration with EMPeror, described further below, as well as various other visualization options.

EMPress' "unrooted" layout algorithm is translated from code from Gneiss [133], which was in turn adapted from PyCogent [96], and is an implementation of the equal-angle algorithm described in reference [47]. EMPress' "rectangular" and "circular" layout algorithms are adapted from code from TopiaryExplorer [150] and resemble the rooted tree drawing algorithms described in reference [47]. EMPress also includes the ability to reorder sibling clades in the rectangular and circular layouts by the number of tips contained within each clade; this functionality was inspired by iTOL's [104] "leaf sorting" option and uses tree traversal code adapted from scikit-bio (http://scikit-bio.org).

In order to integrate EMPress and EMPeror, we link together events triggered

by each of the applications by inserting "callback" code that can be executed in one application when a given event occurs. These events notify each tool that a particular action needs to take place, and if needed, what data should be used in this context. For example, when a user selects a group of samples in EMPeror, the "select" event is triggered with a collection of sample objects. EMPress responds to this event by searching for the tips in the tree corresponding to features contained within these samples and updates the color according to the object's attributes. The subscription mechanism also enables users to select a node in EMPress to highlight the samples containing this node or one of its descendant tips in EMPeror, link biplot [2] arrows in EMPeror to nodes in the tree, highlight groups by double-clicking a category in EMPeror's color legend, and synchronize animated ordinations [199] by coloring the tree according to the current frame on screen.

Sharing EMPress visualizations.

When used as a QIIME 2 [18] plugin, EMPress generates visualizations in the QZV format (which can be viewed using https://view.qiime2.org, in addition to other methods); when used outside QIIME 2, EMPress creates visualizations as a directory (containing an HTML file that can be opened to show the visualization). In either case, an EMPress visualization is easily shareable with a wide audience of users who may not have EMPress or QIIME 2 installed, for example, via uploading the visualization file(s) to GitHub or by hosting the file(s) on any other website. We note that the ability to share visualizations is not unique to EMPress and is also inherent to other QIIME 2 [18] plugins and in other tree visualization tools like iTOL [104].

1.2.7 Acknowledgements

We thank members of the Knight Lab and IBM AIHL Bioinformatics team for feedback during code reviews and presentations. We gratefully acknowledge the authors from the originating laboratories responsible for obtaining the specimens and the sub-

mitting laboratories where genetic sequence data were generated and shared via the GISAID Initiative, on which a portion of this research is based; these authors are listed in Supplemental Table S1, available online at https://doi.org/10.1128/mSystems.01216-20.

This work is partly supported by IBM Research AI through the AI Horizons Network and the UC San Diego Center for Microbiome Innovation (to K.C., M.W.F., J.T.M., Q.Z., G.A., T.Z., J.G.S., A.D.S., S.J.S., Y.V.-B., and R.K.); CCF foundation no. 675191 (R.K. and P.C.D.), U19 AG063744 01 (R.K., J.M.G., and P.C.D.), CDC contract no. 75D30120C09795 (K.G.A. and R.K.), and U19 AI135995 (K.G.A.).

K.C., Q.Z., Y.Y., J.T.M., T.Z., J.G.S., and R.K. conceived the original idea for the project. K.C., M.W.F., G.R., D.M., A.G., S.J., M.E., Y.Y., E.S., J.T.M., G.A., T.Z., Q.Z., and Y.V.-B. wrote source code and/or documentation for the project. K.C., M.W.F., A.G., and Y.V.-B. wrote code to facilitate integration with EMPeror. L.P., H.-C.K., S.J.S., A.D.S., Y.V.-B., and R.K. managed the project. K.C., M.W.F., G.R., N.H., K.L.B., A.T., J.M.G., A.P.C., N.L.M., C.M., P.C.D., K.G.A., L.P., and Y.V.-B. analyzed and interpreted the data sets presented in this paper. K.C., M.W.F., G.R., D.M., N.H., K.L.B., and Y.V.-B. contributed text to section 1.2.6, "Materials and Methods". All the authors contributed to the final version of the manuscript.

We declare no competing interests.

Chapter 1.2, in full, is a reprint of the material as it appears in "EMPress Enables Tree-Guided, Interactive, and Exploratory Analyses of Multi-omic Data Sets." Cantrell K, **Fedarko MW**, Rahman G, McDonald D, Yang Y, Zaw T, Gonzalez A, Janssen S, Estaki M, Haiminen N, Beck KL, Zhu Q, Sayyari E, Morton JT, Armstrong G, Tripathi A, Gauglitz JM, Marotz C, Matteson NL, Martino C, Sanders JG, Carrieri AP, Song SJ, Swafford AD, Dorrestein PC, Andersen KG, Parida L, Kim H-C, Vázquez-Baeza Y, and Knight R. *mSystems 6*(2), 2021. The dissertation author was a primary investigator and co-first author of this paper.

Chapter 2

Metagenome assembly of long and accurate reads using iterative downsampling

2.1 Abstract

Long and accurate reads have revolutionized genome assembly, enabling "telomere-to-telomere" assembly projects for a variety of genomes. Despite these advances, metagenome assembly remains a challenging problem; many assemblers, even those designed specifically in the context of long and accurate metagenomic reads, produce fragmented assemblies that omit or misrepresent real sequences in the input data. To address these challenges we present metaLJA, a metagenome assembler based on the state-of-the-art LJA algorithm for single-genome de Bruijn graph assembly of long and accurate reads. metaLJA uses iterative rounds of downsampling, assembly, and graph simplification in order to produce assemblies of groups of similar genomes within a metagenome.

2.2 Introduction

The past few decades' refinement of sequencing technologies and assembly algorithms has led to substantial improvements in our ability to assemble genomes—from single genomes [139] to metagenomes [15]. The recent development of sequencing technologies

that can produce long and accurate reads, such as Pacific Biosciences' high-fidelity (HiFi) reads [210], has played a critical role in this shift. Longer reads can span additional repetitive regions and disambiguate the assembly graph [58]; more accurate reads can enable the more sensitive identification of whether variable positions in the assembly are attributable to sequencing error or to real variation [125].

Even armed with long and accurate reads, however, generating accurate metagenome assemblies remains challenging. Uneven sequencing coverage, intergenomic repeats, and strain-level variation all add further difficulties in addition to those faced in single-genome assembly [97, 138]. Furthermore, current benchmarking methods do not sufficiently describe the scope of errors in metagenome assemblies; a recent study demonstrated that multiple state-of-the-art long-read metagenome assemblers [97, 141, 48, 13] produced outputs containing subtle errors, such as chimeric contigs and prematurely circularized sequences [193].

In 2022 our lab published LJA [8], an assembler for long and accurate reads. By enabling efficient construction of the de Bruijn graph [30] using large K-mer sizes, among various other algorithmic techniques, LJA supports the automatic assembly of multiple human chromosomes. However, LJA is designed for single-genome assembly; it struggles to assemble metagenomic datasets, in which the coverage and ploidy of the underlying genomes can both vary wildly.

To extend LJA to the problem of metagenome assembly we describe metaLJA, a tool that uses LJA to approach this problem indirectly. By downsampling a metagenome to a fraction of its reads, assembling these reads using LJA, smoothing these reads to match the assembly, and iteratively repeating this process with larger and larger downsamplings, metaLJA is able to reconstruct multiple complete genomes from long and accurate metagenomic reads.

2.3 Results

Here we demonstrate metaLJA by applying it to two datasets of HiFi reads sequenced from metagenomes—a mock community and a chicken gut. These datasets were among those used as benchmarks in [48], and are useful case studies here as well.

We note that the assembly algorithms used by metaLJA are still in active development; the results presented herein illustrate that there is substantial room for the tool to be improved.

2.3.1 Assembly of a mock community

Assembling sequencing reads from a mock community—that is, a sample of microbes with a known composition—provides a reasonable (albeit not comprehensive [193]) way to evaluate the performance of a metagenome assembler. The paper introducing hifiasm-meta [48] ran various metagenome assemblers on sequencing reads from the mock community ATCC MSA-1003 (GenBank accession number SRR11606871), originally produced in [75]. To get a sense for how metaLJA compares with these other assemblers, we ran metaLJA on this same dataset; Figure 2.1 visualizes the final assembly graph output by metaLJA.

Since this mock community contains 20 bacterial genomes, an ideal assembly graph produced from these reads would consist of 20 circular contigs, each in its own connected component of the graph (in addition to a handful of other components consisting of short linear or circular contigs corresponding to plasmid sequences, the second chromosome of *Rhodobacter sphaeroides* [184], etc.).

For the sake of brevity, we define a *long circle* as a connected component of the assembly graph containing a single circular contig with length $\geq$ 1 Mbp. This length threshold resembles the typical distribution of bacterial genome lengths [36]; we note that metaLJA currently outputs homopolymer-compressed sequences, which may result in us underreporting the number of long circles in metaLJA's output. Erring on the side of

Figure 2.1. metaLJA final assembly graph of HiFi reads sequenced from the ATCC MSA-1003 mock community [75], visualized using Bandage [212]. This mock community is comprised of 20 genomes at varying abundances (https://web.archive.org/web/20211016063400/https://www.atcc.org/products/msa-1003). We note that we ran metaLJA with whirl simplification turned off.

being conservative seems reasonable here.

metaLJA's final assembly graph for this mock community, as shown in Figure 2.1, contains 13 long circles. This count is comprable to the outputs of other long-read metagenome assemblers shown in [48]: hifiasm-meta [48]'s assembly graph contains 15 long circles, and metaFlye [97]'s assembly graph contains 14 long circles. We note, however, that the number of long circles alone is not a sufficient metric for evaluating assembly quality—for example, the assembly graph produced by hifiasm-meta for this dataset, as discussed in [193], is surprisingly complex (containing 10,346 nodes and 248 edges).

2.3.2 Assembly of a chicken gut metagenome

To give another view of metaLJA's performance, we used it to assemble sequencing reads from a chicken cecum metagenome sample (GenBank accession number SRR15214153) [48, 217] that was also used to benchmark hifiasm-meta [48]. Figure 2.2 visualizes the final assembly graph output by metaLJA for this dataset.

Unlike the mock community, the "ground truth" microbial composition of this sample is not known. However, the metaLJA seems more fragmented than ideal; we note the strangely large amount of isolated long linear sequences in its assembly. As shown in [48], hifiasm-meta [48]'s assembly of this dataset produced 73 near-complete or > 1 Mbp circular contigs; HiCanu [141]'s assembly produced 37 near-complete or > 1 Mbp circular contigs; and metaFlye [97]'s assembly produced 35 near-complete or > 1 Mbp circular contigs. By comparison, metaLJA was only able to assemble 11 long circles.

The reasons for why metaLJA's assembly of this dataset produces so many linear contigs are not yet clear; we suspect that metaLJA's removal of putatively-complete sequences from the assembly may be prematurely fragmenting later iterations' assemblies, causing some circular sequences to be represented only as linear contigs. We are currently working to diagnose and address this and other issues in order to bring metaLJA up to par with the state-of-the-art.

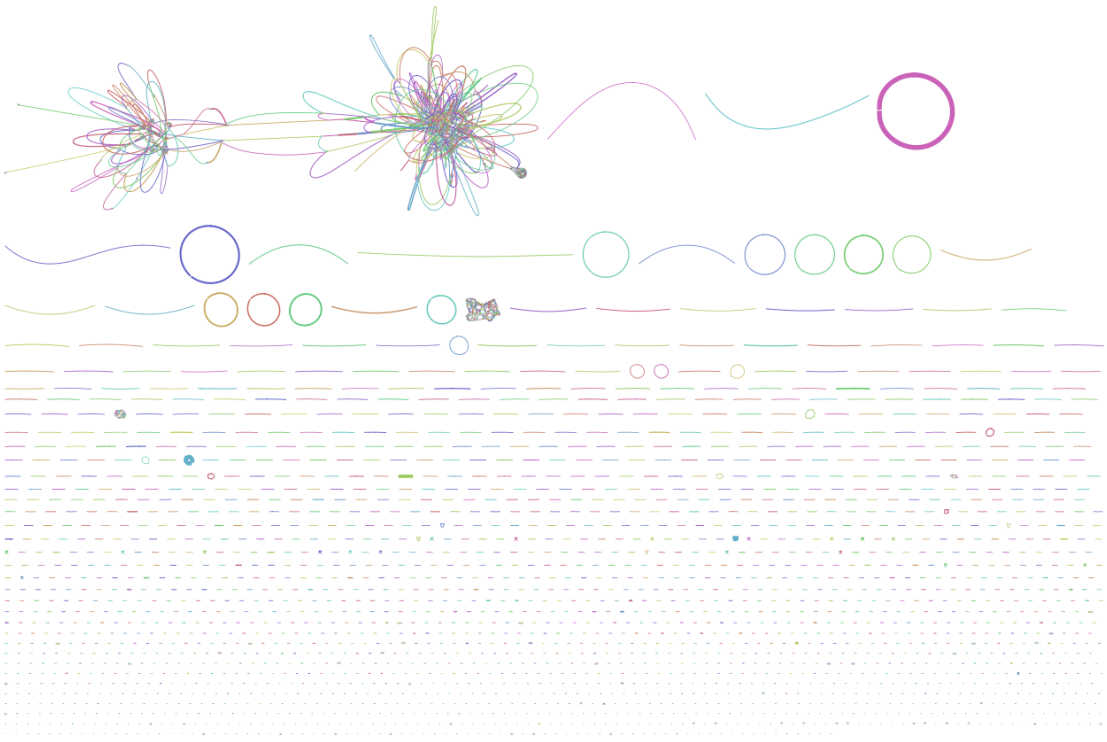


Figure 2.2. metaLJA final assembly graph of HiFi reads sequenced from a chicken cecum metagenome [48, 217], visualized using Bandage [212]. We note that we ran metaLJA with whirl simplification turned off.

2.4 Methods

2.4.1 Iterative downsampling and assembly

Given a *downsampling factor* $D \in \mathbb{Z}^+$, we define a *D-downsampling* of a set of R reads as a random sample consisting of $\lfloor R/D \rfloor$ of these reads. The metaLJA algorithm involves repeatedly performing downsampling operations of increasing size (i.e. decreasing D) on a metagenomic read-set, in order to produce larger and larger manageable assemblies of the reads. (By default, metaLJA uses a fixed random seed in order to ensure that running it multiple times will cause it to make the same D-downsamplings, at least for the initial iteration.)

Given an initial downsampling factor (default $D = 500$), metaLJA's first *assembly iteration* begins by producing a D-downsampling of the input reads and assembling these $\lfloor R/D \rfloor$ reads using LJA [8]. As of writing, metaLJA's downstream steps use the compressed de Bruijn graph output by LJA after error correction but before multiplex de Bruijn graph construction, since the latter graph lacks easily-accessible edge coverages.

After using LJA to construct this de Bruijn graph, metaLJA then repeatedly performs simplification operations on the graph in order to reduce its complexity (section 2.4.2). It then uses minimap2 [106] to align all R reads (potentially ignoring those marked as "finished"; see section 2.4.4) to a subset of selected "reference" edges in the simplified graph. Using this alignment, metaLJA identifies putatively-completely-assembled sequences from the graph and adjusts its internal classifications of reads—including potentially smoothing them to match the assembly—as needed (section 2.4.5).

After processing the alignment of reads to reference edges in the simplified graph, the assembly iteration for this D-downsampling is considered complete. If $D = 1$ (i.e. this iteration considered all input reads), or if no reads remain available for downsampling (section 2.4.4), then metaLJA concludes. Otherwise, work remains to be done: metaLJA performs a new downsampling and assembly iteration using a reduced downsampling factor

of $\lfloor D/2 \rfloor$.

2.4.2 Graph simplification

The de Bruijn graph produced from assembling a large metagenomic read-set is often fairly complex, even after compressing non-branching paths and performing error correction [8]. To reduce the complexity of this graph—and enable the extraction of long contiguous sequences from it for use in alignment—metaLJA performs various simplification operations on the graph to identify and resolve certain kinds of common structures [149], repeating these operations until nothing remains to be simplified in the graph. Here we briefly describe the kinds of structures metaLJA identifies and the ways in which these structures are resolved.

Simple bulges

We define a *simple bulge* as a pair of vertices A and B where there exist at least two edges $A \to B$. (We allow for cyclic simple bulges, i.e. those where $A = B$.) metaLJA resolves a simple bulge from A to B by removing all edges $A \to B$ except for the one with the longest sequence. To preserve interpretability of edge coverages in the graph, metaLJA also adds the coverage of the removed edges from this simple bulge to the coverage of the remaining edge.

Single- and multi-edge path (SME) bulges

We define a *SME bulge* as a pair of vertices A and B where:

1. There exist exactly two paths from A to B.
2. One of these paths is an edge directly from A to B.
3. The other of these paths contains $2 \leq E \leq maxMultiEdgeCount$ edges (default $maxMultiEdgeCount = 10$), and does not return to A.

4. These paths' sequences are similar (section 2.4.3).

5. This bulge is not cyclic, i.e. $A \neq B$.

metaLJA resolves a SME bulge from A to B by removing its "single-edge path"—that is, the lone edge $A \to B$. As with simple bulge resolution, metaLJA adds the coverage of this edge to the coverages of the edges along the remaining "multi-edge path" of this SME bulge.

Simple whirls

We define a *simple whirl* as a pair of vertices A and B where:

1. A and B are not the only vertices in their weakly-connected component.

2. The out-degree of A is exactly one.

3. The in-degree of B is exactly one.

4. There exists exactly one edge $A \to B$.

5. There exists exactly one edge $B \to A$.

6. Coverage$(A \to B) >$ Coverage$(B \to A)$.

7. The length of the sequence defined by $A \to B$ (including both its prefix and suffix K-mer) is less than $longSimpleWhirlRepeatLength$ (default 100 kbp).

metaLJA resolves a simple whirl $A \to B \to A$ using the following procedure:

1. Set Coverage$(A \to B) =$ Coverage$(A \to B) -$ Coverage$(B \to A)$.

2. Replace the edge $B \to A$ with a loop edge $B \to B$ representing the sequence $B \to A \to B$, with coverage equal to Coverage$(B \to A)$.

We note that conditions 2 and 3 in the definition above are important to respect, in order to keep whirl resolution a relatively "safe" procedure that does not substantially change the repeat structure described by the de Bruijn graph. In certain cases (as with the mock community dataset shown in section 2.3.1) we have found that disabling whirl simplification—even when using all conditions described above—seems to improve the quality of metaLJA's assemblies, resulting in the assembly of additional long circular sequences. We are currently investigating the reasons for this phenomenon.

Non-branching paths

We define *non-branching paths* in the same way as maximal non-branching paths are defined in [28]. We identify both linear non-branching paths from A to B (in which all vertices along the path have an in-degree of one and an out-degree of one, with the exception of A and B), and cyclic non-branching paths from A to A (in which all vertices including A have in-degree one and out-degree one).

In both cases, we resolve these structures by replacing all edges along the path with a single edge from starting vertex to the ending vertex of the path. We set the coverage of this edge as the average coverage of its composite edges, weighted by their lengths.

Tips

There are many ways to define "tips" in a de Bruijn graph. To strike a reasonable balance between preserving the structure of mostly linear components while properly simplifying mostly circular components, we describe the following procedure for identifying and removing tips.

Definitions.

We define the *total degree* of a vertex in a directed graph as the sum of this vertex's in-degree and out-degree.

We define an *in-tip* as an edge from $A \to B$ where A has a total degree of one; and we define an *out-tip* as an edge from $A \to B$ where B has a total degree of one.

We define an *in-terminus* as a vertex with at least two incoming edges, at least one of which is an in-tip. Similarly, we define an *out-terminus* as a vertex with at least two outgoing edges, at least one of which is an out-tip.

Finally, we define a *complete (incomplete) in-terminus* as an in-terminus that only (does not only) have in-tips among its incoming edges. We similarly define a *complete (incomplete) out-terminus* as an out-terminus that only (does not only) have out-tips among its outgoing edges.

Resolution.

For each complete in-terminus, remove all adjacent in-tips except the one with the longest length; add the coverages of the removed in-tips of this terminus to the one "preserved" in-tip. Repeat the process analogously for each complete out-terminus.

For each incomplete in-terminus (with n incoming edges that are not in-tips), remove all adjacent in-tips and distribute the total coverage of the removed in-tips equally (dividing this total coverage by n) to the remaining non-in-tip incoming edges. Repeat the process analogously for each incomplete out-terminus.

2.4.3 Classifying sequences as similar or dissimilar

SME bulge resolution (section 2.4.2) can be overzealous, if done carelessly. To avoid resolving structures consisting of two very dissimilar paths that nonetheless begin and end in the same vertex, metaLJA only resolves a SME bulge if its path sequences appear sufficiently similar.

The sequences of two paths in the de Bruijn graph (both starting in a vertex A and ending in a vertex B) will by definition have the same prefix and suffix K-mers. To account for this, metaLJA's similarity-checking procedure extracts the inner sequences

(removing the prefix and suffix K nucleotides) of both paths. We refer to these inner sequences as s_1 and s_2.

If both $|s_1|$ and $|s_2|$ are less than a threshold $minSimilarityLength$ (default 100), then we classify these sequences as similar—this case implies that these sequences are small enough that comparing their similarity using alignment would likely be uninformative.

If at least one of $|s_1|$ or $|s_2|$ is at least $minSimilarityLength$, we use Edlib [181] to align these sequences using edit distance. (If either $|s_1|$ or $|s_2|$ is 0, but the other path's sequence is sufficiently long, we skip alignment and classify these sequences are dissimilar. This case can happen if the prefix and suffix K-mers of a path overlap with each other.)

We define the similarity of the Edlib alignment between s_1 and s_2 using the formula

$$\text{Similarity}(s_1, s_2) = \frac{\text{Number of match operations in Alignment}(s_1, s_2)}{\max(|s_1|, |s_2|)}$$

If $\text{Similarity}(s_1, s_2) \geq minSimilarity$ (default $minSimilarity = 0.9$), we classify s_1 and s_2 as similar. Otherwise, we classify them as dissimilar.

2.4.4 The life cycle of a read

Throughout metaLJA's algorithm, we associate each read with one of five possible "statuses." These classifications are stored internally as five bit arrays, and provide a reasonably elegant way to define the current status of the assembly. Section 2.4.5 describes the precise ways in which reads can be shifted between these statuses when metaLJA processes the alignment of reads to reference sequences; here we describe these statuses at a high level to clarify this process.

We label a read with the **candidate** status if it could be selected using downsampling for an assembly iteration, but is not currently selected. Before the first downsampling is performed, all reads are assigned this status.

When we perform downsampling, we assign these selected reads the **unsmoothed**

status. This is a temporary designation, indicating that these reads will be used in the upcoming assembly as they currently are (without any smoothing being done to them).

While processing the alignment of reads to edges, we may smooth the sequence of a read to match the assembly (and thus reduce the complexity of the next assembly iteration). We assign such reads the **smoothed** status. As the name implies, these reads are analogous to those classified as "unsmoothed"—the next assembly will be produced from reads classified as both unsmoothed and smoothed.

If a read maps to a single reference edge in the assembly at high quality, but if this read did not contribute to the assembly as either an unsmoothed or smoothed read, then we assign this read the **redundant** status—because this read is apparently not required to assemble the sequence to which it maps. To save time, redundant reads will not be used in future assembly iterations; however, they will still be used in future alignments of reads to reference edges.

The reason we still use redundant reads in future alignments is that, in a later iteration, we may classify a reference edge as putatively complete (section 2.4.5). When we classify a reference edge as complete, we assign all reads that map well to it the **finished** status—indicating that these reads should be removed from later assembly iterations, since we have already used them to assemble what seems like a complete sequence.

2.4.5 Identifying and analyzing "reference" edges in the graph

The boundaries of a de Bruijn graph

We define a vertex in the de Bruijn graph as a *boundary vertex* if it has an in-degree of 0 or an out-degree of 0. We define the highest-coverage edge incident on a boundary vertex as the *representative edge* of this boundary vertex.

Even if the representative edge of a boundary vertex is short, including this edge in the alignment of reads to edges is beneficial: by analyzing how reads map to this edge, we can determine if this edge (and, by extension, this component of the de Bruijn graph)

appears to be *extendable*.

Identifying reference edges

We say that an edge in the simplified de Bruijn graph is a *reference edge*—to which we will align reads—if at least one of the following conditions is met:

1. This edge has a length of at least *longLength* nucleotides (default 100 kbp).

2. This edge is representative of a boundary vertex in the graph.

Both of these cases imply that this edge is important enough to merit inclusion in the alignment of reads to edges for this assembly iteration.

Classifying the boundaries of a de Bruijn graph as extendable

The fragmentation of genomes into many contigs is a common problem in metagenome assemblies [55], to the point where many metagenome assemblers' output graphs—even when produced from long and accurate reads—include a large amount of short isolated contigs. One of the fundamental tasks we worked on while developing metaLJA was studying why certain circular genomes were fragmented into separate linear components of the de Bruijn graph.

This problem raises the question, especially when assembling real metagenomic datasets with an unknown "ground truth" microbial composition, of if the boundary vertices (as defined above) of a de Bruijn graph can be extended. This information can be useful both in deciding how to handle these reads during an assembly iteration (in which we have not necessarily seen all reads in the input read-set yet) and in evaluating the quality of an assembly.

Identifying the extending-reads of a boundary-representative edge.

Using minimap2 [106] to align reads to edges in the graph brings with it the advantage of being able to identify "clipped" alignments—in which only part of a read is

mapped to a sequence [188, 193]. Here we consider situations where a clipped read coincides almost perfectly with the end of a boundary-representative edge sequence, indicating that this read seems to extend this boundary of the de Bruijn graph in this direction. We note that we consider "supplementary" alignments output by minimap2, in which different parts of a read may be aligned to multiple places, but exclude "secondary" alignments that represent the same part of a read being mapped to multiple places [188].

We say that a linear alignment of a read to an edge sequence is *start-extending* if this alignment:

1. spans at least $minExtendReadSpan$ (default 3 kbp) on the read,
2. has percent identity of at least $minWellAlignedPID$ (default 95%),
3. is from a suffix of the read (but not the entirety of the read),
4. begins at a 0-indexed position on this edge of less than the amount of bases clipped from the prefix of the read, and
5. begins at a 0-indexed position on this edge of less than or equal to $maxExtendReadDistFromEnd$ (default 20 bp).

Conditions 1 and 2 ensure that this read's alignment is of reasonably high quality, and conditions 3 and 4 ensure that this read extends the prefix of this edge sequence by at least a single base. Condition 5 adds some slight tolerance to account for the case where a read's alignment is not perfectly flush with the start of the edge sequence (for example, if the read has a few mismatches near the start of the edge sequence, the more optimal alignment may start slightly further in on the edge sequence). End-extending alignments can be characterized analogously by adjusting the final three conditions.

Taken together, these conditions allow us to answer the question of whether or not a read seems to extend an edge in the de Bruijn graph—and, further, to answer

the question of whether or not a boundary in the de Bruijn graph is supported by the reads. Boundary-representative edges with large amounts of reads that extend this edge's boundary imply that this de Bruijn graph does not adequately represent the full scope of these reads' sequences.

Defining a boundary-representative edge as extendable.

After considering all alignments of reads to an edge, we define a boundary-representative edge as start-extendable (end-extendable) if the number of start-extending (end-extending) reads of this edge is at least *minExtendReads* (default 10). We note that an edge can be both start-extending and end-extending, for example if it comprises an isolated linear component of the graph.

Using extendability information.

To assist users in understanding the assembly, metaLJA's visualizations of simplified de Bruijn graphs annotate boundary-representative edges with information about the number of extending reads. Additionally, isolated linear edges that are not extendable on either side are labeled as "complete" and removed from later assembly iterations.

Classifying reference edges as complete

In our experience, even small downsamplings are often sufficient to assemble entire genomes (or groups of similar genomes) that were present at extremely high abundance in a sample. To save time from repeating the assembly of these genomes in later iterations, and to prevent these genomes from introducing tangles in later iterations' de Bruijn graphs, metaLJA classifies certain edges in the graph as *complete* and removes certain reads mapping to them from later assembly iterations.

We define an edge as complete if either of the following conditions is met:

1. This edge is a **putatively-complete circular sequence:** it has a length of at least *longLength* nucleotides (default 100 kbp; this is the same parameter as that used in

section 2.4.5), it is located in an isolated weakly-connected component of the graph, and it is circular.

2. This edge is a **putatively-complete linear sequence:** it is not start- or end-extendable, it is located in an isolated weakly-connected component of the graph, and it is linear.

metaLJA saves information about edges that have been classified as complete from each assembly iteration; when outputting the final assembly, it "injects" these complete edges back into the assembly graph in order to accurately represent everything that it was able to assemble.

Processing the alignment of reads to reference edges

Read alignment is not as straightforward as mapping a single read to a single reference sequence [188], particularly in the era of long reads [193]. Here we describe how metaLJA adjusts read statuses (section 2.4.4) while processing the alignment of reads to reference edges, to account for the many ways in which a read can map in practice to these edges. Before describing this logic, we will first define some prerequisite terms that will be useful as we process the alignment.

Well-aligned reads.

We define a read r as *well-aligned* to a set of edge sequences if the following conditions are met:

1. **Consistent orientations:** All of r's alignments to a single edge have consistent orientations—that is, there does not exist some edge c to which both r and ReverseComplement(r) are aligned.

2. **High span:** The fraction of r aligned to the edges is at least $minWellAlignedSpanFraction$ (default 95%).

3. **High percent identity:** The total percent identity of r's alignments to the edges is at least $minWellAlignedPID$ (default 95%).

Singly-well-aligned reads.

We define a read as *singly-well-aligned* to an edge sequence c if the following conditions are met:

1. **Well-aligned:** r is well-aligned to the set of edge sequences, as defined above.
2. **Well-aligned to this single edge:** The fraction of r aligned to specifically c is at least $minWellAlignedSpanFraction$ (default 95%).

Contributing reads.

We define a read as *contributing* to an assembly iteration if it was used when running LJA, either as an unsmoothed or as a smoothed read. This is useful information to have when determining the importance of a read in the assembly process.

Logic for processing the alignment.

Consider a single read r that has not been assigned the status of "finished." In the alignment of reads to reference edges:

1. If r has **a single linear alignment** to an edge c...
    a. If r is **singly-well-aligned** to c...
        i. If c is classified as a **complete sequence**...
            1. Label r as a **finished** read: it will be removed from future assembly iterations and associated with c.
        ii. If c is **not classified as a complete sequence**...
            1. If r **contributed** to this assembly iteration...
                A. Label r as a **smoothed** read: adjust its sequence to match c, and use this smoothed sequence in the next assembly iteration.

2. If r **did not contribute** to this assembly iteration...

   A. Label r as a **redundant** read: it will not be used in future assembly iterations, but it will still be used in future iterations' alignments of reads to edges.

b. If r is **not singly-well-aligned** to c...

   i. If r **contributed** to this assembly iteration...

      1. Label r as an **unsmoothed** read: this read maps to this edge, but it does not map well to it. We will reuse this read in the next assembly iteration. (If this read already contributed as an unsmoothed read in the current iteration, it will remain unsmoothed; if this read was a smoothed read in the current iteration, it will be switched back to an unsmoothed read at this point.)

   ii. If r **did not contribute** to this assembly iteration...

      1. Label r as a **candidate** read.

2. If r has **multiple linear alignments** to the edges...

   a. If r is **singly-well-aligned to a complete sequence** c...

      i. Label r as a **finished** read: it will be removed from future assembly iterations and associated with c.

   b. If r is at least **well-aligned** to the edges...

      i. Label r as a **smoothed** read, *regardless of if it contributed to this iteration*: adjust its sequence to the edges to which it aligned, and use this smoothed sequence in the next assembly iteration. Reads like this are valuable because they can help resolve fragmented regions of the assembly.

   c. If r is **not well-aligned** to the edges...

i. If r **contributed** to this assembly iteration...

1. Label r as an **unsmoothed** read.

ii. If r **did not contribute** to this assembly iteration...

1. Label r as a **candidate** read.

3. If r has **zero linear alignments** to the edges...

a. If r is a **candidate** read or a **unsmoothed** read...

i. Leave r as it is.

b. If r is a **smoothed** read...

i. Label r as an **unsmoothed** read.

c. If r is a **redundant** read...

i. Label r as a **candidate** read.

2.4.6 Software dependencies

metaLJA's code directly relies on the Python packages bitarray (https://github.com/ilanschnell/bitarray), click (https://click.palletsprojects.com/), Edlib [181], NetworkX [65], pyfastx [38], and pysam (https://github.com/pysam-developers/pysam), in addition to LJA [8], minimap2 [106], SAMtools [107, 34], and seqtk (https://github.com/lh3/seqtk).

2.4.7 Software design

metaLJA's code is implemented in Python. As of writing, metaLJA supports all Python versions $\geq$ 3.6; its code is automatically tested using Python 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12, and 3.13.

2.5 Discussion

We have described metaLJA, a tool that uses LJA [8] to perform assemblies of iterative downsamplings of long and accurate reads from a metagenome.

Although this software is already straightforward to use, its results seem to lag behind those of other metagenome assemblers. This is due in part, we suspect, to subtleties in how metaLJA's iterative algorithm adjusts the set of reads used at every iteration: prematurely removing sequences that seem like complete genomes (but are really only parts of larger genomes) could explain why metaLJA produces fragmented assemblies of some circular genomes, since this process "punches holes" in such sequences. We are currently investigating this phenomenon in order to improve metaLJA.

2.6 Acknowledgements

We thank Nuno Bandeira, Melissa Gymrek, Mikhail Kolmogorov, and Siavash Mirarab for feedback and advice.

Chapter 2, in full, is a reprint of the material as it appears in "Metagenome assembly of long and accurate reads using iterative downsampling." **Fedarko MW**, Zhang Z, Bankevich A, and Pevzner PA. *In preparation.* The dissertation author was the primary investigator and first author of this paper.

Chapter 3

Analyzing rare mutations in metagenomes assembled using long and accurate reads

3.1 Abstract

The advent of long and accurate "HiFi" reads has greatly improved our ability to generate complete metagenome-assembled genomes (MAGs), enabling "complete metagenomics" studies that were nearly impossible to conduct with short reads. In particular, HiFi reads simplify the identification and phasing of mutations in MAGs: It is increasingly feasible to distinguish between positions that are prone to mutations and positions that rarely ever mutate, and to identify co-occurring groups of mutations. However, the problems of identifying rare mutations in MAGs, estimating the false-discovery rate (FDR) of these identifications, and phasing identified mutations remain open in the context of HiFi data. We present strainFlye, a pipeline for the FDR-controlled identification and analysis of rare mutations in MAGs assembled using HiFi reads. We show that deep HiFi sequencing has the potential to reveal and phase tens of thousands of rare mutations in a single MAG, identify hotspots and coldspots of these mutations, and detail MAGs' growth dynamics.

3.2 Introduction

3.2.1 Deep DNA sequencing and rare mutations

Representing a species using a "reference genome" alone is an inherently limited approach, because any population of a species—even in an isolate [105], let alone a metagenome—has mutations with highly variable frequencies throughout the genome. In the past decade, deep DNA sequencing has enabled studies of diversity in cell populations that represent a departure from the "reference genome" concept [213]. These studies' foci span many areas, including viral quasi-species [73], bacterial strains [192, 163, 105], cancer evolution [57, 4], cell-free DNA [45], forensic applications [82], and somatic mutations [171, 83].

Previous experimental evolution studies have greatly contributed to our understanding of mutation rates and the emergence of novel bacterial strains [9]. However, they have mainly focused on relatively frequent mutations (e.g., mutations with frequency exceeding 1%) in isolates rather than in metagenomes. Recent studies have shown the importance of detecting rare mutations [192, 105, 220]: The tasks of identifying rare mutations in a microbial population and monitoring how some of them evolve into frequent mutations are important for understanding the dynamics of the acquisition of antibiotic resistance in a pathogen. For example, the presence of a strain with a drug-resistant mutation with a population frequency of 0.1%—a rare, albeit nonzero, frequency—may lead to faster emergence of drug resistance in this population, compared with a population without a single microbial cell carrying this mutation. However, sequencing errors in reads often prevent the detection of rare mutations, particularly when the mutation frequency is less than the error rate in reads [213].

Short-read DNA sequencing has greatly contributed to the detection of rare mutations [164], but phasing these mutations using short reads alone remains challenging: Short-read metagenomic studies rarely result in the complete assembly of even a single

metagenome-assembled genome (MAG) [140]. The recent emergence of the long and accurate Pacific Biosciences (PacBio) high-fidelity (HiFi) reads [210] has led to the era of "complete metagenomics" by enabling complete or nearly complete assemblies of hundreds of MAGs from a single HiFi data set [97, 15, 93]. This shift allows fundamentally new possibilities in the analysis of metagenomes.

3.2.2 Identifying rare mutations in metagenomic data

Any attempt to identify rare mutations must be mindful of their false-discovery rate (FDR): the fraction of identified mutations that are false. In general, a sequencing technology's error rate is not a single fixed value: Error rates in reads differ depending on many factors [213], for example nucleotide substitution types or sequence-specific errors [135, 210]. Identifying rare mutations is therefore a difficult problem that may result in a large FDR.

LoFreq [213] is a variant caller that has been shown to perform well for short-read sequencing data [213, 165, 92]; however, its usual reliance on Phred quality scores limits its applicability to HiFi sequencing data, for which Phred scores are not always available [52]. If Phred scores are not available, LoFreq's "LoFreq-NQ" module attempts to learn error probabilities for the 12 nucleotide substitution types (A $\rightarrow$ C, C $\rightarrow$ A, A $\rightarrow$ G, etc.); these error probabilities are used to inform variant calling in lieu of Phred scores [213]. Although useful, this approach makes the assumption that the probability of a given substitution happening owing to sequencing error—rather than real variation—remains constant for all positions in the genome. This assumption will almost certainly be broken in practice because the effects of a given single-nucleotide mutation vary depending on the position at which this mutation occurs. For example, in the standard genetic code, the mutation A $\rightarrow$ C is synonymous for the third position in the codon CTA, because both CTA and CTC code for leucine; however, this same mutation is nonsynonymous for the third position in the codon CAA because CAA codes for glutamine but CAC codes

for histidine. The probability with which we assume that A $\rightarrow$ C could happen owing to sequencing error could thus be adjusted to account for this biological context; we contend that knowledge about this sort of "context-dependent" information should improve the quality of mutation identifications.

Other technology-agnostic methods besides Lo-Freq NQ, such as DeepVariant [151], could also be applied to the problem of identifying mutations in HiFi metagenomic data; however, there remains potential to take contextual sequence information into account, as discussed, to further improve the algorithms used by these tools. Additionally, machine-learning-based methods that have been trained on human sequencing data may not perform similarly well for metagenomic data [166].

3.2.3 Simulation models of sequencing data

The MetaSim simulation model of Illumina reads [160], which was used in benchmarking LoFreq [213], has proved to be valuable in many applications; however, it—and other simulation models—still has some limitations. For example, extant simulation models do not model the cross talk between wells in the patterned system on Illumina plates [205] or dimness owing to slowly amplifying clusters (affected by insert size or GC bias). Many bioinformaticians may not be aware about these small variations in error rates, because they hardly affect base-calling in reference genomes; however, they nonetheless affect the identification of rare mutations and thus make estimating the FDR of these identifications challenging.

Developing realistic simulation models for HiFi reads represents a similar challenge [142]. There have been recent developments on this front, including Sim-It [37] and HIsim [134, 185]; however, we will propose an approach for estimating the FDRs of identified mutations that can bypass the need for a simulation model and should be applicable to arbitrary long and accurate sequencing data.

3.2.4 Phasing rare mutations in HiFi metagenomic data

Given a set of identified mutations in a metagenome, the next step in assembling strains of a microbe is phasing—for example, answering the question of whether identified mutations at N positions within a MAG represent N alternate strains of this MAG (each with a single mutation), a single alternate strain with N mutations, or something in between. This strain separation problem [202] is important because strains with small genomic differences may have vastly different phenotypes: for example, although some *Escherichia coli* strains are harmless, others may cause disease outbreaks [51, 202]. The detection and phasing of rare mutations are prerequisites for numerous downstream applications, such as monitoring rare drug-resistant subpopulations [105]. It is therefore important to identify as many rare mutations as possible—and to determine which of them are carried together—while simultaneously controlling the FDR of these identifications.

We define a strain as a unique haplotype supported by the available sequencing data, an approach analogous to that taken by [138] and [202]. The field of metagenomic phasing is still in its early stages: Although previous studies have succeeded in separating strains at the level of individual genes or other "genomic features" [27, 111, 155, 138], strain separation at the level of a long region or even a complete genome is not yet a solved problem, especially for high-complexity metagenomes [202].

Even though long-read technologies promise to resolve complete genomes of strains within a bacterial community [14, 15, 48, 176], many long-read assemblers suppress rather than reveal small variations to achieve high assembly contiguity [100, 98, 15]. Exceptions include the metaFlye assembler [97], which includes a "strain mode" (the `--keep-haplotypes` flag) aimed at strain detection, and Strainberry [202], which starts from a "strain-oblivious" assembly and iteratively phases it to generate a "strain-aware" assembly. A critical factor for the success of strain separation in both metaFlye and Strainberry is the amount of variation between strains—these tools typically succeed when

the percent identity between the strains is below 97%–99% and when there are very few (typically two to three) strains. Another important concern is the FDR of the identified rare mutations, because false-positive mutations mislead strain separation tools. The success of strain separation is thus dependent both on the reliable identification of rare mutations and on the ability to control their FDR.

3.2.5 The strainFlye pipeline

Here we present strainFlye, a pipeline for the FDR-controlled identification, phasing, and analysis of rare mutations in MAGs sequenced using long and accurate reads. Our approach draws on seemingly unrelated yet very relevant prior work on estimating the FDR of peptide identifications in proteomics. The first approach to peptide identification was proposed in 1994 [43] and was followed by many other tools addressing the same problem. However, it remained unclear how accurate these tools were and how to benchmark them: The target-decoy approach (TDA) for computing the FDR of peptide identifications, which revolutionized the field of proteomics, was introduced years afterward [129, 41, 40, 86]. Although FDR evaluation is now the requirement in proteomics, this is not the case in studies of rare mutations in metagenomes. As accurate sequencing technologies enable the identification of thousands of rare mutations within complete MAGs, more than can be easily investigated manually, we believe that applying FDR estimation to this process will become a requirement. We describe an analog of the TDA for evaluating the FDR of identified rare mutations in a metagenome and further extend this idea by developing a *context-dependent* TDA.

We show that deep HiFi-based sequencing can reveal tens of thousands of mutations (with controlled FDR) in a MAG and that these mutations can form hotspots (that may point to selection) and coldspots (that can be used for designing drugs targeting such regions), and we present new methods for phasing these mutations.

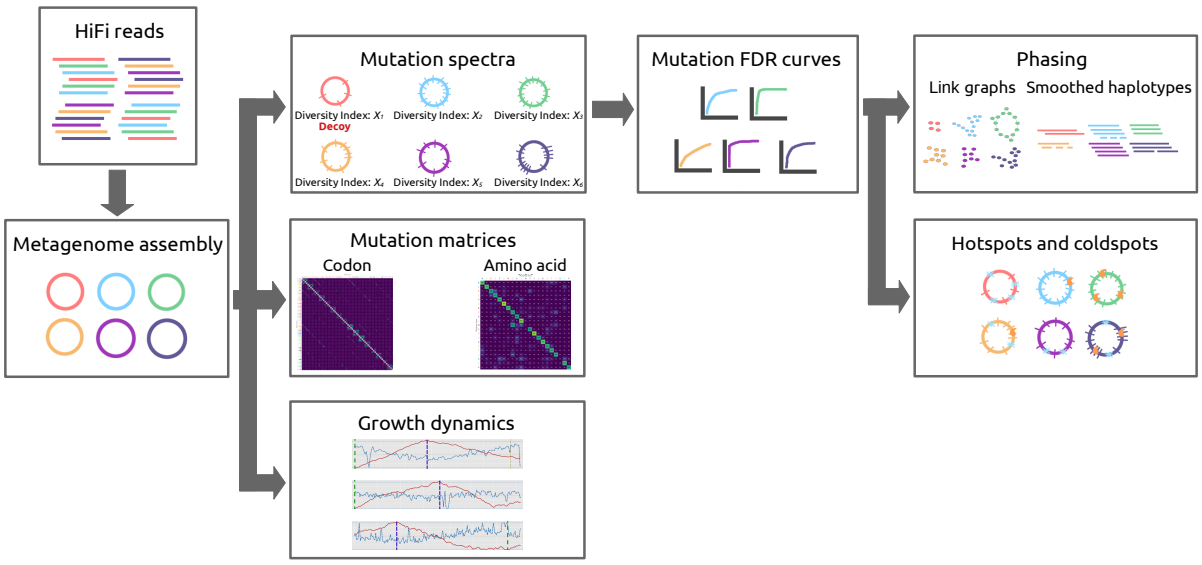


Figure 3.1. strainFlye pipeline. Given a metagenome assembly of HiFi reads, strainFlye identifies rare mutations in each MAG, estimates their FDR, performs phasing analyses using these mutations, identifies hotspots and coldspots of mutations, produces MAG-specific codon and amino acid mutation matrices, and computes information about MAGs' growth dynamics.

3.3 Results

3.3.1 Demonstrating strainFlye

Figure 3.1 illustrates the strainFlye pipeline. The only two required inputs to strainFlye are a HiFi read-set and the contigs (in the case of metaFlye output, edge sequences) assembled from these reads. strainFlye can also optionally take as input an assembly graph indicating contigs' adjacencies for use in adjusting one of the alignment filtering steps (Appendix B.1, "Read alignment").

To benchmark strainFlye, we used a HiFi read-set from a sheep gut metagenome (section 3.5, "Methods") [97, 15]. This read-set is herein referred to as "SheepGut." We selected this data set because it resulted in, as of writing and to our knowledge, the largest number of complete MAGs among all metagenomic data sets analyzed so far [15]. The data set includes 22,118,393 reads with total length 255,708,236 kbp and average length 11.6 kbp.

We assembled the SheepGut data set using metaFlye [97]). The resulting assembly graph includes 78,793 edges (468 of which have lengths of at least 1 Mbp) distributed across 45,988 weakly connected components. Appendix B.2, "Assembly graph" (Figure B.1) provides further details about this assembly graph and how we ran metaFlye.

We classify a contig as *high-coverage* if its coverage is at least *minCov* (default value 1000x) and *long* if its length is at least *minLength* (default value 1 Mbp). We selected three high-coverage and long contigs from the SheepGut metaFlye assembly graph to illustrate various steps of strainFlye's pipeline. These three "selected MAGs" are herein referred to as CAMP, BACT1, and BACT2, as abbreviations of their respective Kaiju [124] taxonomic classifications: *Campylobacter jejuni*, *Bacteria*, and *Bacteroidales*. Appendix B.2, "Assembly graph" (Table B.1; Figure B.2) and Appendix B.3, "Coverages and deletion-rich positions" (Figures B.3; B.4; B.5) provide further information about these MAGs and their coverages.

The bulk of this paper's analyses focus on the SheepGut data set; however, we provide a brief demonstration of strainFlye on another HiFi metagenomic data set (referred to as "ChickenGut") in Appendix B.4, "Demonstrating strainFlye on the ChickenGut dataset".

3.3.2 Computing mutation spectra

We use the term *mutation spectrum* to refer to the collection of mutation frequencies across all positions in a contig. We found that applying LoFreq [213] to the large SheepGut readset was time-consuming (Appendix B.5, "Applying LoFreq to the SheepGut dataset"). Below we show that, in the case of HiFi reads, a simple variant calling method (herein referred to as NaiveFreq) generates similar sets of rare mutations as LoFreq in a fraction of the time. We limit our focus to substitution mutations because the substitution-based error rate in HiFi reads is an order of magnitude smaller than the indel-based error rate [210].

Given an alignment of reads to contigs (Appendix B.1, "Read alignment"), for each position *pos* in a contig, we consider the number of reads spanning this position with a match/mismatch operation in the alignment (spelling one of the four nucleotides A, C, G, or T; we ignore degenerate nucleotides aligned to a position). We define the sum of the numbers of these reads as $reads(pos)$. We define the number of reads of the second-most common nucleotide aligned to *pos* as $alt(pos)$ (breaking ties arbitrarily).

NaiveFreq estimates the mutation frequency of *pos* as $freq(pos) = \frac{alt(pos)}{reads(pos)}$; this value is constrained to the range $[0\%, 50\%]$. Given a frequency threshold percentage $p \in (0\%, 50\%]$, NaiveFreq classifies a position *pos* as a p-mutation if $freq(pos) \geq p$ and if $alt(pos) \geq minAltPos$ (by default, we set $minAltPos = 2$ because, in general, a single read with an alternative nucleotide is an unreliable indicator of a mutation). For the sake of simplicity, we only consider single-allelic mutations (i.e., we do not attempt to call multiple p-mutations at a given position), although the methods proposed in this paper could be extended to account for multiallelic mutations.

NaiveFreq is implemented in strainFlye's pipeline in the `strainFlye call p-mutation` command. On its own, NaiveFreq is not especially useful: p-mutations represent a very primitive way of defining mutations. NaiveFreq's utility comes from the ease with which we can vary p to produce many sets of identified mutations for a contig. As we will show, this property will simplify the process of generating an FDR curve [86] that shows how an increase in the number of identified mutations impacts the FDR estimate associated with these identifications.

3.3.3 The target-decoy approach for estimating the FDR of identified mutations

Whether rare mutations are identified by a state-of-the-art algorithm like LoFreq [213] or through NaiveFreq, it is important to estimate their FDR. In the absence of widely adopted realistic simulation models of errors in HiFi reads, we propose an analog of the

target-decoy approach (TDA) commonly used in areas of bioinformatics without realistic simulation models for data generation [40, 64].

In proteomics, the key idea of the TDA is to use a *decoy* protein database (that has no real matches with observed mass spectra) to evaluate the FDR of spectral matches against a real protein database [40, 86]. We contend that the FDR of identified rare mutations in a given contig can be evaluated similarly, using a contig without any (or with very few) real rare mutations as our decoy.

Estimating the FDR of identified mutations in a contig using the TDA necessitates first attempting to select a *decoy contig* within a metagenome that does not contain any real mutations. We can then apply a mutation identification tool to this decoy contig (containing L positions), assume that all M identified mutations in the decoy contig are false, and compute a mutation rate $rate_{decoy} = \frac{M}{3L}$. The multiplication by three in the denominator accounts for how there are three possible single-nucleotide mutations at each position. Afterward, if analysis of a different ("target") contig using the same mutation identification tool results in a mutation rate of $rate_{target}$ (computed analogously as $\frac{M_{target}}{3L_{target}}$), we can estimate the FDR of the identified mutations in the target contig as $\frac{rate_{decoy}}{rate_{target}}$. (We note that our use of the term "mutation rate," which represents the ratio of the numbers of observed and possible mutations, differs somewhat from the commonly used definition of the mutation rate as the number of mutations per base pair per generation—for example, as used in [9].)

Applying the TDA in this way faces the immediate challenge that "ideal" decoy contigs without any real mutations are unlikely to exist within a metagenome. Therefore, the estimate $\frac{rate_{decoy}}{rate_{target}}$ represents an upper bound for the FDR that, depending on the choice of decoy contig, may greatly inflate the estimated FDR. However, since our analysis revealed great variations (by orders of magnitude) in mutation rates across various contigs within a metagenome, we can simply select the least mutated contig and use it as a (non-ideal but pragmatic) substitute for a decoy (section 3.5, "Methods"). We also modify

the standard TDA to focus our FDR estimation on "rare" mutations with frequencies below a configurable threshold, and thus limit the effects of "indisputable" mutations on the estimation of realistic FDRs (section 3.5, "Methods").

We note that the TDA was initially designed to estimate FDRs in situations without an adequate model for data generation where many unknown factors affect the fidelity of the data. As such, the TDA is well-suited for analyzing reads with poorly understood sources of errors—for example, cross-talk between wells in the case of Illumina reads, or the complex derivation of accurate HiFi reads from error-prone CLR reads. We emphasize that the target-decoy approach results in an empirical FDR estimate, rather than an exact formula for computing the FDR. Many recent papers [64, 90, 91, 42] discuss pros and cons of the TDA, and describe modifications of it that result in even more accurate FDR estimates.

Similarly to how the TDA is used in proteomics (in a way that does not explicitly address complex effects such as internal ions, etc.), our use of the TDA for analyzing rare mutations does not explicitly address complex effects such as well-cross talk, dimness, etc. Instead of attempting to provide some sort of confidence score for individual mutation identifications, the TDA estimates the fraction of erroneously identified mutations "at bulk" within a target contig without trying to investigate their specific sources [64].

3.3.4 Estimating the FDR of identified rare mutations using the TDA

At $p = 0.5\%$, NaiveFreq identifies 249, 17,069, and 1,632 rare ($freq(pos) < 5\%$, as described in section 3.5, "Methods") p-mutations in CAMP, BACT1, and BACT2, respectively. This illustrates that there exists a difference of nearly two orders of magnitude in mutation rates across these MAGs (6.4×10^{-5}, 2.6×10^{-3}, and 1.9×10^{-4} for CAMP, BACT1, and BACT2, respectively). If CAMP, which has a relatively low mutation rate, is selected as a decoy, then the FDR for BACT1 at $p = 0.5\%$ is estimated as $\frac{6.4 \times 10^{-5}}{2.6 \times 10^{-3}} \approx 2.4\%$

(Figure 3.2). For comparison, Appendix B.5, "Applying LoFreq to the SheepGut dataset" demonstrates this estimation process using LoFreq outputs on the three selected MAGs.

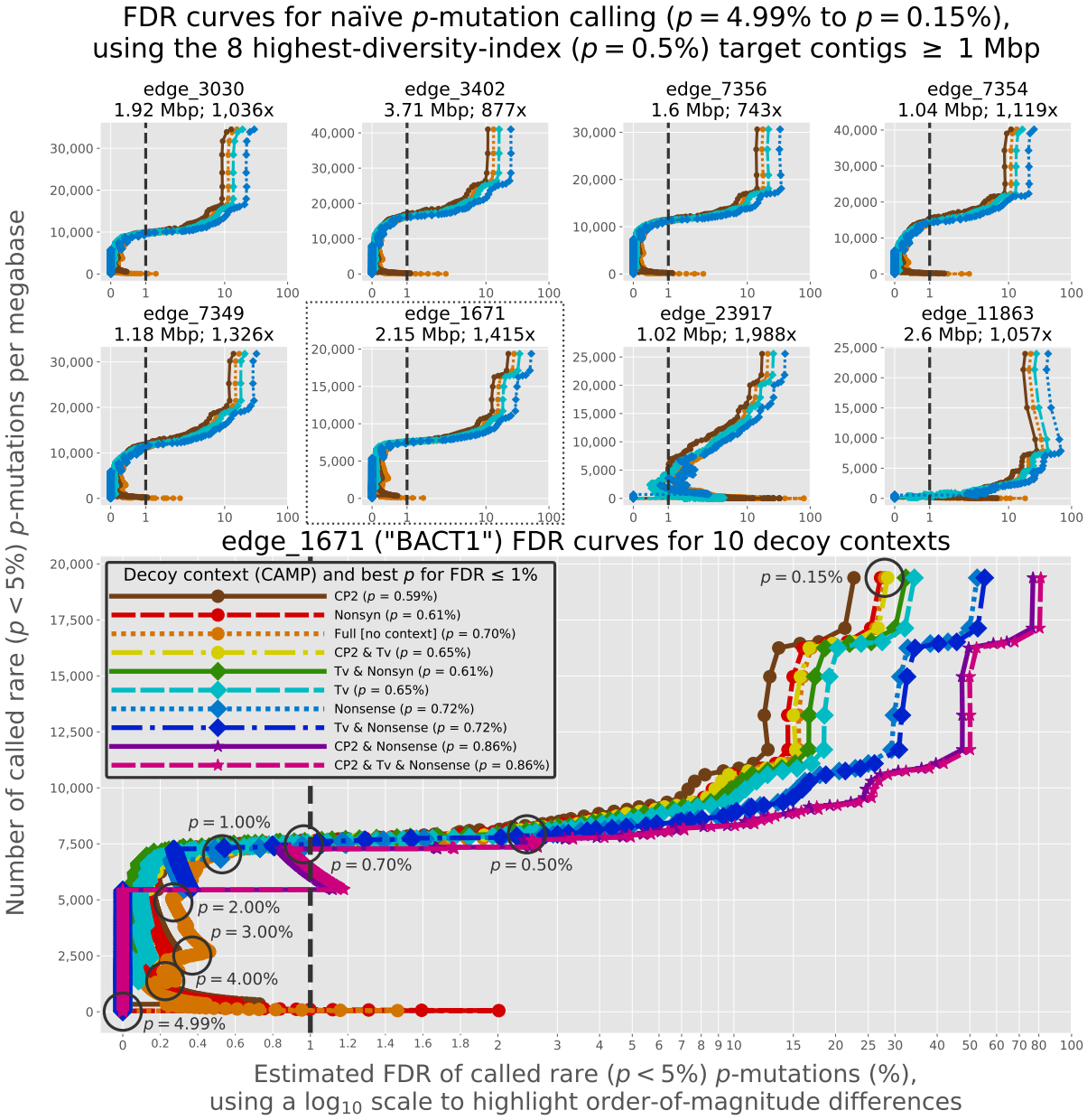
Appendix B.6, "Growth of the number of p-mutations per megabase as p decreases" (Figure B.8) illustrates how the number of p-mutations in the three selected MAGs grows as the frequency threshold p decreases. Each value of p we use to call p-mutations implies a mutation rate for the decoy and target contig alike; Figure 3.2 shows multiple FDR curves [86] for eight high-diversity-index (section 3.3.7, "Diversity indices") target contigs in the SheepGut dataset, computed by adjusting the values of p used. The orange curves in this figure use the entirety of CAMP as a decoy contig; the other curves use *context-dependent* decoy contigs constructed from CAMP, as will be described shortly, to provide different estimates of the FDR. Appendix B.4, "Demonstrating strainFlye on the ChickenGut dataset" (Figure B.6) provides an analogous version of this figure for the ChickenGut dataset.

The `strainFlye fdr estimate` command performs decoy contig selection and FDR estimation; furthermore, given the resulting FDR estimates, the `strainFlye fdr fix` command selects the "optimal" value of p for each target contig and filters each target contig's identified rare mutations to match this threshold value (section 3.5, "Methods").

3.3.5 Context-dependent TDA

The described approach to FDR estimation, although useful, suffers from the fact that—even in relatively-low-mutation-rate contigs like CAMP—many naïvely called mutations represent real rather than false variations. To address this, we describe a *context-dependent target-decoy approach* based on the observation that certain types of mutations are rarer than others. Some positions in a genome are less likely to mutate than other positions, and some specific mutations at a given position are less likely to occur than other mutations. Thus, constructing a decoy contig consisting solely of these relatively mutation-resistant positions and/or mutation types should result in a more

Figure 3.2. FDR curves for eight target contigs in SheepGut. We generate each FDR curve by using NaiveFreq to identify p-mutations in a selected decoy contig (CAMP) as well as the target contig, varying p from 4.99% to 0.15% in increments of 0.01%. **(Top)** Demonstration of four basic decoy contexts, resulting in four FDR curves per target contig. The "Full" context considers all mutations in all positions of the decoy contig; the "CP2" context considers all mutations in only positions located in the second codon position of a single predicted gene in the decoy contig; the "Tv" context only considers transversion mutations in all positions of the decoy contig; and the "Nonsense" context only considers single-nucleotide nonsense mutations in only positions located in a single predicted gene in the decoy contig. **(Bottom)** Demonstration of ten decoy contexts, using BACT1 as the target contig. In addition to the four contexts shown in the above plots, this includes the "Nonsyn" context (corresponding to nonsynonymous mutations) as well as combinations of contexts (Appendix B.8, "Nonsynonymous, nonsense, and transversion decoy contexts"). Fixing the estimated FDR to 1% (indicated by the vertical dashed line shown in all plots) implies a "best" (smallest) value of p for a target contig that allows calling the rarest p-mutations while keeping the estimated FDR $\leq 1\%$: for BACT1, these values are listed in the legend for each decoy context. For clarity, we circle and label certain values of p on the "Full" curve.


FDR curves for naïve p-mutation calling (p = 4.99% to p = 0.15%),
using the 8 highest-diversity-index (p = 0.5%) target contigs ≥ 1 Mbp
edge_3030
1.92 Mbp; 1,036x
edge_3402
3.71 Mbp; 877x
edge_7356
1.6 Mbp; 743x
edge_7354
1.04 Mbp; 1,119x
edge_7349
1.18 Mbp; 1,326x
edge_1671
2.15 Mbp; 1,415x
edge_23917
1.02 Mbp; 1,988x
edge_11863
2.6 Mbp; 1,057x
edge_1671 ("BACT1") FDR curves for 10 decoy contexts
Decoy context (CAMP) and best p for FDR ≤ 1%
CP2 (p = 0.59%)
Nonsyn (p = 0.61%)
Full [no context] (p = 0.70%)
CP2 & Tv (p = 0.65%)
Tv & Nonsyn (p = 0.61%)
Tv (p = 0.65%)
Nonsense (p = 0.72%)
Tv & Nonsense (p = 0.72%)
CP2 & Nonsense (p = 0.86%)
CP2 & Tv & Nonsense (p = 0.86%)
p = 0.15%
p = 1.00%
p = 0.70%
p = 0.50%
p = 2.00%
p = 3.00%
p = 4.00%
p = 4.99%
Number of called rare (p < 5%) p-mutations per megabase
Estimated FDR of called rare (p < 5%) p-mutations (%),
using a log10 scale to highlight order-of-magnitude differences

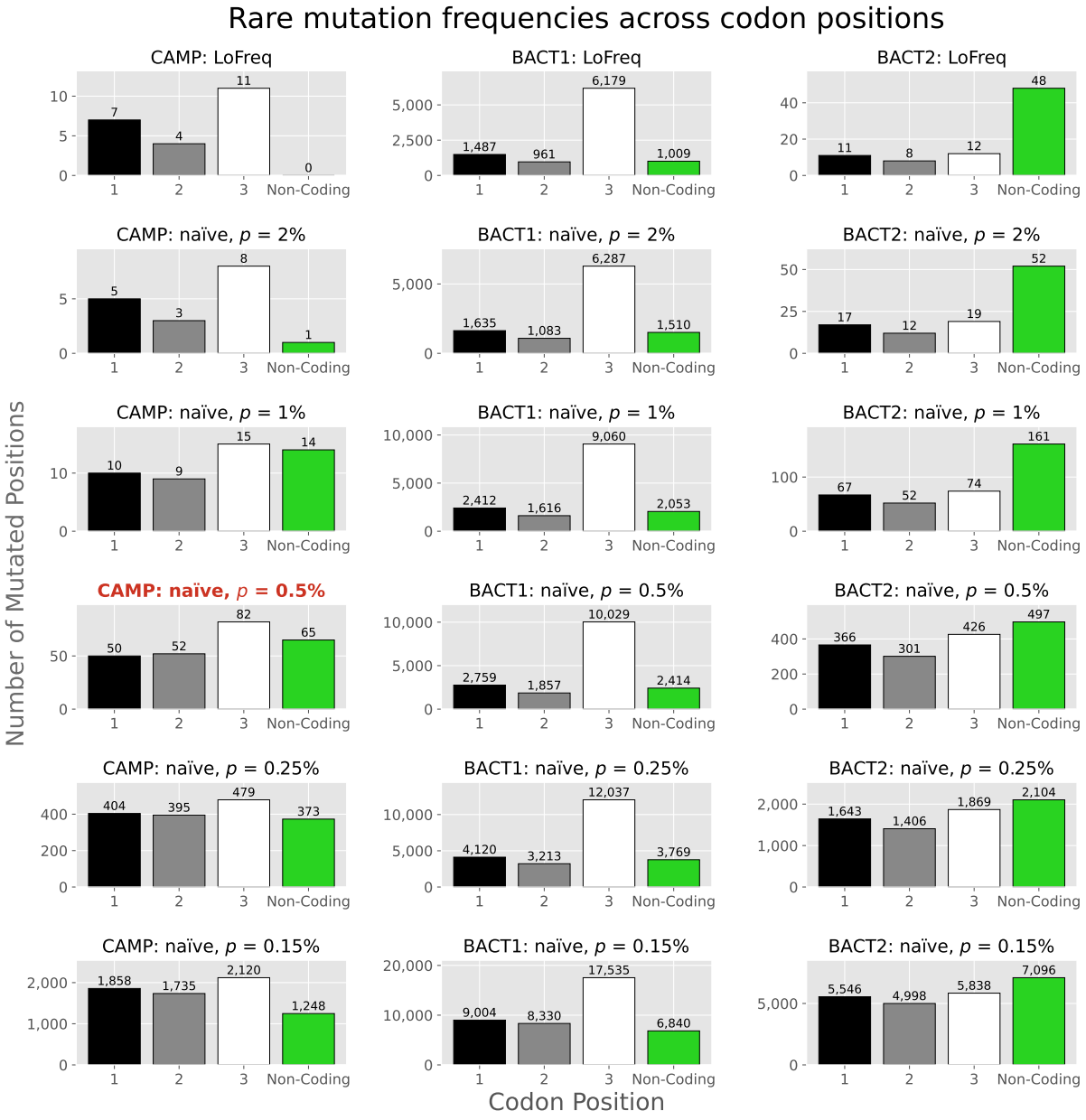
accurate FDR estimate.

Codon positions (CPs) are a simple example of this. Mutation rates vary sharply across the first, second, and third CPs (CP1, CP2, and CP3) within protein-coding regions of genomes [16]: In general, mutations in CP3 are less likely to change the amino acid encoded by a codon than mutations in CP2 or, to a lesser extent, mutations in CP1. Given predicted protein-coding genes in a data set's contigs (section 3.5, "Methods"), we define four "groups" of positions for each contig: first/second/third positions of codons (CP1/CP2/CP3) and noncoding positions (positions located outside of predicted genes). We ignore positions that are located within multiple predicted genes owing to gene overlap. If we define M_P as the number of mutated positions identified within a group of positions P, we would expect $M_{\text{CP2}} < M_{\text{CP1}} < M_{\text{CP3}}$ [16]. Figure 3.3 and Figure B.9 show that, on the three selected MAGs, this property holds for the mutations called by NaiveFreq at many values of p as well as for the mutations called by LoFreq. Appendix B.7, "Codon position analysis details" presents additional information about the details of this analysis.

This result suggests that computing the decoy contig mutation rate as $\frac{M_{\text{CP2}}}{3L_{\text{CP2}}}$ may yield a more realistic FDR estimate. We refer to this modification as a *decoy context* that we apply to the original decoy contig; Figure 3.2 shows the use of this "CP2" decoy context for the CAMP decoy contig, showing that it generally lowers FDR estimates compared with using all of CAMP as a decoy contig.

Besides CPs, there exist other candidate events that show a sharp contrast between various mutation types and thus present promising options for the construction of decoy contigs. Whether owing to selection or other vagaries of the processes by which mutations occur, we expect nonsynonymous, nonsense, or transversion mutations to be rarer, respectively, than synonymous, non-nonsense, or transition mutations [180, 204]. Appendix B.8, "Nonsynonymous, nonsense, and transversion decoy contexts" confirms this (Figure B.10) and describes how strainFlye constructs decoy contexts using these "rarer" types of mutations; Figure 3.2 includes additional FDR curves produced using decoy

Figure 3.3. Rare mutation frequencies vary across codon positions in the three selected MAGs. This plot includes variant calls from LoFreq **(top row)** and results from NaiveFreq for $p \in \{2\%, 1\%, 0.5\%, 0.25\%, 0.15\%\}$ **(bottom rows)**. The expected "pattern" of codon position mutation frequencies ($M_{\text{CP2}} < M_{\text{CP1}} < M_{\text{CP3}}$) holds for all plots except for the case of CAMP at $p = 0.5\%$ (the title for this plot is highlighted in red). As discussed in Appendix B.5, "Applying LoFreq to the SheepGut dataset" (Figure B.7), LoFreq's results are similar to NaiveFreq's at $p = 2\%$. To focus this visualization on "rare" mutations, we limit the mutations included in all rows (including LoFreq's) to those located in positions at which $freq(pos) < 5\%$ (section 3.5, "Methods"). We also treat positions where LoFreq called multiple mutations as if only a single mutation were called at this position (Appendix B.5, "Applying LoFreq to the SheepGut dataset"). Appendix B.7, "Codon position analysis details" (Figure B.9) shows an alternate version of this figure, in which y-axis values are normalized by the total number of positions considered in order to make plots from different MAGs more easily comparable.


Rare mutation frequencies across codon positions
CAMP: LoFreq
BACT1: LoFreq
BACT2: LoFreq
CAMP: naïve, p = 2%
BACT1: naïve, p = 2%
BACT2: naïve, p = 2%
CAMP: naïve, p = 1%
BACT1: naïve, p = 1%
BACT2: naïve, p = 1%
CAMP: naïve, p = 0.5%
BACT1: naïve, p = 0.5%
BACT2: naïve, p = 0.5%
CAMP: naïve, p = 0.25%
BACT1: naïve, p = 0.25%
BACT2: naïve, p = 0.25%
CAMP: naïve, p = 0.15%
BACT1: naïve, p = 0.15%
BACT2: naïve, p = 0.15%
Number of Mutated Positions
Codon Position
1
2
3
Non-Coding

contexts of possible nonsynonymous, nonsense, and transversion mutations, as well as various combinations of these contexts and the aforementioned CP2 context.

These analyses focus on p-mutations, in which mutation frequencies are used to call a position as mutated or not. Frequencies are useful for this task because they enable the (imperfect) comparison of positions or contigs with different coverages; Appendix B.9, "Identifying mutations based solely on read counts" discusses an alternative method for calling mutations based solely on read counts.

3.3.6 Codon and amino acid mutation matrices

Proteins are usually compared using the PAM [35] or BLOSUM [72] matrices, based on frequencies of amino acid substitutions over large evolutionary distances (millions of years). It remains unclear whether these matrices are well suited for comparing proteins encoded by strains that are separated by short evolutionary distances. The complete metagenomics approach [15] provides the potential to derive MAG-based analogs of these matrices for short evolutionary distances. Moreover, it allows the extension of amino acid substitution matrices into more informative codon substitution matrices. To illustrate the potential of this approach, Appendix B.10, "Constructing and visualizing mutation matrices" describes the construction of these matrices in detail and shows visualizations of them (Figures B.11; B.12; B.13) for the three selected MAGs.

3.3.7 Diversity indices

Below we describe the *diversity index* that quantifies the widely varying mutation rates across different species within a metagenome. Diversity indices are useful as a way to select a decoy contig for use with the TDA (section 3.5, "Methods"), and the `strainFlye call p-mutation` command outputs diversity indices for this reason.

We compute diversity indices relative to a given p threshold for naïvely calling p-mutations. We define a position as *sufficiently covered* if its coverage is at least

$minSuffCov$, a threshold that is computed as a function of p (such that lower values of p generally correspond to a larger $minSuffCov$). We then define the diversity index of a contig G as the number of mutations called in the sufficiently covered positions in G, divided by the total number of sufficiently covered positions in G (given the selected p threshold value). Appendix B.11, "Diversity index details" provides details on these definitions and their motivations.

We note that computing the diversity index of a complete bacterial genome in the context of short-read metagenomics is problematic, because studies using short reads alone struggle to assemble complete genomes; however, the emergence of complete metagenomics [15] has opened the possibility of reconstructing many complete MAGs from a microbial sample and thus analyzing their diversity indices. Figure 3.4 shows great variation in the diversity indices of the MAGs in the SheepGut data set.

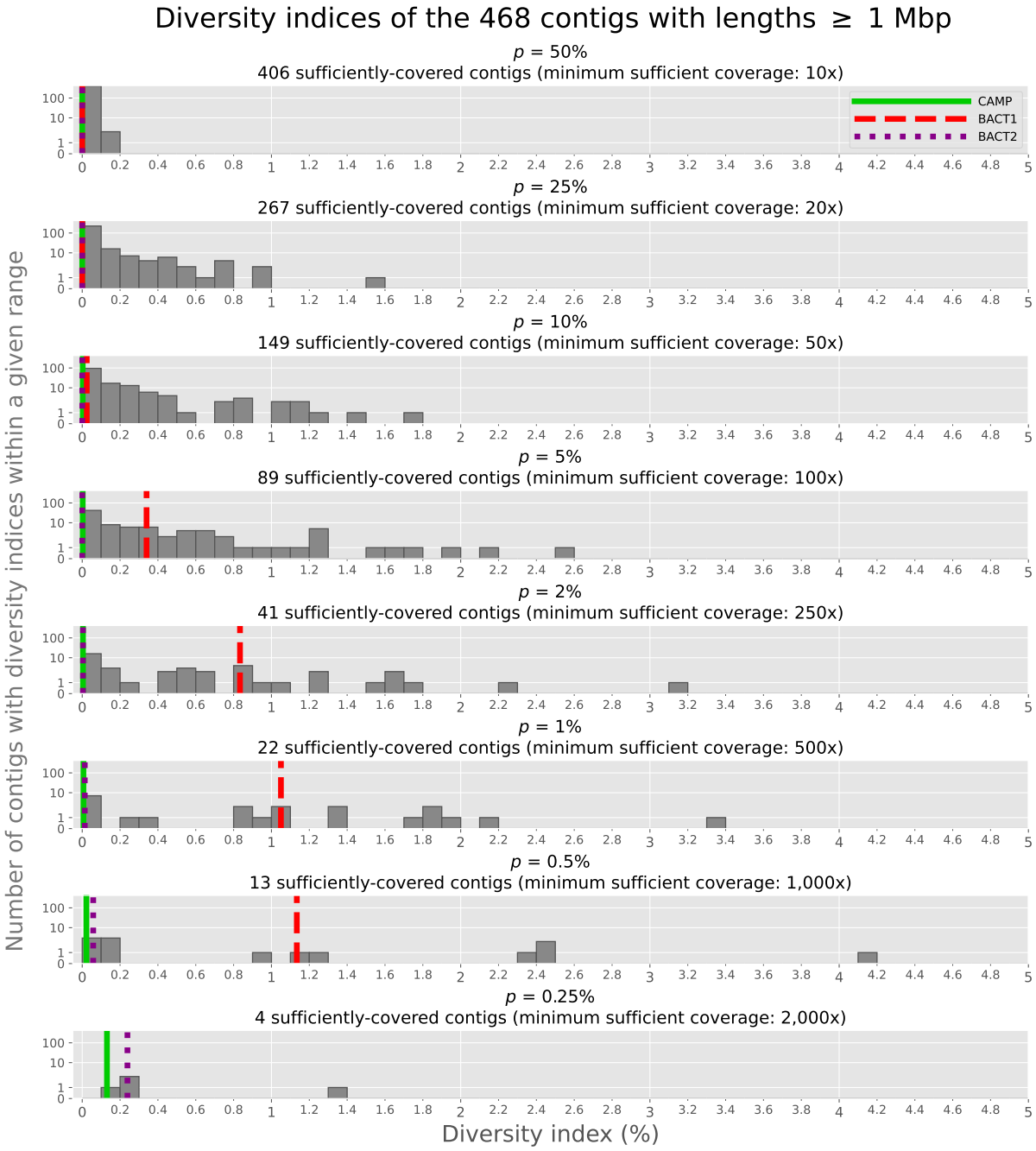
3.3.8 Genomic locations of mutations

Given a set of identified mutations, strainFlye supports the identification of basic hotspots and coldspots [213, 175].

We define the *mutation rate* of a feature (an arbitrary region in a genome, e.g., a predicted gene) as the fraction of mutated positions in this region. The `strainFlye spot hot-features` command takes as input a list of genomic features in contigs and identifies "hotspot" features given simple user-specified thresholds (the minimum number of mutations a feature must have to be considered a hotspot and/or the minimum mutation rate a feature must have to be considered a hotspot). Appendix B.12, "Hotspot genes in the three selected MAGs" (Table B.2) provides information about various hotspot genes for each of the three selected MAGs.

Figure 3.5 shows the mutation spectra of the highest-mutation-rate genes of the three selected MAGs (using NaiveFreq results at $p = 0.5\%$). The genes shown in Figure 3.5 (top, bottom) show essentially binary splits between their positions' mutation frequencies.

Figure 3.4. Diversity indices vary widely across the 468 long contigs in SheepGut. Smaller values of p allow us to "call" increasing amounts of p-mutations in a MAG, at the cost of requiring higher sequencing coverage to reliably distinguish these mutations from errors (we note that this differs from ordinary usage of NaiveFreq, which does not explicitly take coverage into account). Appendix B.11, "Diversity index details" provides details about some of the high-diversity-index contigs shown in this figure and about how minimum sufficient coverages are determined for each value of p (we use $minReadNumber = 5$ here). The diversity index values for the three selected MAGs are highlighted on each row of the histogram as vertical lines. Although it is shown in Figure 3.3, we have omitted $p = 0.15\%$ from this figure because its minimum sufficient coverage is 3333.33x; the only MAG that is sufficiently covered for this value of p is CAMP.


Diversity indices of the 468 contigs with lengths ≥ 1 Mbp
p = 50%
406 sufficiently-covered contigs (minimum sufficient coverage: 10x)
p = 25%
267 sufficiently-covered contigs (minimum sufficient coverage: 20x)
p = 10%
149 sufficiently-covered contigs (minimum sufficient coverage: 50x)
p = 5%
89 sufficiently-covered contigs (minimum sufficient coverage: 100x)
p = 2%
41 sufficiently-covered contigs (minimum sufficient coverage: 250x)
p = 1%
22 sufficiently-covered contigs (minimum sufficient coverage: 500x)
p = 0.5%
13 sufficiently-covered contigs (minimum sufficient coverage: 1,000x)
p = 0.25%
4 sufficiently-covered contigs (minimum sufficient coverage: 2,000x)
CAMP
BACT1
BACT2
Number of contigs with diversity indices within a given range
Diversity index (%)

However, the gene shown in Figure 3.5 (middle) offers a less clear interpretation: There exist three main groups of positions with high mutation frequencies (with frequencies $\sim$ 1%–2%, 4%–5%, and 6%–7%) covering this gene, as well as a group of mostly unmutated positions. Appendix B.13, "Identifying strains in the most mutated gene of BACT1" (Figure B.15) uses long reads to inspect the structure of mutations within this gene and analyze whether they are indeed associated with four different strains.

Appendix B.14, "Plots of mutation locations" (Figure B.16) visualizes the locations of mutations in CAMP, BACT1, and BACT2; this visualization shows that there exist clear "coldspot" gaps between mutations, noticeably in BACT1 (Table B.3). The `strainFlye spot cold-gaps` command can identify these gaps and compute the probability of the largest gap in a contig being of at least a certain length (Appendix B.15, "Investigating coldspots").

3.3.9 Growth dynamics of a metagenome

Since short-read sequencing rarely results in the assembly of complete MAGs, methods for analyzing microbes' growth dynamics using short-read sequencing data [99, 85] typically rely on reference databases. By improving initial MAG completeness, deep HiFi sequencing provides us the ability to analyze MAGs' growth dynamics completely *de novo*; Appendix B.16, "Growth dynamics" (Figure B.17) illustrates that this analysis represents yet another benefit of "complete metagenomics."

3.3.10 Phasing identified mutations

The ability to call many mutations with controlled FDR in a contig is essential for phasing the strain-level haplotypes represented in this contig, because haplotype phasing relies on the presence of reads that span multiple mutations [138]. Fortunately, long and accurate HiFi reads simplify the process of phasing detected mutations and thus performing strain separation.

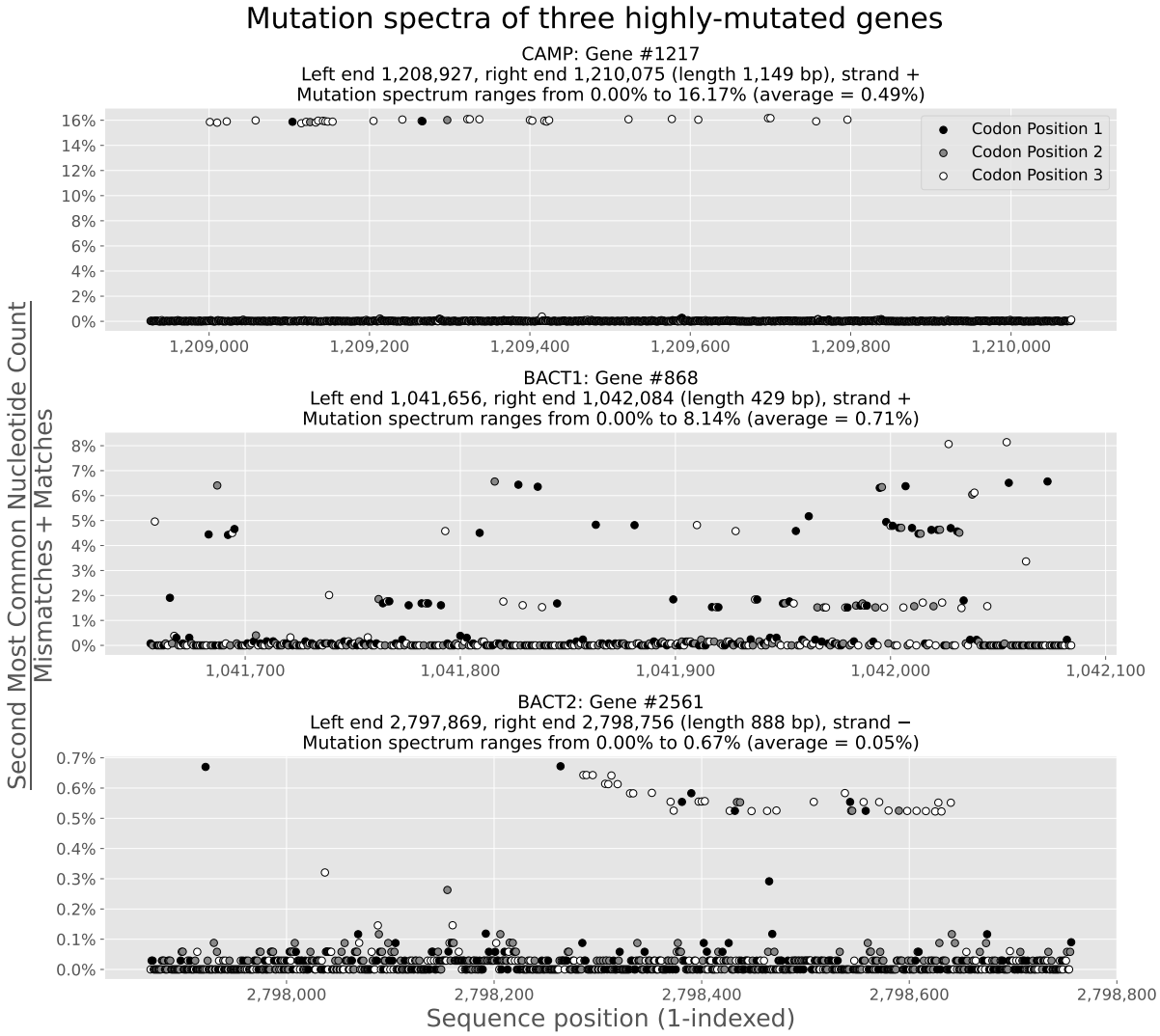


Figure 3.5. Mutation spectra of highly mutated genes in the three selected MAGs. Each plot shows $freq(pos)$ for every position pos within the selected gene; each of these genes has the highest mutation rate in its parent MAG (for NaiveFreq mutation calls at $p = 0.5\%$) and is thus described in the corresponding MAG's first row in B.2. We exclude deletions from the denominator of $freq(pos)$ (similar to most of the other analyses in this paper), which is responsible for some of the "jumps" in the **middle** plot (gene 868 in BACT1). Positions are colored by their codon position within the gene, using colors matching those in Figure 3.3. These figures are analogous to other plots of variation along a gene sequence in the literature, for example Figure 1 in [198]. Figure B.14 in Appendix B.12, "Hotspot genes in the three selected MAGs" visualizes the coverages of each of these genes.

To reveal the conserved and diverged regions in various strains of a contig, the `strainFlye smooth` module attempts to construct the de Bruijn graph [30] of these strains, not unlike the de Bruijn graphs that are used for analyzing variations among various human genomes [81]. Because errors in reads make this graph very complex, the first command in this module, `strainFlye smooth create`, converts reads aligned to a given contig to "smoothed reads," which match the assembled contig at all positions except for those positions in the read aligned to identified mutations (section 3.5, "Methods"). In addition to smoothed reads, `strainFlye smooth create` also constructs "virtual reads" to fill in low-coverage regions in contigs and maintain contiguity (section 3.5, "Methods"). The next command, `strainFlye smooth assemble`, uses LJA [8] to construct the multiplex de Bruijn graph of the collection of smoothed and virtual reads for each contig (section 3.5, "Methods").

Figure 3.6 and Figure B.21 show visualizations of the multiplex de Bruijn graphs produced by LJA given the smoothed and virtual reads produced for each of the three selected MAGs. Ideally, these smoothed reads would have been assembled perfectly, such that one isolated edge would exist for each unique haplotype. The graphs shown here do not reach this ideal, but they come close: Many regions of these MAGs are now represented as linear sequences of "bubbles" representing the variation between haplotypes at a given position in the MAG.

In addition to the `smooth` module, strainFlye also provides the `link` module for strain analysis. This module constructs a *link graph* representing the alleles at each mutated position in a contig, and the frequencies with which these alleles are linked by reads. These graphs, described in Appendix B.17, "The link graph structure for haplotype visualization" (Figures B.18; B.19), can be helpful as a visualization tool of what regions of a contig can be phased from the available data.

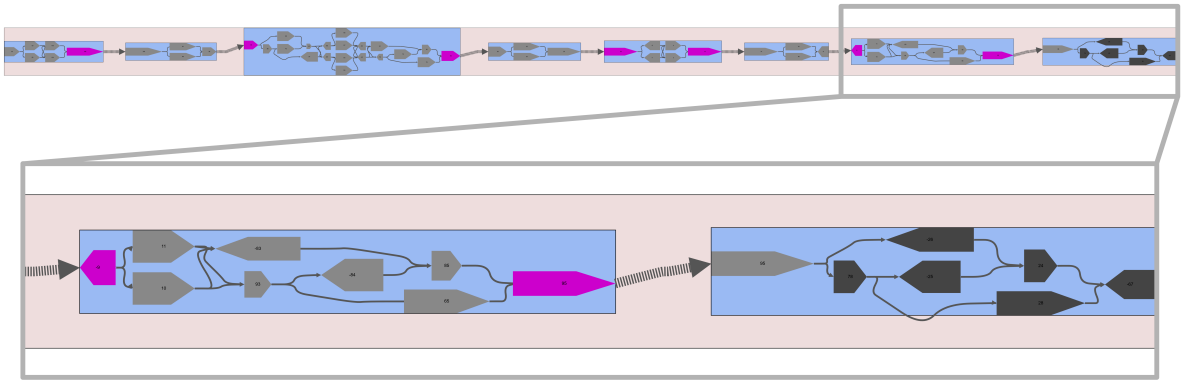


Figure 3.6. Multiplex de Bruijn graph produced by LJA for CAMP's smoothed and virtual reads. We generated smoothed reads based on mutations identified by NaiveFreq at $p = 1\%$. This is a MetagenomeScope visualization [46] of the GFA file produced by LJA. Gray pentagons (nodes in this visualization) correspond to "segments" described in this GFA file, which in turn correspond to edges in the de Bruijn graph. Blue regions of the graph indicate "bubble" patterns MetagenomeScope identified in the graph [125], highlighted for clarity. Segments colored in pink represent segments that are shared between multiple bubbles: MetagenomeScope duplicates segments (creating a link between the two copies of a duplicate segment) in order to simplify the visualization of adjacent bubbles in the graph. This visualization makes clear that the entire de Bruijn graph can be represented as a linear sequence of bubbles. This topology is consistent with the expected structure of an assembly graph of multiple strains of a genome, for example, as shown in Figure 4 of [97]; the branching paths in these bubbles likely represent strain-level diversity. The **rightmost** two bubbles in the graph are shown up in a box **below** the main drawing of the graph. The six dark-colored segments highlighted in the **rightmost** bubble correspond to the segments that overlap gene 1217, the most mutated gene in CAMP (Figure 3.5, top). There exist three paths through these segments' bubbles: the **top** two paths correspond to the "reference" haplotype of gene 1217 and the **bottom** path corresponds to the "alternate" haplotype of gene 1217 (Appendix B.18, "Haplotypes of the most mutated gene in CAMP"; Figure B.20). The reason for the reference haplotype being represented by two distinct paths is that these paths cover other mutated positions located earlier in CAMP.

3.4 Discussion

We have presented strainFlye, a pipeline for the identification and analysis of rare mutations within MAGs sequenced using HiFi reads. We showed variations in the frequencies of mutations between CPs and mutation types within MAGs, variations in the numbers of mutations across MAGs in a data set, the identification of hotspots and coldspots of mutations, and the ability of HiFi reads to link mutated positions throughout large portions of MAGs.

These advances set the stage for further improvements in identifying mutations and phasing using complementary linked read technologies, such as reads generated using Hi-C [20] or TELL-seq technology [26]. Complementary short-read sequencing technologies, as well, hold promise for further improving our confidence in the identification of rare mutations [15, 119].

We note that the methodologies developed here could have uses not only in isolation but also as extensions to existing tools for variant calling and metagenomic data analysis. Recent work on optimizing LoFreq in the context of high-coverage viral data sets [92] has highlighted the potential for the community to continue to refine existing methods to handle the ever-increasing challenges associated with new types of data.

Although HiFi reads simplify many tasks in metagenomic data analysis, there remain countless open problems. One such problem is the study of lethal positions: A bacterial gene is classified as *lethal* (*nonlethal*) if its deletion allows (does not allow) an organism to survive under some defined conditions [214]. Although identification of all lethal SNVs in a bacterial genome provides much more information than identification of all *lethal genes* [215], finding these mutations remains an open problem. Moreover, even the simpler problem of identifying a large set of nonlethal mutations in a bacterial genome remains an open problem. Information about such sets is valuable because it can inform various studies, for example, analyses of positions in a genome that lead to antibiotic

resistance [105].

We have shown the possibility of revealing tens of thousands of putatively nonlethal mutations in multiple MAGs in SheepGut (Figure 3.2) using deep metagenomic sequencing. Even though this example represents—for any of these MAGs alone, to our knowledge—the largest collection of rare mutations in a bacterial genome reported to date, the true extent of diversity in a bacterial community (when the frequency threshold is reduced and the coverage increases by an order of magnitude) remains unknown. Future "ultradeep" sequencing projects (e.g., with coverages as large as 100,000x) may shed light on this question and narrow the set of lethal mutations in bacterial genomes.

3.5 Methods

3.5.1 Automatically selecting a decoy contig

In the SheepGut data set, CAMP serves as a long high-coverage decoy contig with relatively few rare mutations (Figures 3.3; 3.4) However, there is no guarantee that such a contig will exist in an arbitrary data set. The `strainFlye fdr estimate` command allows the user to automatically select (or to specify) a decoy contig.

To automatically select a decoy contig, strainFlye first identifies the set of all long high-coverage contigs (section 3.3.1, "Demonstrating strainFlye"). If this set is empty, strainFlye will raise an error explaining the situation to the user; this situation implies that the user will need to lower the *minLength* and *minCov* thresholds in order to find a decoy contig. If this set consists of a single contig, strainFlye will select it as the decoy.

If there are n long high-coverage contigs, strainFlye makes use of diversity index information computed for a set D of various values of the frequency threshold p (section 3.3.7, "Diversity indices"). Each entry in D implies an n-dimensional vector of diversity indices computed for this threshold value (with one entry for each long high-coverage contig). Because the diversity index can be undefined for low-coverage contigs, depending

on the minimum sufficient coverage for the corresponding value of p (Figure 3.4; Appendix B.11, "Diversity index details"), strainFlye focuses only on the $D_{good} \subseteq D$ threshold values for which at least two long high-coverage contigs have defined diversity indices. If $|D_{good}| = 0$, that is, no vector of diversity indices has at least two long high-coverage contigs with defined diversity indices, strainFlye raises an error explaining the situation to the user.

If $|D_{good}| \geq 1$, then, for each of the corresponding "good" vectors of diversity indices, strainFlye finds the minimum and maximum diversity index across the long high-coverage contigs. It then assigns each of these contigs a score in the range $[0, 1]$ using linear interpolation: The contig with the lowest diversity index in this vector is assigned a score of zero, the contig with the highest diversity index in this vector is assigned a score of one, and all other contigs with defined diversity indices are scaled in between. Contigs in this vector with an undefined diversity index are assigned a score of one in order to penalize them for not being sufficiently covered.

Finally, strainFlye computes a total score for each long high-coverage contig by summing its scores for each of the $|D_{good}|$ diversity index vectors. The contig with the lowest total score is selected as the decoy contig, breaking ties arbitrarily.

We note that there is no guarantee that any of the long high-coverage contigs will have a "low" amount of rare mutations; it may be the case that all of these contigs have high diversity indices. In this case, the automatic selection process will select a decoy contig with a relatively high mutation rate, and strainFlye will thus produce inflated FDR estimates. This is an inherent downside of the TDA; however, we expect that the rising sizes of sequencing data sets will increase the likelihood of suitable decoy contigs existing.

3.5.2 Accounting for indisputable mutations

We classify a p-mutation as *indisputable* if its mutation rate $freq(pos)$ (defined in the section 3.3.2, "Computing mutation spectra") is greater than or equal to a specified

highFrequency threshold (default value 5%) and as a *rare* mutation otherwise. Although the reliable identification of rare mutations (with frequencies below the error rate in reads) is a challenging task, nearly all indisputable mutations are real, especially in high-coverage MAGs like those studied in this paper. When computing the mutation rates of a decoy or target contig G (with M_G mutations and length L_G), as discussed in the section 3.3.3, "The target-decoy approach for estimating the FDR of identified mutations", we thus redefine $rate_G = \frac{M_{rare}}{3L_G}$, where M_{rare} is the number of rare mutations in the contig. This modification can lower the computed mutation rates for a given contig and thus result in more realistic FDR estimates.

3.5.3 Fixing the estimated FDR of identified rare mutations in a target contig

We have described how to use the TDA to estimate the FDR of a set of identified rare mutations in a contig and to draw an FDR curve for this contig. Given a user-configurable upper bound f on the estimated FDR of rare mutation identifications (e.g., $f = 1\%$, as shown in Figure 3.2), the `strainFlye fdr fix` command can fix the estimated FDR for a target contig to be $\leq f$ as follows.

First, we select the lowest value of p at which the estimated FDR for this contig is $\leq f$. Because our FDR estimates do not necessarily increase monotonically as p decreases (Figure 3.2), the vertical line implied by fixing the FDR to a given upper bound may be crossed by a target contig's FDR curve multiple times [86]. However, the number of mutations per megabase produced by a given p value in the target contig does increase monotonically as p decreases. Because of this, the lowest threshold value yielding an acceptable estimated FDR corresponds to the largest set of rare mutations for this target contig with an acceptable estimated FDR.

Because we have assumed that all "indisputable" mutations (section 3.5, "Methods") are correct, strainFlye outputs—for each target contig—the set of all rare mutations

supported by the "optimal" value of p selected, as well as the set of all indisputable mutations. strainFlye also outputs the set of all indisputable mutations within the decoy contig, although it does not output any rare mutations from the decoy contig (as of writing, this is performed regardless of the decoy context used). These mutations can be used as the starting point for downstream analyses (e.g., phasing).

We note that, for p-mutations, the level of granularity used in varying p can have an impact on the selection of this "optimal" value. strainFlye's pipeline adjusts p in increments of 0.01% (Figure 3.2; B.8), which we expect should be sufficient for most current data sets.

3.5.4 Predicting protein-coding genes in contigs

Many of the described "decoy contexts" in section 3.3.5, "Context-dependent TDA", as well as many of the other shown analyses, rely on the availability of gene predictions: We compute these using Prodigal [80]. Notably, we use Prodigal's `-c` option in order to disallow the prediction of incomplete genes that run off the end of a contig; this restriction simplifies these analyses. For the three selected SheepGut MAGs shown in this paper, we ran Prodigal in its "normal" or "single" mode (processing one contig at a time): This matches the behavior of `strainFlye fdr estimate` when the decoy context(s) requested by a user requires gene predictions in the decoy contig.

We note that our use of Prodigal makes the implicit assumption that the contigs on which we run it are prokaryotic. It should be feasible to extend our analyses to work with alternative gene prediction tools if desired, although for now we have focused our efforts on analyzing prokaryotic contigs.

3.5.5 Constructing smoothed reads

Given a set of identified mutations in a contig, and a read aligned to this contig, we identify all mutated positions in the contig to which a linear alignment of this read has

(mis)match operations. We then convert each linear alignment of this read to the contig into a "smoothed read" that completely matches the contig, with the exception of the read's nucleotides at all identified mutated positions spanned by the alignment.

This process involves some simplifying assumptions: For one, it necessarily ignores both indels and non-"identified" single-nucleotide mutations. If a read's nucleotide at any of the identified mutated positions to which it has a (mis)match operation does not match the "reference" or "alternate" nucleotide at this position (in the case of mutations identified by NaiveFreq, these two will always correspond to the first- and second-most common nucleotide at a position), then we discard this alignment of this read and do not generate a smoothed read from it. This operation implicitly ignores the contig's "reference" nucleotide at this mutated position, so there is the (unlikely) possibility for the reference nucleotide at an "unreasonable" position (where the reference and consensus nucleotide disagree) (for discussion, see Appendix B.12, "Hotspot genes in the three selected MAGs") to be completely unrepresented in the smoothed reads if it is not the "reference" or "alternate" nucleotide at a mutated position. Similarly, if a read spans a mutated position with a deletion, we discard this alignment of this read.

We note that these "discarding" steps are only applied to individual linear alignments of a read at a time; for example, a read with two linear alignments (one "representative," one supplementary) (https://samtools.github.io/hts-specs/SAMv1.pdf) to a contig could result in the creation of zero, one, or two smoothed reads for this contig. The input alignment file should not contain any secondary alignments or overlapping supplementary alignments on a contig (Appendix B.1, "Read alignment") so, even if a single read is converted to multiple smoothed reads in a contig, these smoothed reads should all cover disjoint regions within the contig.

This can lead to the generation of smaller haplotype assembly graphs than may be expected, because we effectively ignore haplotypes that disagree with any of the identified mutations in a contig. This is a likely factor in why the BACT1 assembly graph (shown

in Figure B.21 in Appendix B.19, "Smoothed haplotype assembly graphs") is simple compared with the CAMP and BACT2 assembly graphs: Similar to how BACT1 has an order of magnitude more $p = 1\%$ mutations (22,415) than CAMP (83) and BACT2 (380), BACT1 has an order of magnitude more $p = 1\%$ mutations with at least one deletion in their pileup (8269 / 22,415) compared with CAMP (57 / 83) and BACT2 (274 / 380). This results in a comparatively large number of reads being discarded when attempting to generate smoothed haplotypes for BACT1.

Finally, we note that this process also splits supplementary alignments of a single read into distinct smoothed reads because it is unclear how to produce a single smoothed read from a read that has multiple alignments to a contig. However, because of the filtering of overlapping supplemental alignments described in Appendix B.1, "Read alignment" these distinct smoothed reads should not overlap with each other on a contig.

3.5.6 Constructing virtual reads

Assembly and/or alignment artifacts can lead to certain positions in a MAG having relatively low coverage. These coverage drops—for example, the drops shown in Appendix B.3, "Coverages and deletion-rich positions"—complicate the assembly of haplotypes of these nonetheless already-assembled MAGs. Positions without any coverage at all will split the resulting de Bruijn graph, and positions with low coverage can still result in discontiguous assemblies owing to assemblers treating these regions' corresponding smoothed reads as erroneous. To address this problem, we create virtual reads that can span uncovered or low-coverage positions in a MAG. We note that the term "virtual reads" is used in a similar context in [8].

Consider a MAG with average coverage C_m, defining the coverage at each position only based on the number of matching and mismatching reads in the input alignment (ignoring deletions). We define a position with coverage (based only on smoothed reads) C_p in this MAG as *low-coverage* if $C_p < minWellCovFrac \cdot C_m$, where $minWellCovFrac$

is a percentage in the range $[0\%, 100\%]$ (default value 95%). We chose this default value based on testing LJA with the smoothed and virtual reads constructed from CAMP until the resulting assembly became contiguous, as is shown in Figure 3.6. To construct virtual reads, we identify all "runs" of consecutive low-coverage positions in the MAG. For each of these runs, we define its average coverage as C_r. We then create a virtual read that matches the MAG "reference" sequence at all positions throughout this run, as well as $vrFlank$ (default value 100) positions before and after the start and end of the run (clamping to the start or end of the MAG as needed, in case the run is close to a boundary of the MAG). Finally, we generate $round(C_m - C_r)$ copies of this virtual read in order to "lift" the coverage throughout this run of low-coverage positions to roughly match C_m.

Like the construction of smoothed reads, the construction of virtual reads also involves making some concessions; for example, in the case in which an identified mutation occurs within a virtual read or its flanking $vrFlank$ positions, we simply set the virtual read's nucleotide at this mutated position equal to the MAG reference. Or, in strange circumstances, the flanking positions in the virtual reads created for a low-coverage region might overlap the virtual reads generated for other nearby low-coverage regions. These artifacts may cause undesirable effects in the haplotype assembly process. We have nonetheless found that using virtual reads, in addition to just smoothed reads, helps improve the contiguity of the multiplex de Bruijn graphs produced by LJA.

Lastly, we note that we do not construct virtual reads that connect the end and start of a MAG, even if this MAG is represented in the original assembly graph as a single circular contig (e.g., as BACT1 and BACT2 are). Although this can have the effect of linearizing circular contigs, we expect that this should not make a large difference in downstream analyses using the resulting smoothed haplotypes.

3.5.7 Assembling smoothed and virtual reads

After constructing smoothed and virtual reads, the `strainFlye smooth assemble` command provides these reads as input to LJA, an assembler designed for HiFi reads [8]. We run LJA without its error correction (mowerDBG) step; instead, we apply a simple k-mer coverage filter that removes very-low-coverage edges from the initial de Bruijn graph generated by LJA's jumboDBG tool. This method of running LJA is compared against other assembly methods and discussed in detail in Appendix B.18, "Haplotypes of the most mutated gene in CAMP".

3.5.8 Data sets

The SheepGut read-set is available at the NCBI BioProject database (https://www.ncbi.nlm.nih.gov/bioproject/) under accession number PRJNA595610. We used the HiFi sequencing data from accession IDs SRX7628648 and SRX10647529, in particular. We note that although there exists additional Hi-C and Illumina short-read sequencing data for SheepGut [15], we have only made use of the HiFi data (from the two accession IDs given above) in this paper.

To simplify reproduction of our analyses, we have also made the metaFlye assembly graph produced for the SheepGut data set (Appendix B.2, "Assembly graph"), which represents a starting point for the analyses shown in this paper, available on Zenodo (https://doi.org/10.5281/zenodo.6545141).

The chicken gut metagenome read-set is available at the NCBI Sequence Read Archive (SRA; https://www.ncbi.nlm.nih.gov/sra) under accession number SRR15214153. We retrieved the hifiasm-meta [48] assembly of this data set produced by [48] from Zenodo (https://doi.org/10.5281/zenodo.6330282).

3.5.9 Software dependencies

The strainFlye pipeline is implemented as a Python 3 command-line tool. The pipeline code directly relies on pysam (https://github.com/pysam-developers/pysam), pysamstats (https://github.com/alimanfoo/pysamstats), scikit-bio (http://scikit-bio.org), NetworkX [65], pandas [211, 143], click (https://click.palletsprojects.com/), NumPy [196], SciPy [203], SAMtools [107, 34], BCFtools [34], minimap2 [106], Prodigal [80], and LJA [8].

The analyses shown throughout this paper (including Bash scripts, Python 3 scripts, and Jupyter notebooks) [95] additionally make use of metaFlye [97], CheckM [145], barrnap [173], the SRA toolkit (https://github.com/ncbi/sra-tools), LoFreq [213], the Integrative Genomics Viewer (IGV) [190], Bandage [212], MetagenomeScope [46], Jupyter [95], nbconvert (https://nbconvert.readthedocs.io), scikit-learn [147], matplotlib [78], Logomaker [187], and the neato [53] and sfdp [76] layout methods in Graphviz [54]. We installed software primarily using conda (https://conda.io), mamba (https://mamba.readthedocs.io), and pip (https://pip.pypa.io).

Finally, we note that we produced Figure 3.1 using LibreOffice Draw (https://www.libreoffice.org/discover/draw/) and GIMP (https://www.gimp.org/). We produced the example link graphs shown within this figure, in particular, using Graphviz' [54] neato [53] layout method. We also modified the MetagenomeScope visualization of a multiplex de Bruijn graph shown in Figure 3.6 using LibreOffice Draw, in order to add a box highlighting a region of the graph.

3.5.10 Software availability

strainFlye's code is available on GitHub (https://github.com/fedarko/strainFlye) and as Supplemental Code S1, available online at https://www.genome.org/cgi/doi/10.1101/gr.276917.122. Additional ad hoc code for performing the analyses shown in this paper

is available on GitHub (https://github.com/fedarko/sheepgut) and as Supplemental Code S2, available online at https://www.genome.org/cgi/doi/10.1101/gr.276917.122.

3.6 Competing interest statement

The authors declare no competing interests.

3.7 Acknowledgements

We thank the editor and anonymous reviewers for their feedback. We thank Andrey Bzikadze for initial exploration of the TDA and context-dependent TDA in the case of short-read sequencing. We thank Nuno Bandeira for detailed feedback, including bringing the possibility of pseudogenes impacting these analyses to our attention. We thank Anton Bankevich for help with applying LJA [8] to haplotype assembly. We acknowledge Nicholas Bokulich for bringing Figure 1 of [198] to our attention: https://forum.qiime2.org/t/9061/13. We thank Lee Katz and Kai Blin for advice on parsing gene prediction data. We thank Heng Li for answering questions about minimap2. We thank Andrei Osterman and Semen Leyn for helpful discussions and advice on analyzing the SheepGut data set, including alignment filtering and interpretation of variation in CAMP. M.W.F. thanks Alex Richter for bringing the K_a and K_s values to his attention. The rainbow colorscheme used in Figure 3.2 was partially inspired by Figure 2 of [40]. This work was partly supported by IBM Research AI through the AI Horizons Network and the UC San Diego Center for Microbiome Innovation (to M.W.F.); and by National Institutes of Health Common Fund Award, National Institute of Diabetes and Digestive and Kidney Diseases, U24DK131617-01.

Chapter 3, in full, is a reprint of the material as it appears in "Analyzing rare mutations in metagenomes assembled using long and accurate reads." **Fedarko MW**, Kolmogorov M, and Pevzner PA. *Genome Research 32*(11-12), 2022. The dissertation

author was the primary investigator and first author of this paper.

Chapter 4

Efficient creation and visualization of exact dot plot matrices

4.1 Abstract

Dot plots, which represent matches between two sequences, are popular visualization methods for comparing sequences. The underlying matrix represented by a dot plot, describing every k-mer shared between two sequences, can also have use besides visualization—for example, in identifying synteny blocks, which typically manifest as diagonal clusters of matching k-mers. However, many dot plot visualization tools do not provide easy programmatic access to this matrix. In some cases this is attributable to the fact that these tools do not actually create the "true" dot plot matrix: instead, many tools attempt to reduce time or space requirements by creating an approximate representation of this matrix (for example, by only identifying the most "significant" matches). Furthermore: often we wish to display many dot plot visualizations in a single figure. This sort of figure, in which many dot plots are "tiled" in a grid, can be useful for showing a high-level visual overview of the similarities between many pairs of sequences (for example, the legs of bulges in an assembly graph). Generating each of the dot plots in such a figure, and combining these plots into a larger figure, is cumbersome to do using graphical-user-interface-based dot plot tools alone. A programmatic interface for creating many dot plots and organizing them into a larger figure would be helpful to improve the speed and reproducibility of this

process. To enable exact dot plot matrix construction and programmatic access—and to expedite the process of constructing complex figures consisting of many dot plots tiled together—I present wotplot, a well-tested and easy-to-use Python library that satisfies these needs.

4.2 Introduction

4.2.1 Motivation

Since their introduction in 1970 [59], dot plots have become a valuable visualization method for comparing sequences. Dot plots have been used to identify synteny blocks [148], visualize the repeat structure of highly-reptitive genomic regions [21, 219], and compare strains of bacterial genomes [33], among myriad other tasks.

Dot plots are also often used to characterize genomic inversions [25]; these diagrams reveal differences between "canonical" inversions and other inversion-like structures that can lead to false positive inversion identifications, such as inverted duplications and inverted repeats. During in-progress research on developing a method for identifying microinversions [25] from the bulges of a de Bruijn graph [30], I needed a tool that could provide access to the exact dot plot matrix between two sequences (for performing filtering and clustering on reverse-complementary matches) and visualize many dot plots at once (in order to facilitate quick manual evaluation of many bulges' putative microinversions at once). I was unable to find any software packages which met these needs; this led me to create wotplot, the tool described here.

4.2.2 Related work

A variety of other tools have been developed for the visualization of dot plots; here I briefly examine and compare some of these tools with the proposed approach, using a laptop with 8 GB of RAM as a test machine. Although performance (i.e. the time required

to generate a dot plot) was not my main goal in the development of wotplot, I use the example of comparing two strains of *E. coli* (shown in Figure 4.1) as a simple informal benchmark for evaluating how fast these tools are at producing a single dot plot of two bacterial genomes. For reference, wotplot can create and visualize this dot plot—using $k = 20$—in roughly 36 seconds on this machine.

Gepard [101] can visualize dot plots of large sequences; like wotplot, it supports the use of suffix arrays to speed up match detection [112]. Using Gepard to create a dot plot of the two *E. coli* strains shown in Figure 4.1, using a "word length" of 20, runs faster than wotplot on this example—Gepard takes about 18 seconds. However, Gepard does not seem to provide access to the resulting dot plot matrices, rendering the tool infeasible for analyses where the exact output matrix is needed. Additionally, combining dot plot visualizations produced by Gepard's graphical user interface is somewhat cumbersome; Gepard does have a command-line interface available that makes this task easier to automate, but each of the plots output by Gepard appear to unavoidably include long descriptions of parameters embedded in the figures themselves, which is often undesirable when tiling many dot plots into a single figure. Finally, Gepard struggles to visualize dot plots of relatively small sequences legibly; comparing two 18-nucleotide sequences from an example figure in [28], for example, results in an extremely small plot that cannot apparently be enlarged within Gepard or using any of its parameters.

FlexiDot [174] contains a wide variety of options for creating tiled grids of dot plots. However, like Gepard, FlexiDot—as of the current latest version, 2.0.1—does not seem to provide access to the resulting dot plot matrices. FlexiDot version 1.06 (the newest version available from April 14, 2019 until May 21, 2025) did not seem to scale well to the visualization of large sequences; using FlexiDot version 1.0.6 to construct the $k = 20$ dot plot of the two *E. coli* strains shown in Figure 4.1 ran for over ten minutes before I killed the process. Notably, newer versions of FlexiDot have been released in recent weeks that make the tool more efficient; FlexiDot version 2.0.1 is able to create a dot plot of the

two *E. coli* strains shown in Figure 4.1 (using parameters `-k 20 -c -m 1`) in about 41 seconds, which is now only slightly slower than wotplot.

ModDotPlot [186] also supports many visualization options, including the creation of tiled grids of dot plots. Although the tool is designed for the interactive visualization of dot plots (e.g. by zooming in on repetitive regions), it also supports the generation of static plots, like wotplot. The tool uses "modimizers" to speed up the process of sequence comparison; using it to generate a static plot comparing the two *E. coli* strains shown in Figure 4.1 (using a k-mer size of 20) takes about 55 seconds, which is slightly slower than wotplot. Although ModDotPlot is quite useful, particularly for interactive visualization, its output heatmaps are not perfectly comparable to exact dot plots (where "dots" correspond only to exact k-mer matches, given a fixed value of k). Additionally, the tool does not support short input sequences; it has been recommended that it is used only for input sequences greater than 10 kbp long (https://github.com/marbl/ModDotPlot/issues/30), which is an undesirable constraint for arbitrary dot plot visualization. (As shown in Figure 4b of [97], many bubbles in metagenome assembly graphs have lengths less than 10 kbp.) These points aside, it is worth noting that ModDotPlot can write out information about the similarities it identifies to a BED file [156], which could then be converted to a matrix format. This seems to be a unique feature among the tools surveyed here.

Many useful tools for dot plot visualization exist, including and beyond the three described above. However, none of the tools I could find met the exact needs I had. Next I will describe wotplot, the tool I created to address these needs.

4.3 Results

4.3.1 Creating exact dot plots of long sequences

Figure 4.1 shows a dot plot produced by wotplot comparing the genomes of two *Escherichia coli* strains: the model organism *E. coli* K-12 [161] and the pathogenic strain

E. coli O157:H7 [71]. Comparing these genomes at this high level demonstrates that, although these strains' genomes are broadly similar (as indicated by the red diagonal line of forward matches spanning the plot), the O157:H7 strain's genome contains numerous insertions relative to the K-12 strain's genome (indicated by areas where the red diagonal line "jumps" to the right). This figure, which can be created in roughly 36 seconds on a laptop with 8 GB of RAM, demonstrates wotplot's ability to create and visualize exact dot plot matrices comparing entire bacterial genomes on a laptop computer.

Beyond the fast generation of exact dot plots, I note that the `DotPlotMatrix` object produced by wotplot (containing a sparse representation of all k-mer matches between the input sequences) is an ordinary Python object that can be written to disk using Python's built-in `pickle` module. These matrices can thus be analyzed and visualized any time after they have been created, without needing to recompute them.

4.3.2 Visualizing multiple dot plots in a single figure

wotplot facilitates the rapid visual comparison of multiple sequences—for example, pairs of contigs from a metagenome assembly that have been binned together, or pairs of bulge sequences from an assembly graph. One of the main reasons I created this library was to simplify the process of creating a "grid figure" containing many dot plots.

wotplot's default visualization functions directly use matplotlib [78], which provides utilities for creating figures of subplots arranged in a variety of layouts. This integration enables the programmatic creation of complex grid figures of many dot plots. As a simple example of this, Figure 4.2 uses wotplot and matplotlib to create a grid figure showing an all-versus-all comparison of five random sequences.

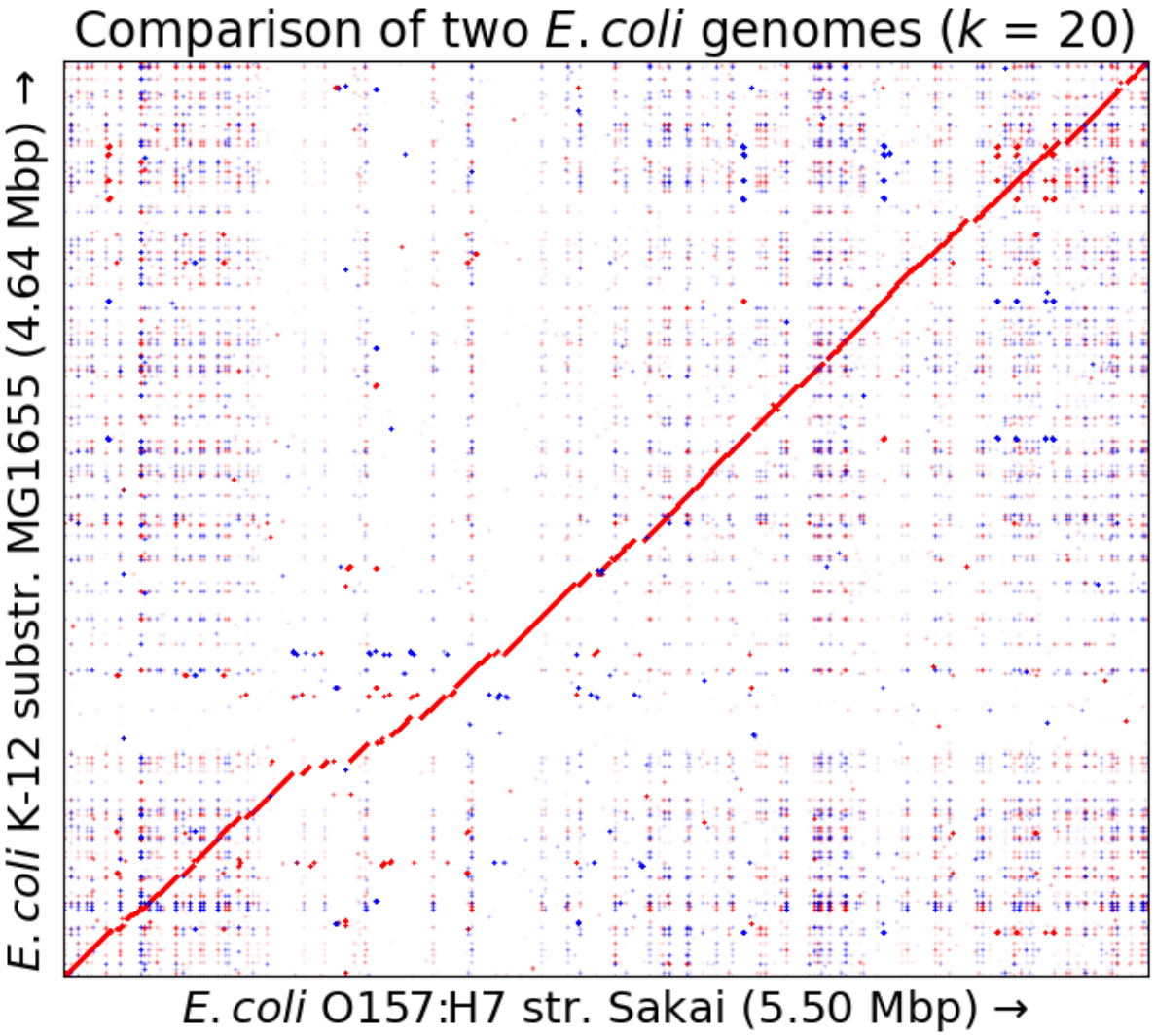
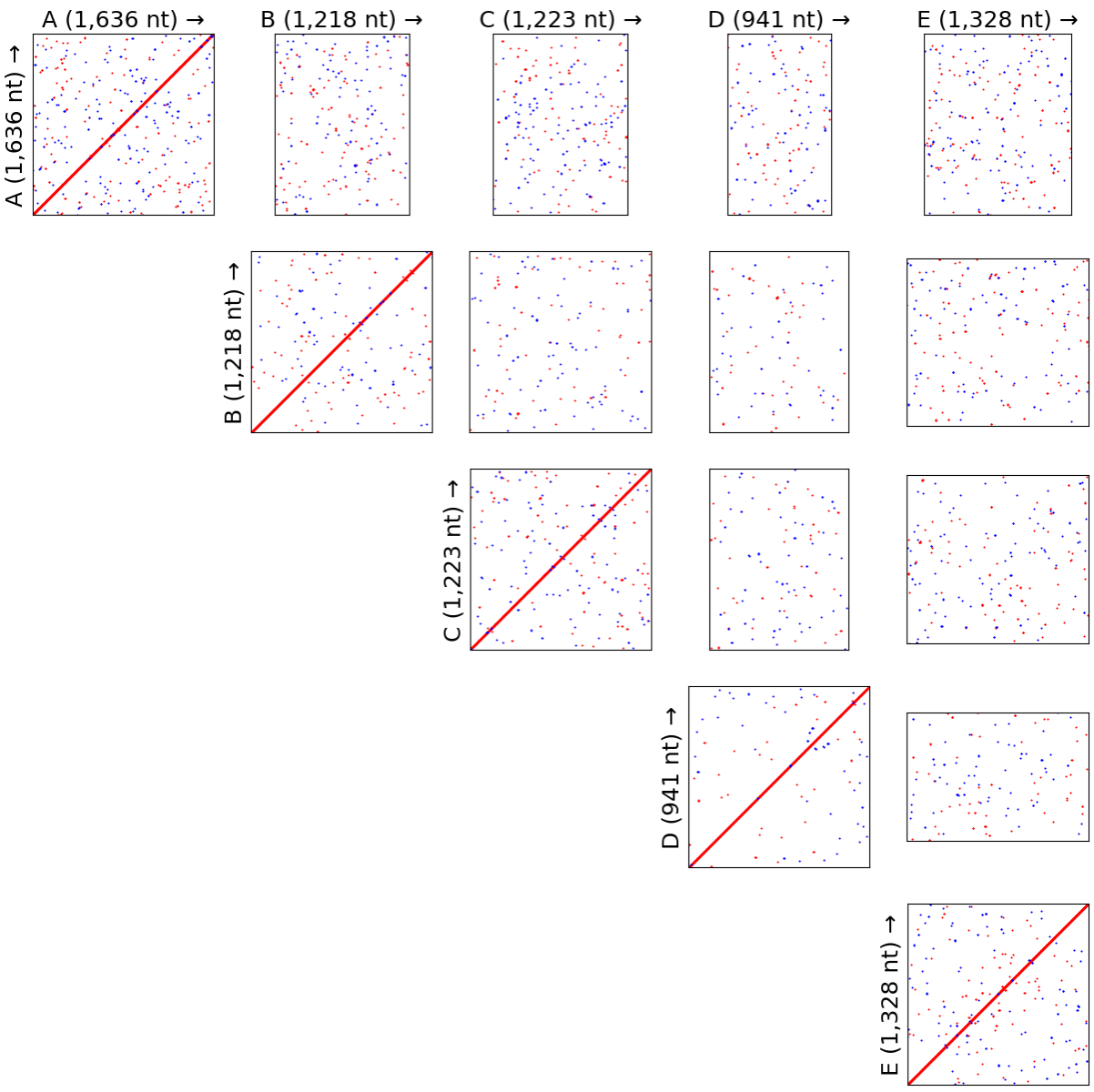


Figure 4.1. Exact $k = 20$ dot plot comparing the genomes of *Escherichia coli* O157:H7 (x-axis) and *Escherichia coli* K-12 (y-axis), analogous to the plot shown in Figure 2a of [71]. Every visible dot in the matrix represents the occurrence of a 20-mer that is present in both genomes. Red dots correspond to forward matches, blue dots correspond to reverse-complementary matches, and purple dots correspond to palindromic matches. wotplot can generate this figure in roughly 36 seconds on a laptop with 8 GB of RAM. Note that, to simplify comparison, I did not include the two plasmid sequences of the O157:H7 strain in this figure.

Figure 4.2. All-versus-all dot plot comparisons of five random sequences. I generated five random sequences (each with a length randomly chosen from the range $[500, 2000]$), and used wotplot and matplotlib [78] to create and visualize a grid of $k = 7$ dot plots comparing each pair of these sequences. Each sequence takes up its own row and column of the grid; since the grid is symmetric about the diagonal (the dot plot of sequences A and B is equivalent to a rotated version of the dot plot of B and A), I have only drawn the upper triangle of this grid. Since each sequence perfectly matches itself, the self-dot-plots comparing sequences with themselves (shown on the diagonal of this figure) have clear red diagonal lines throughout indicating perfect forward matches. The "off-diagonal" dot plots do not display this pattern.


Dot plots (k = 7) of five random sequences
A (1,636 nt) →
B (1,218 nt) →
C (1,223 nt) →
D (941 nt) →
E (1,328 nt) →
A (1,636 nt) →
B (1,218 nt) →
C (1,223 nt) →
D (941 nt) →
E (1,328 nt) →

4.4 Methods

4.4.1 Space-efficient matrix storage

A naïve implementation of dot plot matrix construction will become impractical for large sequences, because the space required to store the "full" dot plot matrix of two sequences of lengths m and n is $O(mn)$.

Say we wish to construct the dot plot matrix of two *Escherichia coli* strains' genomes—for example, between *E. coli* K-12 [161] and *E. coli* O157:H7 [71], as shown in Figure 4.1. Even if we make the unrealistically low assumption that every cell in the matrix takes up only a single bit of storage, storing the full dot plot matrix of these two genomes (of lengths 4.64 Mbp and 5.50 Mbp, respectively) would require roughly 3.19 terabytes. Provisioning this amount of memory is infeasible on most modern computers.

To address this problem wotplot uses SciPy [203] to store the dot plot matrix in sparse format, which only requires the storage of information about nonzero (i.e. match) cells in the matrix. Most dot plot matrices (using sufficiently large k-mer sizes) are extremely sparse, so storing these matrices in a sparse format greatly reduces their space requirements: there are 3,536,693 nonzero cells in the $k = 20$ dot plot matrix of *E. coli* K-12 vs. *E. coli* O157:H7, meaning that this matrix is 99.999986% sparse. If we again assume that each stored entry in this matrix takes up one bit of storage, then the sparse matrix now requires only 442.09 kilobytes—a much more feasible amount of memory. (In practice, we can use Python's `pickle` module to write the dot plot matrix object for this example to disk. The resulting file is approximately 66.73 megabytes, which is still relatively small.)

4.4.2 Space- and time-efficient identification of shared k-mers

Using a sparse representation reduces the space requirements of our constructed dot plot matrix, but the process of actually constructing this matrix (based on the locations of

all pairs of shared k-mers between the two input sequences) remains a potential performance bottleneck.

We could compute this information using a simple quadratic approach, in which every k-mer in one sequence (of length m) is explicitly compared with every k-mer in the other sequence (of length n). However, the $O(mn)$ time requirements of such a solution are impractical for large sequences.

These prohibitive time requirements can be reduced by precomputing information about the k-mers present in the sequences and using this information to speed up the computation of the dot plot matrix [29]. However, this can lead to impractical space requirements: for example, naïvely storing all k-mers in one sequence (of total length n) and these k-mers' locations requires that—in the worst-case scenario (where every k-mer in the sequence is unique)—we store $(n - k + 1)$ k-mers and their locations. For large n and large k, this space requirement is problematic.

There are a variety of ways to surmount this problem, including for example the use of hash maps or Bloom filters [8]. Like Gepard [101], wotplot addresses this problem by using suffix arrays [112] to identify shared k-mers between the two input sequences.

However, there are multiple ways to use suffix arrays to identify shared k-mers; currently, wotplot supports the use of two algorithms that solve this problem. In both cases, wotplot makes use of `pydivsufsort` [1], a Python package that provides access to the `libdivsufsort` library for fast suffix array construction [50].

Identifying shared k-mers using suffix arrays and LCP arrays

Currently, the default shared k-mer identification method provided by wotplot directly uses the `pydivsufsort` library [1]'s `common_substrings()` method (https://github.com/louisabraham/pydivsufsort/issues/42).

This method concatenates the two input strings, uses divsufsort [50] to construct the suffix array of this concatenate, and uses Kasai's algorithm [88] to construct the longest

common prefix (LCP) array of this concatenate. pydivsufsort uses various optimizations to speed up the process of identifying common substrings using these data structures. wotplot runs this method twice, once after reverse-complementing one of the strings, to identify reverse-complementary and palindromic matches.

Although this method is quite fast (using it, wotplot can generate a $k = 20$ dot plot of the two *E. coli* strains shown in Figure 4.1 in roughly 36 seconds), benchmarking demonstrates that it requires an infeasible amount of memory when comparing sequences longer than around 20 Mbp each on a laptop with 8 GB of RAM. This is likely due in part to the method's use of Kasai's algorithm, which has substantial space requirements [115].

Below we discuss the other algorithm wotplot supports for identifying shared k-mers, which has requires much less memory at the cost of increasing the required runtime.

Identifying shared k-mers using suffix arrays alone

As an alternative method, wotplot can avoid constructing an LCP array—and limit its "initial" work to simply constructing the suffix arrays of both input strings. This method is referred to as "suff-only" in wotplot's interface.

Given two sequences (of lengths m and n), this method constructs their suffix arrays separately, again using divsufsort [1, 50]. It then iterates simultaneously through both suffix arrays from start to end in order to record all positions of all shared k-mers. This approach is reasonably space-efficient compared to the `common_substrings()` method, since it does not incur the space requirements of running Kasai's algorithm (and the space requirement of storing the resulting LCP array). (To identify reverse-complementary and palindromic matches, wotplot repeats this procedure after reverse-complementing one of the strings.)

This method is slower than the `common_substrings()` method (using wotplot to generate the $k = 20$ dot plot of the two *E. coli* strains shown in Figure 4.1 using this method takes roughly 2 minutes and 35 seconds); in part, this is likely because the lack of

an LCP array makes iteration through the suffix arrays slower than it could be. However, due to this method's notably lower space requirements, it enables the creation of exact dot plot matrices of very large sequences on low-memory systems. On a laptop with 8 GB of RAM, this method was able to create the $k = 20$ dot plot matrix of two random 150 Mbp sequences in 1 hour, 8 minutes, and 46.89 seconds.

4.4.3 Rapid visualization of large dot plot matrices

wotplot uses matplotlib [78]'s `spy()` function to visualize only the nonzero (match) entries of dot plot matrices; even large matrices can be plotted very quickly in this way.

Creating figures programmatically is often an iterative process [62], and the separation of dot plot matrix creation and visualization expedites this process. After creating the $k = 20$ dot plot matrix of two *E. coli* strains shown in Figure 4.1 (which takes approximately 35 seconds), visualizing this matrix typically takes less than a second. This simplifies the process of interactive figure creation, making it easy for researchers to repeatedly try out various visualization parameters until they are satisfied with the resulting figure.

4.4.4 Data availability

I downloaded the *E. coli* K-12 genome used to create Figure 4.1 from RefSeq (ID `NC_000913.3`). I also downloaded the *E. coli* O157:H7 genome used to create Figure 4.1 from RefSeq (ID `NC_002695.2`). To facilitate comparison with the K-12 genome, I omitted the two plasmid sequences of O157:H7 when creating Figure 4.1.

4.4.5 Software dependencies

wotplot's code directly relies on NumPy [70], SciPy [203], matplotlib [78], and pydivsufsort [1]. In addition to these dependencies, I used pyfastx [38] to load the *E. coli* sequences used to generate Figure 4.1, and used the memory_profiler [146] package to

benchmark memory use.

4.4.6 Software availability

wotplot's source code is available on GitHub at https://github.com/fedarko/wotplot. As of writing, wotplot supports all Python versions $\geq$ Python 3.6; its code is automatically tested on Python 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12, and 3.13.

wotplot's GitHub repository includes a detailed tutorial explaining how to use this software, including demonstrations of how to generate Figures 4.1 and 4.2. It also presents benchmarking results demonstrating the use of wotplot on pairs of randomly-generated DNA sequences of increasing length, showing how wotplot's time and memory requirements increase accordingly.

4.5 Discussion

Here I presented wotplot, a software library for the efficient creation and visualization of dot plot matrices. I created this library to fill a gap in the field, and the library has served this purpose well. It has been useful in the development of metaLJA in Chapter 2, and I hope it will be similarly useful for other bioinformatics projects.

Although the tool satisfies the needs I had when creating it, there is still room to make it more efficient. For example, during the process of dot plot matrix creation, we are only interested in identifying exact matches of length k between our input sequences. However, we use suffix arrays to identify these matches. Since creating the suffix array of a string requires the comparison (implicit or explicit) of all characters within each suffix of this string, rather than just the first k characters of each suffix of this string, the use of suffix arrays to identify matching k-mers could be thought of as overkill. In this vein, there are doubtlessly more efficient algorithms and implementation techniques that could be applied to speed up this tool.

4.6 Acknowledgements

Chapter 4, in full, is a reprint of the material as it appears in "Efficient creation and visualization of exact dot plot matrices." **Fedarko MW**. *In preparation.* The dissertation author was the primary investigator and sole author of this paper.

Appendix A

Supplemental material for Chapter 1

A.1 Supplemental material for Chapter 1.1

A.1.1 Computing feature differentials using Songbird

As discussed in the main text, the initial focus of this re-analysis was on visualizing the associations of features with different *Scomber japonicus* body sites. To assess this, we ran Songbird [132] using the formula `C(sample_type_body_site, Treatment('sea water'))`. This produced six fields of differentials:

1. `Intercept`
2. `C(sample_type_body_site, Treatment('sea water'))[T.fish GI]`
3. `C(sample_type_body_site, Treatment('sea water'))[T.fish digesta]`
4. `C(sample_type_body_site, Treatment('sea water'))[T.fish gill]`
5. `C(sample_type_body_site, Treatment('sea water'))[T.fish pyloric caeca]`
6. `C(sample_type_body_site, Treatment('sea water'))[T.fish skin]`

The last five fields of differentials (2–6) describe the association of features with samples from each of the studied body sites, using seawater samples as a reference via Treatment coding.

(For reference, the fourth differential field, `C(sample_type_body_site, Treatment('sea water'))[T.fish gill]`, is what is shown in the rank plot sub-figures in the main text.)

The first field, `Intercept`, is less easily interpretable and not particularly relevant to the case study; this field is produced automatically by Patsy (https://patsy.readthedocs.io), the library used by Songbird to represent input formulae as design matrices.

Songbird mathematical details

As is also described in [132], the multinomial regression used in Songbird is given as follows:

$$\beta \sim N(0, \sigma)$$
$$\eta_i = \text{alr}^{-1}(X_i\beta)$$
$$Y_i \sim \text{Multinomial}(\eta_i)$$

where $\beta \in \mathbb{R}^{k \times d-1}$ represents the regression coefficients for d features and k covariates, X_i are the covariate measurements for each sample i, and Y_i are the feature counts in sample i. A normal prior with variance σ is used to regularize the regression coefficients. A maximum likelihood procedure is employed to identify the optimal regression coefficients.

The vectors $\beta_k \in \mathbb{R}^{d-1}$ are in alr coordinates, and as a result can be represented as compositions by $\text{alr}^{-1}(\beta_k) \in S^d$. In [132], these vectors are referred to as *differentials*. Since ranking is shift invariant, the ordering of this composition is agnostic to the choice of reference frame. As a result, the features can be sorted by their coefficients in β_k. By default, these differentials are represented in clr coordinates.

The regression implemented in Songbird is similar to the methodology in other differential abundance tools, such as ALDEx2 [49] and DESeq2 [109]. The estimated

regression coefficients from any of these tools can be visualized in Qurro: a fully worked example demonstrating the use of Qurro with ALDEx2 outputs is linked to from Qurro's Tutorials section of its README, located at https://github.com/biocore/qurro.

Differential names

Due to some current technical limitations, Qurro (as of writing) changes or removes certain special characters like `[` or `'` from field names. This is why the gill differential field name shown in Qurro—`C(sample_type_body_site, Treatment(sea water))(T:fish gill)`—has a slightly different name than it did in Songbird's output. (This behavior is documented in Qurro's README, which is distributed with its source code at https://github.com/biocore/qurro.)

A.1.2 Qurro log-ratio-selection controls used

The log-ratios selected in Figures 1.1 and 1.2 were selected using Qurro's filtering controls in the following way.

Figure 1.1 (*Shewanella* to *Synechococcales*)

1. The numerator was selected by filtering to features where the `Taxon` field contained the text `Shewanella`.

2. The denominator was selected by filtering to features where the `Taxon` field contained the text `Synechococcales`.

Figure 1.2 (*Shewanella* to bottom $\sim 10\%$ features)

1. The numerator was selected by filtering to features where the `Taxon` field contained the text `Shewanella`.

2. The denominator was selected by filtering to features where the gill differential value—that is, `C(sample_type_body_site, Treatment(sea water))(T:fish gill)`—

was less than `-2.102`. (This value was chosen in order to make the denominator include exactly the bottom 98 features.)

Screenshots of using these controls in Qurro are shown in Figures A.1 and A.2.

Screenshot details

As the warning shown on the left side of the screenshots in Figures A.1 and A.2 explains, the rank plot in these screenshots has been scaled so that each bar (feature) has a width of less than 1 pixel: this is done in order to show more of the rank plot on the screen at once. Unchecking the `Fit bar widths to a constant plot width?` checkbox resets the bar widths in the rank plot to a larger value comparable to that shown in Figures 1.1 and 1.2, albeit one that results in the full rank plot not being visible all at once on most screens without horizontally scrolling.

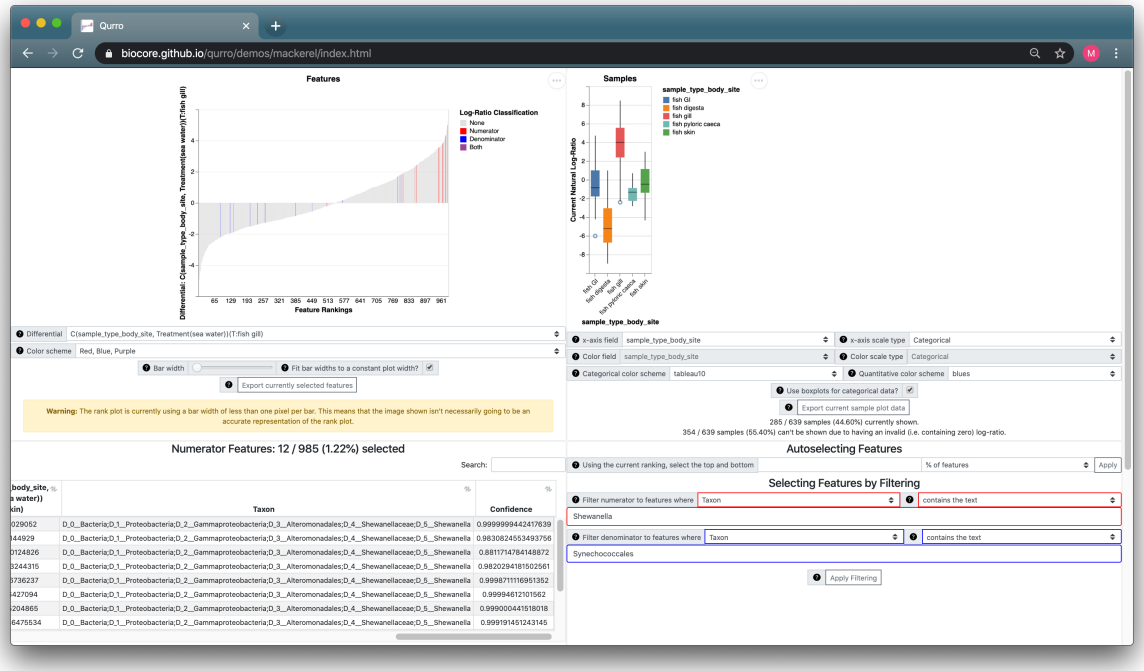
These visualizations were generated using Qurro version 0.6.0 and are displayed here on a macOS 10.15.3 laptop using Google Chrome version 80.0.3987.132. When taking these screenshots, the browser was zoomed out somewhat to show more of the controls and the `Numerator Features` table at the bottom-left of the screen was scrolled to the right to show selected numerator features' classified taxonomy information.

A.1.3 Details on Qurro (and Songbird) input data filtering

Running Qurro requires a few distinct input files (or QIIME 2 artifacts, if running it as a QIIME 2 plugin): a feature table, a "rankings" file, a sample metadata file, and optionally a feature metadata file.

If any features within the feature table are not present in the input rankings, then Qurro will not include these features in the output visualization (since they would not be displayable on the rank plot). This means that, although Qurro doesn't impose very strict filtering guidelines on its own by default, the filtering behaviors of upstream "ranking" tools will necessarily impact the amount of data shown in Qurro.

Figure A.1. Screenshots of a Qurro visualization of the case study data, showing the controls used to recreate Figures 1.1A–C. Note the text entered in the `Selecting Features by Filtering` section at the bottom-right of the screen, which shows the textual queries used to select a log-ratio of *Shewanella* features to *Synechococcales* features.


Qurro
biocore.github.io/qurro/demos/mackerel/index.html
Features
Log-Ratio Classification
None
Numerator
Denominator
Both
Feature Rankings
Samples
sample_type_body_site
fish GI
fish digesta
fish gill
fish pyloric caeca
fish skin
Current Natural Log-Ratio
Differential
Color scheme
Red, Blue, Purple
Bar width
Fit bar widths to a constant plot width?
Export currently selected features
Warning: The rank plot is currently using a bar width of less than one pixel per bar. This means that the image shown isn't necessarily going to be an accurate representation of the rank plot.
x-axis field
x-axis scale type
Categorical
Color field
Color scale type
Categorical
Categorical color scheme
tableau10
Quantitative color scheme
blues
Use boxplots for categorical data?
Export current sample plot data
285 / 639 samples (44.60%) currently shown.
354 / 639 samples (55.40%) can't be shown due to having an invalid (i.e. containing zero) log-ratio.
Numerator Features: 12 / 985 (1.22%) selected
Search:
Taxon
Confidence
Autoselecting Features
Using the current ranking, select the top and bottom
% of features
Apply
Selecting Features by Filtering
Filter numerator to features where
Taxon
contains the text
Shewanella
Filter denominator to features where
Synechococcales
Apply Filtering

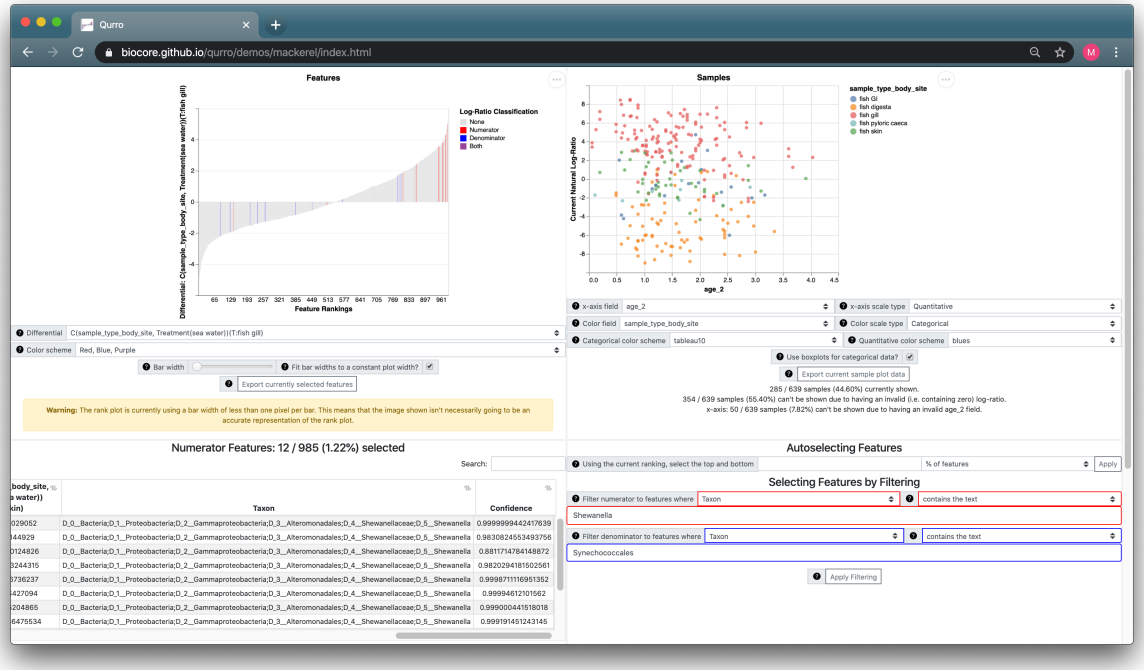
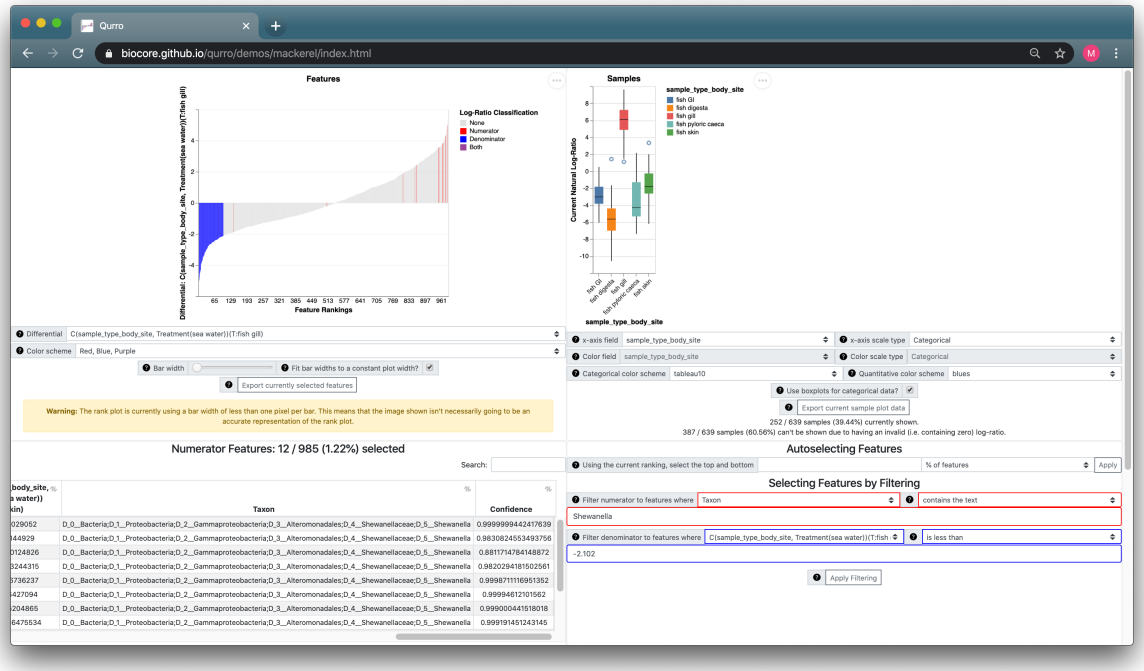
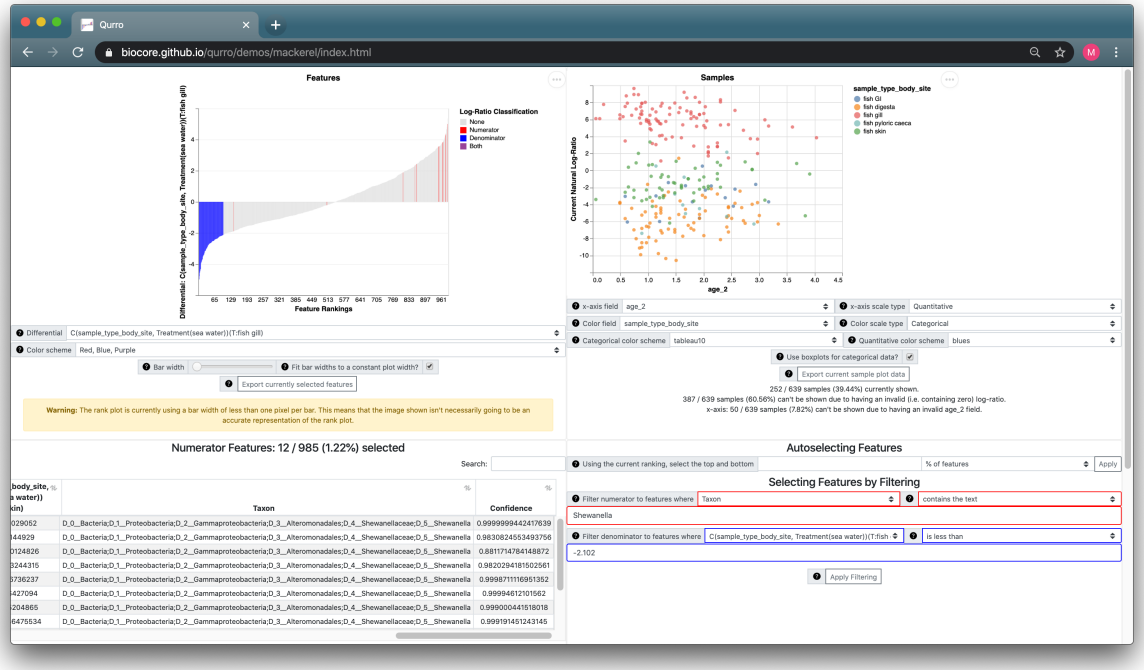

Qurro
biocore.github.io/qurro/demos/mackerel/index.html
Features
Log-Ratio Classification
None
Numerator
Denominator
Both
Feature Rankings
Samples
sample_type_body_site
fish GI
fish digesta
fish gill
fish pyloric caeca
fish skin
Current Natural Log-Ratio
age_2
Differential
Color scheme
Red, Blue, Purple
Bar width
Fit bar widths to a constant plot width?
Export currently selected features
Warning: The rank plot is currently using a bar width of less than one pixel per bar. This means that the image shown isn't necessarily going to be an accurate representation of the rank plot.
x-axis field
age_2
x-axis scale type
Quantitative
Color field
sample_type_body_site
Color scale type
Categorical
Categorical color scheme
tableau10
Quantitative color scheme
blues
Use boxplots for categorical data?
Export current sample plot data
285 / 639 samples (44.60%) currently shown.
354 / 639 samples (55.40%) can't be shown due to having an invalid (i.e. containing zero) log-ratio.
x-axis: 50 / 639 samples (7.82%) can't be shown due to having an invalid age_2 field.
Numerator Features: 12 / 985 (1.22%) selected
Search:
Taxon
Confidence
Autoselecting Features
Using the current ranking, select the top and bottom
% of features
Apply
Selecting Features by Filtering
Filter numerator to features where
Taxon
contains the text
Shewanella
Filter denominator to features where
Synechococcales
Apply Filtering

Figure A.2. Screenshots of a Qurro visualization of the case study data, taken analogously to those in Figure A.1 but this time showing the controls used to recreate Figures 1.2A–C. The difference between these screenshots and those in Figure A.1 is due to the selected denominator features, which now comprise the bottom 98-ranked features for the gill differentials rather than the classified *Synechococcales* features. Although it is cut off somewhat by the dropdown's width, the first dropdown after the `Filter denominator to features where` label indicates that the `C(sample_type_body_site, Treatment(sea water))(T:fish gill)` differential field is selected.


Qurro
biocore.github.io/qurro/demos/mackerel/index.html
Features
Log-Ratio Classification
None
Numerator
Denominator
Both
Feature Rankings
Samples
sample_type_body_site
fish GI
fish digesta
fish gill
fish pyloric caeca
fish skin
Current Natural Log-Ratio
Numerator Features: 12 / 985 (1.22%) selected
Taxon
Confidence
Autoselecting Features
Selecting Features by Filtering
Shewanella
-2.102
Apply Filtering


Qurro
biocore.github.io/qurro/demos/mackerel/index.html
Features
Log-Ratio Classification
None
Numerator
Denominator
Both
Feature Rankings
Samples
sample_type_body_site
fish GI
fish digesta
fish gill
fish pyloric caeca
fish skin
Current Natural Log-Ratio
age_2
Numerator Features: 12 / 985 (1.22%) selected
Taxon
Confidence
Autoselecting Features
Selecting Features by Filtering
Shewanella
-2.102
Apply Filtering

Since this impacts the case study, we go into detail about this behavior here.

Songbird's `--min-feature-count`

For the case study dataset described in the manuscript, there were 23,253 features present in the feature table before running Songbird. However, Songbird applies a default `--min-feature-count` (i.e. the minimum number of samples a feature must appear in) of 10: this resulted in a large amount of features being removed from the visualization due to only appearing in a handful of samples. This is why there are just 985 features in the resulting Qurro visualization. (When generating a Qurro visualization, Qurro will output details explaining—if applicable—why certain samples/features have been removed from the visualization.)

Why aren't there any seawater samples shown in the paper figures?

One of the things we noticed midway through this case study was that *Shewanella* spp., for the most part, did not appear in seawater samples. To help explain this, we prepared a Jupyter Notebook [95] that shows why these samples have been dropped. This notebook is available in the repository https://github.com/knightlab-analyses/qurro-mackerel-analysis.

Non-numeric `age_2` values.

Since the `age_2` field refers to the estimated age of a sample's host fish, this field is not meaningful for non-fish samples like seawater. As shown in the notebook, all of the 50 seawater samples in our feature table have a non-numeric `age_2` value—this is one of the "reasons" Qurro has for dropping samples from the sample plot, and it explains why seawater samples cannot be shown in Figures 1.1C or 1.2C.

Relative lack of *Shewanella* features.

As shown in the notebook, only one of the 50 seawater samples in our feature table included a feature classified as *Shewanella*. This particular *Shewanella* feature only appears

in two samples in the feature table (including the aforementioned seawater sample), so it is not ranked by Songbird due to the default `--min-feature-count` described above.

From the Qurro visualization's perspective, then, none of the seawater samples contains any *Shewanella* features—so visualizing both of the log-ratios shown in the paper's case study will necessarily involve filtering out all of the seawater samples, unless imputation of some form were to be used. This is the reason why seawater samples are not shown in Figures 1.1B and 1.2B, and it's a reason (in addition to the `age_2` reason) why seawater samples are not shown in Figures 1.1C and 1.2C.

Reflection on this phenomenon.

We note that the large amount of samples dropped here was likely caused in part by the nature of the case study. Since in general different body sites are expected to harbor different microbial communities, it makes sense that taxa common in one sample type might go almost or completely undetected in other sample types.

When looking for differentially abundant taxa across more subtly different sample categories (e.g. skin samples at different timepoints in the progression of atopic dermatitis, as shown in [132]), we expect that sample dropout like what we observed with seawater samples here will be less of an issue.

A.2 Supplemental material for Chapter 1.2

A.2.1 Differential abundance comparison of oral microbiomes

To provide additional context to the relationship between Songbird [132], ALDEx2 [49], and ANCOM [113]'s measures of differential abundance, Figure A.3 shows a scatterplot of these methods' outputs.

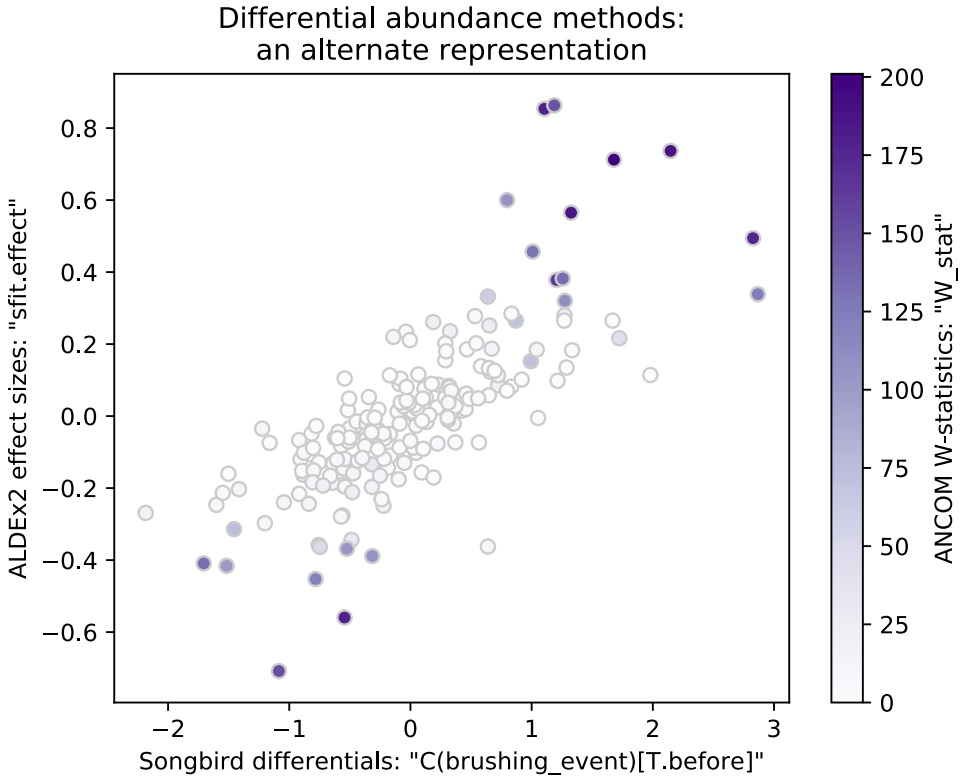


Figure A.3. Scatterplot comparing the three differential abundance methods' results shown in Figure 1.4C. The x-axis and y-axis represent the Songbird differentials and ALDEx2 effect sizes for the features in the dataset, while each feature point is colored by its ANCOM W-statistic. This demonstrates that our starting interpretation of how these results relate—i.e. that Songbird and ALDEx2's results have similar "directionality" between the before- and after- toothbrushing states, and that ANCOM's W-statistic lacks this same directionality—is reasonable. The nine features for which no Songbird differentials were available are omitted from this plot. This scatterplot was produced using matplotlib [78].

A.2.2 Animated analysis of SARS-CoV-2

Supplemental Movie S1 (available online at https://doi.org/10.1128/mSystems.01216-20) demonstrates a longitudinal community analysis of SARS-CoV-2 using phylogenetically-informed animations. Each tip in the tree represents a SARS-CoV-2 genome. Each sphere in the ordination represents the genomes collected over a 7-day window. This movie showcases some analytical capabilities available in EMPress.

We gratefully acknowledge the authors, originating and submitting laboratories of the sequences from GISAID's Database on which this analysis is based. The list is detailed in Supplemental Table S1, available online at https://doi.org/10.1128/mSystems.01216-20.

Appendix B

Supplemental material for Chapter 3

B.1 Read alignment

strainFlye aligns reads to edge sequences in the assembly graph in order to identify mutations that may have been "smoothed over" in the process of metagenome assembly: the resulting alignment is a prerequisite for mutation identification, and impacts all downstream analyses in strainFlye. The `strainFlye align` command uses minimap2 [106] to compute an alignment, after which it filters various sources of noise from the alignment; however, this command can be skipped if a user has an existing BAM file representing an alignment of reads to edges. This Appendix describes how `strainFlye align` works.

Performing alignment.

strainFlye uses minimap2 [106] to align all reads against the assembly graph's sequences. By default, we use the `asm20` preset parameter set for the alignment per recommendation from the minimap2 documentation for use with HiFi data; however, this and other minimap2 parameters are easily adjustable by the user, for example if the `map-hifi` preset (introduced in recent versions of minimap2) is preferred instead. We also use the `--secondary=no` parameter to exclude secondary alignments [188]. After minimap2 creates the initial alignment, we then use SAMtools [107] to convert it to a sorted and indexed BAM file.

In the context of the SheepGut dataset, some edges may have no reads aligned to them, even before we filter the alignment. In our experience this can happen for edges that are shorter than the usual read length; these edges (which tend to have extremely high coverages) seem to be an assembly artifact from metaFlye, likely corresponding to repetitive sequences, and most of the analyses in this paper implicitly ignore these edges.

We note that this alignment process retains supplementary (also referred to as "chimeric") alignments, in which a read may have multiple distinct linear alignments [188]; these alignments can be useful in describing structural variation, or in representing a read that spans the start and end of a single circular sequence. Although we preserve supplementary alignments, we do take pains to remove reads that have alignments overlapping with other alignments from the same read: the remainder of this Appendix provides details on the filtering steps strainFlye performs on minimap2's alignment, as well as potential extensions to these filters.

Filtering out overlapping supplementary alignments.

Given a single read with two linear alignments A_1 and A_2 to a reference genome, there are two types of "overlap" we could define between A_1 and A_2. We note that these types of overlap are not necessarily mutually exclusive.

The first type of overlap is defined with respect to the reference genome. In this case, A_1 and A_2 align to regions of the reference genome that overlap with each other.

The second type of overlap is defined with respect to the read sequence itself. In this case, A_1 and A_2 originate from overlapping regions of the read sequence.

We refer to these types of overlap as *reference* and *read* overlap, respectively. The SAM specification currently states that "[for] a chimeric alignment, the linear alignments constituting the alignment are largely non-overlapping" [188]; however, it is unclear whether the type of overlap described is reference or read overlap.

Neither type of overlap bodes particularly well in the context of our analyses. Here,

we focus on removing reference-overlapping supplementary alignments; the consequences of these alignments on downstream analysis seem more problematic than the consequences of read-overlapping alignments.

While performing the analyses described in this paper, we noticed that a small, albeit substantial, number of supplementary alignments from individual reads exhibited reference overlap. Strangely, these overlaps sometimes spanned thousands of nucleotides in a read; these may thus be a sequencing artifact, since HiFi reads have been shown to be vulnerable to molecular chimeras [210].

Before filtering overlapping supplementary alignments and before filtering partially-mapped reads, the following supplementary alignment statistics held for the three selected MAGs.

In CAMP, 8,645 / 503,385 (1.72%) unique reads aligned to within the MAG had supplementary alignments within CAMP. On average, these 8,645 reads had 2.07 alignments within CAMP. Furthermore, 2,180 / 503,385 (0.43%) unique reads aligned to within CAMP had supplementary alignments within CAMP *and* had reference overlap between at least one pair of their alignments within CAMP. The average length of these overlaps (considering all pairs of overlapping alignments within CAMP from the same read) was 580.12 bp.

In BACT1, 6,067 / 268,075 (2.26%) unique reads aligned to within the MAG had supplementary alignments within BACT1. On average, these 6,067 reads had 2.04 alignments within BACT1. Furthermore, 470 / 268,075 (0.18%) unique reads aligned to within BACT1 had supplementary alignments within BACT1 *and* had reference overlap between at least one pair of their alignments within BACT1. The average length of these overlaps (considering all pairs of overlapping alignments within BACT1 from the same read) was 1,417.80 bp.

In BACT2, 5,497 / 745,461 (0.74%) unique reads aligned to within the MAG had supplementary alignments within BACT2. On average, these 5,497 reads had 2.04

alignments within BACT2. Furthermore, 728 / 745,461 (0.10%) unique reads aligned to within BACT2 had supplementary alignments within BACT2 *and* had reference overlap between at least one pair of their alignments within BACT2. The average length of these overlaps (considering all pairs of overlapping alignments within BACT2 from the same read) was 2,561.42 bp.

Regardless of whether these reads with overlapping supplementary alignments correspond to chimeras or not, the presence of these overlaps complicates interpretation of the alignment data—especially in analyses that take into account the sequence carried on individual reads, for example phasing and codon mutation analyses.

Although it is possible to only filter some of a read's supplementary alignments as needed to prevent reference overlap, we act on the assumption that the presence of *any* reference-overlapping alignments for a given read implies that this read is unreliable for our analyses. This will likely be overly strict—some "correct" reads may simply happen to have reference overlap in their supplementary alignments—but we contend that, in the context of identifying rare mutations, overly strict filtering is defensible. We therefore identify all reads with reference-overlapping supplementary alignments relative to any of the edges in the alignment, and completely filter all of these reads from the alignment. The impacts of this filter on sequencing coverage in the three selected MAGs are shown in Figure B.4.

Slight error in the overlapping supplementary alignment filter.

We note that the version of this filter we used for the alignments throughout this paper was overly strict, due to an off-by-two error in its computation of "overlap" between linear alignments of a read (rather than subtracting 1 from each alignment's end coordinate, it mistakenly added 1). This resulted in a relatively small amount of reads being erroneously removed by our filter and thus not included in this paper's results—anecdotally, many of these removed reads represent cases where minimap2 would create

two distinct linear alignments of a read directly next to each other, but not overlapping, on a single contig.

Based on analyzing initial versions of the SheepGut and ChickenGut alignments (reflecting just the output of minimap2, before applying our filters; we re-ran the alignment processes to obtain these "intermediate" alignment files), the number of reads that would have been erroneously removed by this bug is relatively small: 27,211 reads (0.12%) in SheepGut, and 1,596 reads (0.08%) in ChickenGut. In particular, the three selected MAGs in SheepGut (CAMP, BACT1, BACT2) are only subtly affected: there were 16, 9, and 3 reads erroneously filtered for these MAGs, respectively. (For reference, the statistics quoted earlier in this Appendix about overlapping supplementary alignments in the three selected genomes are correct.) We expect that this bug should have a minimal impact on the paper's conclusions; moving forward, we have also fixed this bug in version 0.2.0 of strainFlye.

Filtering out partially-mapped reads.

Although we instruct minimap2 to not output secondary alignments, our preservation of supplementary alignments means that a single read may still be mapped to multiple contigs.

Reads that are mapped to multiple contigs in this way may be of good quality and worth retaining: for example, if a genome is split into multiple edges within a connected component of the assembly graph, then we might expect some reads' alignments to also be split across different edges within this component. Similarly, since most bacterial genomes are circular [23], reads that happen to cover both the "start" and "end" of a circular MAG may be split into a supplementary alignment [188].

However, some reads are less desirable. For example, a read with alignments to edges located in completely different connected components of the assembly graph—with only short alignments to individual edges—is likely not useful in identifying rare mutations,

and should be removed. This section details our attempt to filter out these sorts of "partially-mapped reads," with the goal of removing undesirable supplementary alignments between unrelated contigs while preserving ordinary supplementary alignments within a contig or between adjacent contigs.

An earlier method we attempted to use for this involved simply filtering out reads that contained more than a set number of soft-clipping operations, as mentioned at https://github.com/samtools/samtools/issues/1169. However, we noticed that this caused coverage drops near the ends of MAGs' sequences due to penalizing the aforementioned "desirable" supplementary alignments.

We therefore took a different strategy in the current implementation of this filter. Our filter considers each edge sequence, and then considers all reads aligned to this edge, counting the number of match and/or mismatch operations in the CIGAR strings of each read's alignment to this edge. If an assembly graph file is provided to `strainFlye align` (so that we know which edges are adjacent to each other), and if this edge has less than 50 adjacent edges in the graph (ignoring links between the edge and itself), the script also checks reads aligned to these adjacent edges and also counts match/mismatch operations from any reads that are aligned to both the adjacent edge and the initially-considered edge. A given read is only included in the filtered alignment for this edge if the sum of all match/mismatch operations in this read in this edge (and in adjacent edges, if applicable) is at least 90% of the read length.

We note that this method is not perfect—for example, supplementary alignments can still have overlaps relative to their read sequence (as described earlier in this Appendix), so the sum of match/mismatch operations as we currently compute them is not necessarily a perfect measure of how much of a read is mapped to a given MAG. Additionally, the reason for the coverage drops near the end of CAMP (Figure B.3; CAMP is the only of the three selected MAGs located in an assembly graph component containing other edges), in spite of our attempts to allow alignments to adjacent edges in the same graph to count

towards the 90% threshold, is unclear. This may be due to the "full genome" of CAMP being split across the edges in its component, and this complicating the alignment process. In spite of these imperfections, from manual examination of its results the filter's behavior seems acceptable. The impacts of this filter on sequencing coverage in the three selected MAGs are implied in the comparison between the second plot for each MAG in Figure B.4 and the plots in Figure B.3: removing partially-mapped reads results in many of the extreme "peaks" in Figure B.4 being removed.

Potential filtering extensions.

Although strainFlye's default alignment filtering methods seem suitable for our purposes, these methods could certainly be extended to perform even more stringent filtering. More stringent methods should further increase the likelihood that any observed mismatch at a position corresponds to a real mutation, rather than an error or an artifact of alignment.

For example, homonucleotide sequences have been known to pose problems for long-read sequencing technologies; although we have not explicitly accounted for this effect in our analyses, it would be possible to limit our focus in certain cases—for example, the construction of a decoy contig—to more complex genomic regions. Similarly, the filtering of contigs with many "deletion-rich" positions (Appendix B.3) may also prove useful.

B.2 Assembly graph

Choice of dataset and assembler.

Although strainFlye was developed to work with output from the metaFlye assembler [97], it can be extended to MAGs generated by other HiFi-based tools such as HiCanu [141], hifiasm-meta [48], and LJA [8]. However, our reliance on long and accurate reads implies that the use of short reads, or long error-prone reads, will likely pose nontrivial challenges for strainFlye.

Running metaFlye.

When assembling the SheepGut dataset using metaFlye [97], we did not use metaFlye's `--keep-haplotypes` mode. We chose this course of action in order to reduce fragmentation in the assembly graph (giving us longer, coarser MAGs to use as a starting point for studying mutations within these MAGs).

Connected components of the assembly graph.

21,750 out of 78,793 edges (27.6%) within the SheepGut assembly graph produced by metaFlye are located within a single large "hairball" connected component of the graph. This is a common characteristic of metagenomic assembly graphs due to, for example, overlap between reads from highly conserved regions (such as 16S rRNA genes) or the presence of eukaryotic genomes within a metagenome (this dataset includes some eukaryotic genomes [97]). The second largest component of the assembly graph contains 252 edges. 45,842 out of 45,988 connected components (99.7%) contain 10 or fewer edges and 43,045 (93.6%) consist of a single edge.

There is a large amount of variance between edge coverages by reads, as reported by metaFlye, in the assembly graph. The maximum edge coverage is 7,536,987x (for a 580bp long edge located in the "hairball" largest connected component). The average coverage across all edges (rounded to the nearest integer) is 951x, while the median coverage is 14x: the discrepancy between average and median is likely due to the presence of high-coverage outlier edges, such as the aforementioned maximum-coverage edge.

Summarizing all graph components by length and coverage.

As shown in [5], many of the connected components of the assembly graph containing a small number of edges likely arise from low-coverage genomes, viruses, or plasmids. These are therefore not relevant to our selection of high-coverage bacterial MAGs for these analyses.

In an attempt to distinguish between these cases and components that capture

bacterial genomes, we plotted a summary of edge coverage and length for each component. Figure B.1 shows a scatterplot where each component is represented as a single symbol, and is plotted by the total edge length in the component and by the sum of (coverage times length) for all edges in the component divided by the total edge length in the component [5].

This plot provides insight into which components in the graph are likely to represent bacterial genomes. We represent the thresholds used in our initial selection process for edges in the graph (length $\geq$ 1 Mbp, coverage $\geq$ 1,000x) as vertical and horizontal lines in the plot, respectively. This display clarifies that, while many components have a high "average" coverage or a high total length, few components have both. These components represent edges that serve as good candidates for demonstrating the analyses shown in this paper.

We do note that the length threshold shown here could arguably be reduced, since some bacteria have genomes smaller than 1 Mbp [36]; and that we initially used these thresholds to search for edges, not components, in the graph. However, this figure nonetheless indicates that the three selected MAGs are, relative to those available from this dataset, of good quality.

Details about the three selected MAGs.

Table B.1 provides basic information about CAMP, BACT1, and BACT2.

BACT1 and BACT2 were assembled into edges located in isolated connected components (i.e. connected components containing just one edge) of the full dataset's assembly graph. The MAG we have named CAMP, however, is located in a component of the graph containing 31 other edges: this component is shown in Figure B.2, for reference. Although CAMP has not been completely assembled, we have nonetheless included it in the analyses throughout this paper due to its relatively high coverage. CheckM [145] estimates "completeness" values of 81.4% for CAMP, 96.9% for BACT1, and 94.6% for

Figure B.1. Summary of coverage and length across all connected components of the assembly graph. Each symbol (asterisk, square, plus sign, triangle, circle) represents a single component in the graph: the x-axis shows the total edge length for each component, and the y-axis shows an aggregate measure of coverage for each component (using coverages as defined by metaFlye and not based on the alignment-based $reads(pos)$ values computed in 3.3.2 and used elsewhere in this paper). In order to relate this plot to our selection criteria for MAGs (discussed in section 3.3.1, "Demonstrating strainFlye"), we plot lines showing the length and coverage minimum cutoffs we used on the plot (length $\geq$ 1 Mbp, coverage $\geq$ 1,000x). The few points located in the top-right "quadrant" defined by these lines represent MAGs that pass these cutoffs. The components containing the three MAGs we focus on throughout these analyses—CAMP, BACT1, and BACT2—are colored specially and shown as distinct symbols in the plot. It is clear that these three MAGs, and only three additional components in the graph, pass both the length and coverage cutoffs. We assign the "hairball" component of the graph a special color and symbol in order to simplify comparison of it with the other components in the graph: although this component has a high total edge length (causing it to appear as an outlier even on a log scale), its aggregate coverage is relatively low. Lastly, we note that the lack of components to the left of x = 10^2 is a reflection of the fact that all components in the graph have a total edge length greater than 100 bp. However, the aggregate measure of coverage is not similarly high: for many components, this value is 0x. We have set both the x and y axes to both have a minimum of 0 in order to clarify this.

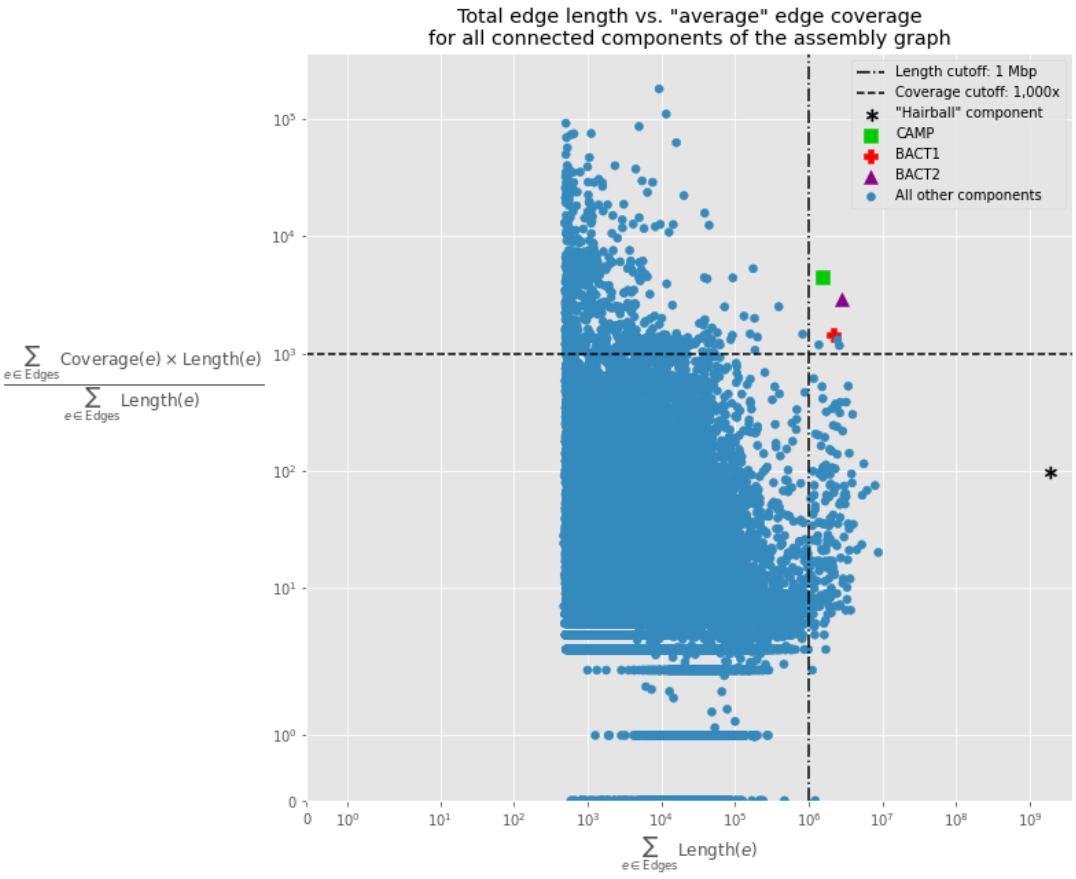

Total edge length vs. "average" edge coverage
for all connected components of the assembly graph
Length cutoff: 1 Mbp
Coverage cutoff: 1,000x
"Hairball" component
CAMP
BACT1
BACT2
All other components
$\frac{\sum_{e \in \text{Edges}} \text{Coverage}(e) \times \text{Length}(e)}{\sum_{e \in \text{Edges}} \text{Length}(e)}$
$\sum_{e \in \text{Edges}} \text{Length}(e)$

Table B.1. The three selected MAGs for mutation analyses. The names CAMP, BACT1, and BACT2 refer, respectively, to edges with IDs 6104, 1671, and 2358 in the metaFlye assembly graph. We note that CAMP's completeness is relatively low because it was not assembled into a single circular sequence. Since predicted PCGs can overlap, we "count" a position present in multiple genes only once when computing the amount of positions in PCGs.

Name	Length (Mbp)	Coverage	Protein-Coding Genes (PCGs)	Intergenic Regions (IRs)	Positions in PCGs (Mbp)	Positions in IRs (Mbp)
CAMP	1.29	4,159x	1,297	788	1.19	0.10
BACT1	2.15	1,415x	1,761	1,559	1.95	0.21
BACT2	2.81	2,993x	2,567	2,196	2.29	0.51

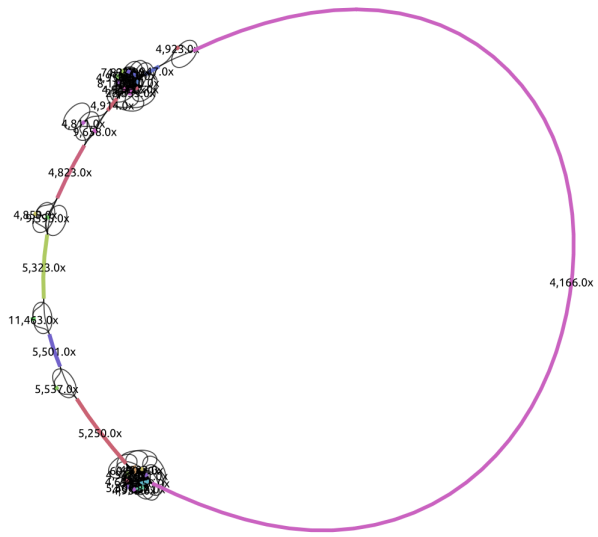


Figure B.2. Visualization of the connected component containing edge 6104 (corresponding to CAMP) in the larger assembly graph, shown to clarify the component's structure. Coverages (reported by metaFlye, and slightly different from the (mis)match coverages based on our alignment that are shown in Table B.1) are shown overlaid on the edges within the graph. The long magenta edge shown with coverage 4,166.0x is edge 6104. We visualized this graph using Bandage [212].

BACT2; and "contamination" values of 0.0% for CAMP, 0.6% for BACT1, and 0.4% for BACT2. The relatively low completeness estimate for CAMP is likely a result of its incomplete assembly.

Taxonomic classifications of the three selected MAGs.

We used Kaiju [124]'s web server to classify these MAGs. We used the "NCBI BLAST nr +euk" reference database and default parameters.

B.3 Coverages and deletion-rich positions

Coverages of the three selected MAGs in SheepGut.

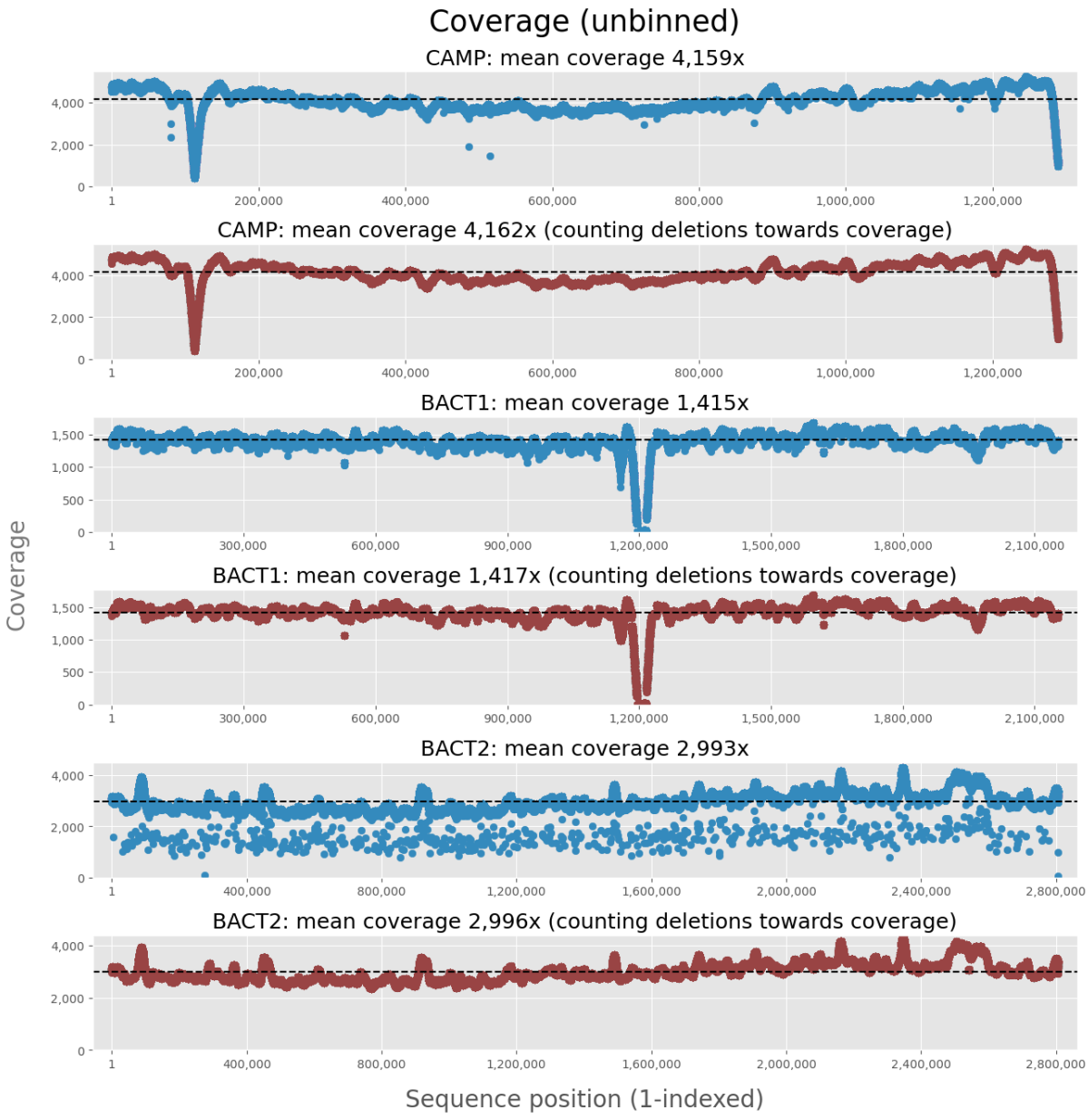
Although many of the analyses in this paper assume that coverage is roughly uniform throughout the three selected MAGs, their coverages do vary somewhat. To illustrate this, Figure B.3 shows plots of coverage for each position in the three selected MAGs. Unlike the coverage plots shown in the Appendix B.16, these coverages are not binned or normalized.

Coverage drops in SheepGut.

Figure B.3 reveals a surprising result in BACT2 (and, to a much smaller extent, CAMP and BACT1): the presence of many individual positions to which many deletions are aligned, causing the coverage based purely on matches and mismatches for these positions to be relatively low. Manual inspection of some of these positions with samtools tview [107] confirms this.

More gradual coverage drops like those seen near the ends of CAMP, or near the middle of BACT1, could be in theory explained by real coverage variation in sequencing or noise introduced through the assembly and/or alignment processes. Figure B.4 presents coverage plots similar to those in Figure B.3, demonstrating that the two gradual coverage drops near the ends of CAMP (and, to an extent, in the middle of BACT1) indeed seem to have been introduced by our alignment filtering steps (Appendix B.1). However, these

Figure B.3. Coverage throughout the three selected MAGs, with and without deletions. Each dot represents a single position in a MAG. For the first plot for each MAG (using blue points), the coverage at a given position is defined as the total number of aligned reads to this position that have a match or mismatch operation, ignoring insertions and deletions; this is also how we computed the average MAG coverages in Table B.1. For the second plot for each MAG (using red points), each position's coverage now includes the number of deletions aligned to it. The average coverage for each plot is shown as a horizontal dashed line. Although most parts of these MAGs have somewhat uniform coverage, especially in the "including-deletions" plots, both plots for CAMP and for BACT1 have noticeable gradual drop(s) in coverage. Furthermore, BACT2 has a relatively large amount of "outlier" positions with many deletions: these positions have relatively low coverages for the first plot, and these low coverages vanish when counting deletions towards coverage. These "deletion-rich" positions are investigated in this Appendix.


Coverage (unbinned)
CAMP: mean coverage 4,159x
CAMP: mean coverage 4,162x (counting deletions towards coverage)
BACT1: mean coverage 1,415x
BACT1: mean coverage 1,417x (counting deletions towards coverage)
BACT2: mean coverage 2,993x
BACT2: mean coverage 2,996x (counting deletions towards coverage)
Coverage
Sequence position (1-indexed)

plots also show that the high amounts of deletions in the aforemented positions in BACT2 were not introduced by these filters, indicating that further investigation is needed as to the origin of this phenomenon.

Deletion-rich positions in SheepGut and ChickenGut.

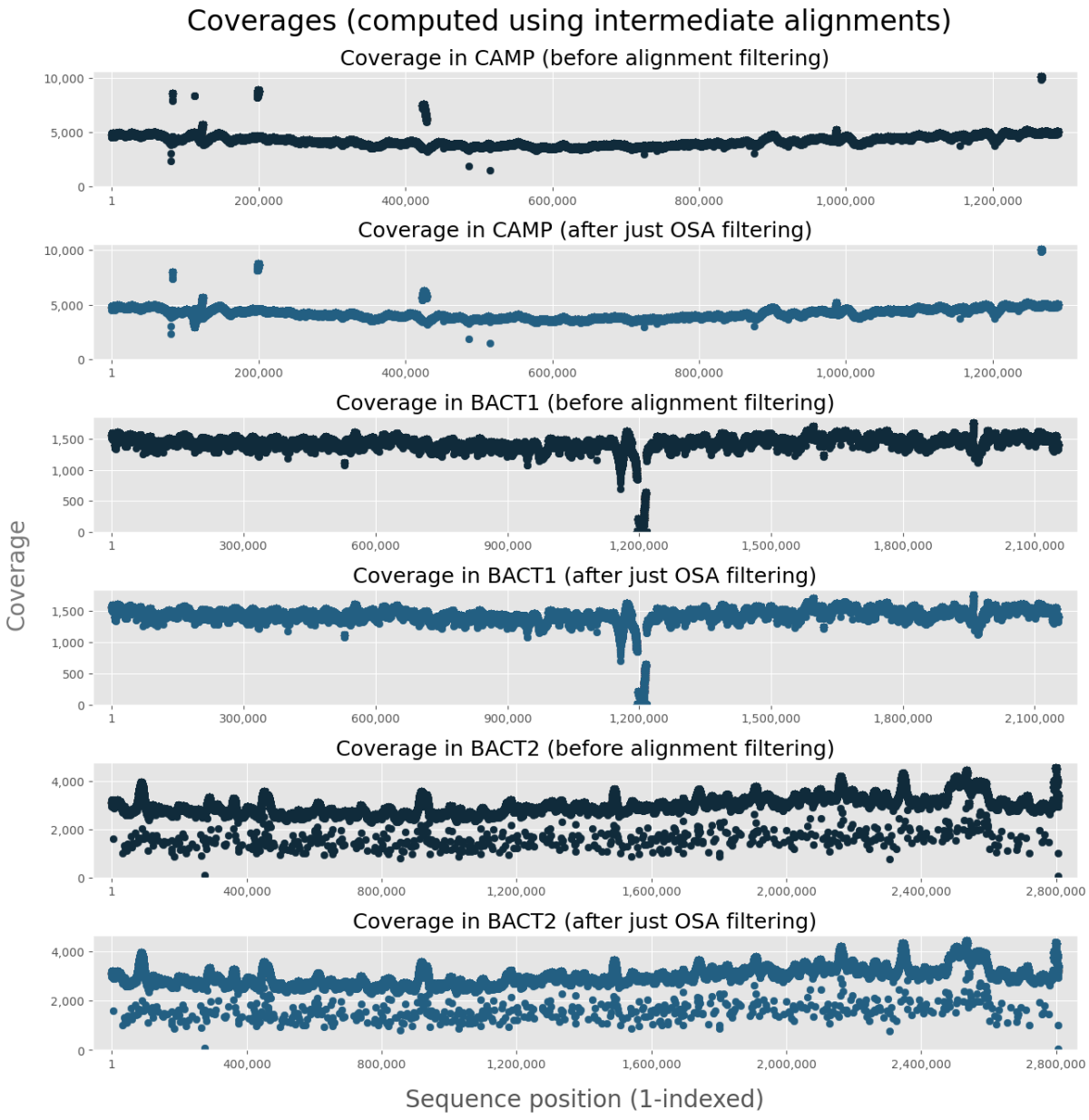
We define a position in a contig as *deletion-rich* if, for some integer d, there are d or more deletions aligned to this position. For example, most of the outlier positions in BACT2 are deletion-rich for $d = 500$.

This definition enables us to ask two immediate questions. First, considering various values of d, do there exist many deletion-rich positions in other contigs in SheepGut besides the three selected MAGs? And, if so: does this pattern also hold in other contigs in other HiFi metagenomic datasets besides SheepGut?

Figure B.5 answers both of these questions affirmatively. First, many contigs in SheepGut (besides the three selected MAGs) contain many deletion-rich positions even for high values of d; second, this pattern also holds for many contigs in ChickenGut (Appendix B.4). Since the contigs in ChickenGut were assembled using hifiasm-meta [48], while the contigs in SheepGut were assembled using metaFlye [97], the persistence of many deletion-rich positions across many contigs in both datasets implies that these positions are not introduced by the assembly process.

These deletion-rich positions may happen to correspond to real variation in these MAGs; they could also, in theory, be introduced by minimap2 [106], which we used to align reads to contigs for both SheepGut and ChickenGut. However, we believe it is much more likely that these positions are an artifact of HiFi sequencing that—for some reason—introduces many deletions at certain positions, e.g., positions corresponding to homonucleotide and dinucleotide runs that often trigger errors in HiFi reads [141]. Other analyses of HiFi data have shown gradual coverage drops and suggested that these may arise from problems in HiFi sequencing [141, 139]; these deletion-rich positions may have

Figure B.4. Coverage throughout the three selected MAGs, before each of the two alignment filtering steps. The first row for each MAG plots the coverage after performing alignment using minimap2 [106], but before any additional filtering. The second row for each MAG plots the coverage after using just the overlapping-supplementary-alignment filter (Appendix B.1; this filter is also impacted by the off-by-two bug discussed there). Finally, Figure B.3 demonstrates the coverages for each MAG after both the overlapping-supplementary-alignment filter and the partially-mapped read filter (Appendix B.1). Primarily, this figure demonstrates to us that the many deletion-rich positions in BACT2 were not introduced, somehow, by these filters. It does, however, indicate that the drops in coverage near the ends in CAMP were introduced by the partially-mapped read filter, since these drops are not observed in this figure but are observed in Figure B.3. Parts of the coverage drop in the middle of BACT1 seem also to have been introduced by the partially-mapped read filter. We note that we needed to re-run the full alignment process for SheepGut to produce this figure, since we did not initially preserve intermediate alignment files in order to save storage space. These intermediate alignments may thus not be a perfect match with the intermediate alignments upstream of the one used elsewhere in this paper.


Coverages (computed using intermediate alignments)
Coverage in CAMP (before alignment filtering)
Coverage in CAMP (after just OSA filtering)
Coverage in BACT1 (before alignment filtering)
Coverage in BACT1 (after just OSA filtering)
Coverage in BACT2 (before alignment filtering)
Coverage in BACT2 (after just OSA filtering)
Coverage
Sequence position (1-indexed)

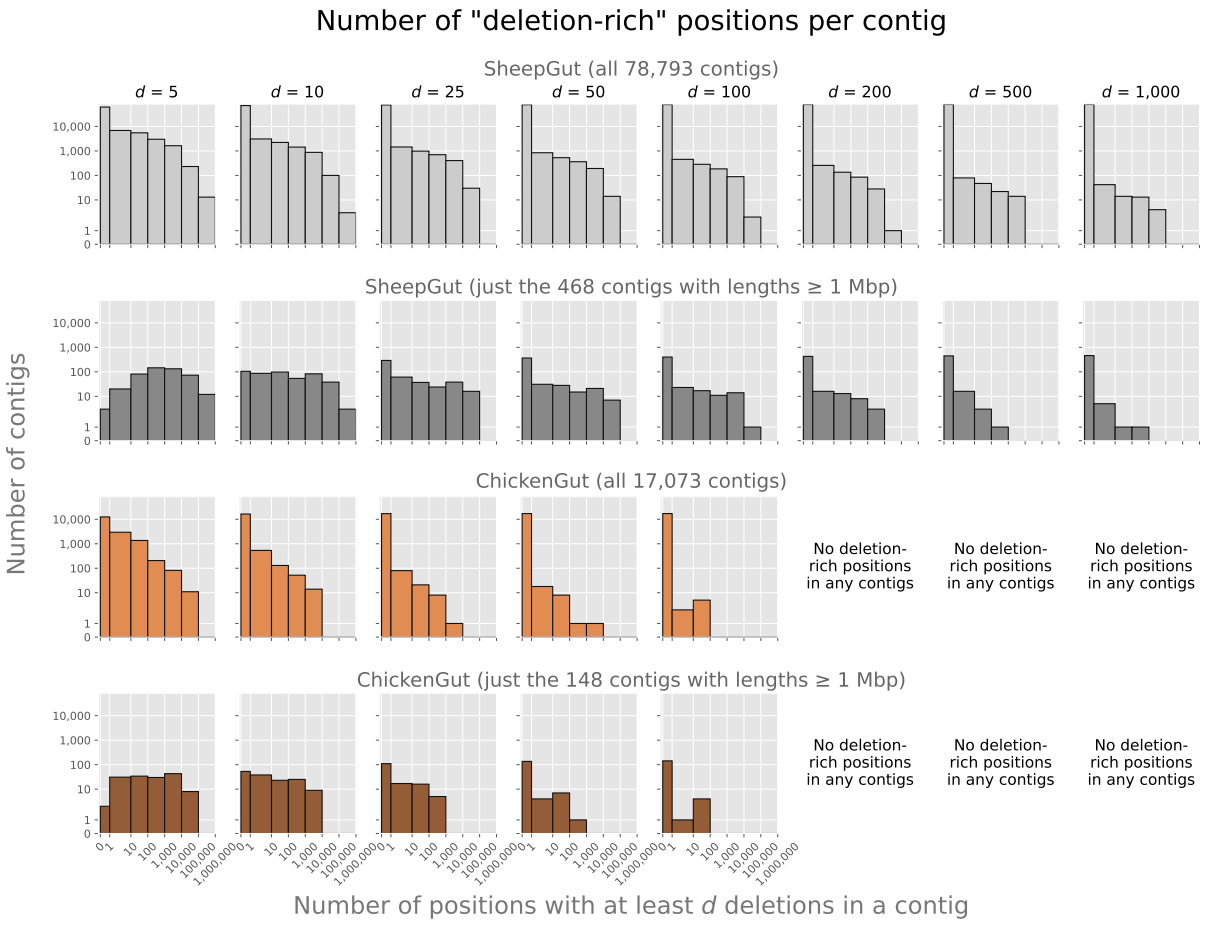


Figure B.5. Histograms of the amounts of deletion-rich positions in each MAG. Each row corresponds to a different dataset, or subset of a dataset; each column describes a value of d we use to classify a position as deletion-rich. These plots demonstrate that, in general, many contigs in each dataset contain many deletion-rich positions, indicating that the high amount of "outlier" positions shown in BACT2 in Figures B.3 and B.4 is not unique to this MAG. We omit histograms for $d \geq 200$ for the two ChickenGut datasets because no deletion-rich positions exist for these values of d in either dataset: we believe that this is primarily due to the coverages in the contigs in ChickenGut being, in general, smaller than those of SheepGut (Appendix B.4).

a similar origin.

Setting aside their origin, deletion-rich positions are outnumbered by other positions in the three selected MAGs. In BACT2, for example: for $d \in \{5, 10, 25, 50, 100, 200, 500, 1{,}000\}$, BACT2 (2,806,161 bp long) has {413,276, 222,869, 64,660, 17,458, 2,272, 652, 591, 541} deletion-rich positions, respectively. The percentages of BACT2's length comprised of deletion-rich positions are thus {14.727%, 7.942%, 2.304%, 0.622%, 0.081%, 0.023%, 0.021%, 0.019%}, showing that (for large values of d) BACT2 has relatively few deletion-rich positions.

Implications of deletion-rich positions and other coverage drops.

It is unclear what the "best practice" is for how to handle the presence of deletion-rich positions in these datasets—whether we should still include deletion-rich positions in our analyses, ignore these positions for the purposes of mutation calling, or even ignore entire contigs that contain many deletion-rich positions. For now, we do not explicitly filter these positions or the contigs that contain them, although doing so may be a promising avenue for future analyses of HiFi data.

Similarly, we do not take special measures to filter contigs with more gradual coverage drops like those in CAMP or BACT1. In general, mutations called in low-coverage regions in a contig should be subject to scrutiny; calling p-mutations while using a higher $minAltPos$ value may improve the reliability of the called mutations (see also Appendix B.9). This is true even in contigs with coverage drops, since using higher values of $minAltPos$ implictly increases the coverage needed to call a p-mutation; however, such a measure will necessarily result in identifying fewer mutations in low-coverage regions. Calling few mutations in low-coverage regions of a contig will have the unpleasant side effect of limiting our ability to phase this contig, and may result in the mis-identification of coldspots. To an extent, these problems are inherent to the dataset under consideration, although they may be possible to circumvent through more

sophisticated mutation identification tools.

B.4 Demonstrating strainFlye on the ChickenGut dataset

To show that strainFlye can be applied easily to other datasets besides SheepGut—and, in addition, to the output of other assemblers besides metaFlye [97]—we applied it to a HiFi read-set (referred to as "ChickenGut") from a chicken gut metagenome that was assembled using hifiasm-meta [48].

Input reads and hifiasm-meta assembly.

We downloaded the reads in this dataset in SRA format (section 3.5, "Methods"), and used the fastq-dump tool to create a FASTQ file of these reads.

We began our analysis of this dataset using the hifiasm-meta assembly generated in [48]. Since hifiasm-meta's output includes multiple assembly graph files, we used the contigs in the assembly graph labelled `chicken.hifiasm-meta.p_ctg.gfa.gz`, based on the advice given in https://github.com/xfengnefx/hifiasm-meta/issues/10. This assembly graph includes 17,073 contigs.

Alignment and naïve p-mutation identification.

Using strainFlye, we converted the assembly graph to a FASTA file of contigs (`strainFlye utils gfa-to-fasta`) and then aligned reads to these contigs (`strainFlye align`; see Appendix B.1).

We then naïvely called p-mutations in these contigs, as described in section 3.3.2, "Computing mutation spectra", using `strainFlye call p-mutation` with $p = 1\%$. This value of 1% serves as the minimum value of p used in drawing the FDR curves below; as we will explain shortly, the coverage of MAGs in ChickenGut is generally lower than that in SheepGut, limiting our ability to call p-mutations with frequency rarer than 1%.

Decoy contig selection and FDR estimation.

In order to visualize FDR curves of mutation identifications in this dataset's contigs, we performed FDR estimation using `strainFlye fdr estimate`. We provided a file of diversity indices produced by `strainFlye call p-mutation` as input to the FDR estimation step, in order to allow the step to automatically select a suitable decoy contig (section 3.5, "Methods").

Since the average coverage of contigs in ChickenGut is generally lower than that in SheepGut, no contigs with lengths $\geq minLength = 1$ Mbp and average coverage $\geq minCov =$ 1,000x exist in this dataset (section 3.3.1, "Demonstrating strainFlye"). In this situation, strainFlye raises an error explaining the situation to the user (section 3.5, "Methods"): here we discuss how we adjusted the parameters of decoy contig selection accordingly.

We can identify potential decoy contigs either by lowering our thresholds for minimum length, or for minimum average coverage. In total, 148 / 17,073 = 0.87% of contigs in ChickenGut have lengths $\geq minLength = 1$ Mbp; we thus focused on lowering our minimum average coverage threshold. These 148 contigs have average coverages ranging from 4x to 304x. Six of these contigs have average coverages of at least 200x; ten have average coverages of at least 150x; and 18 have average coverages of at least 100x. Although there is some ambiguity in what exactly to set the minimum average coverage as, we lowered $minCov$ to 100x. This resulted in the automatic selection of contig `s92.ctg000105c` (length 1.90 Mbp, average coverage 293x) as the decoy.

The selection of a decoy contig implies FDR estimates for the naïve p-mutation identifications of all other contigs in the dataset, for various values of p. Using the default $highFrequency = 5\%$ parameter to define a p-mutation as indisputable (section 3.5, "Methods"), we can draw FDR curves for the "rare" values of $p \in [1.00\%, 4.99\%]$.

Visualizing FDR curves.

We present a visualization of eight high-diversity-index target contigs' FDR curves for p-mutation identification in Figure B.6. This is analogous to the FDR curve visualization for SheepGut shown in Figure 3.2.

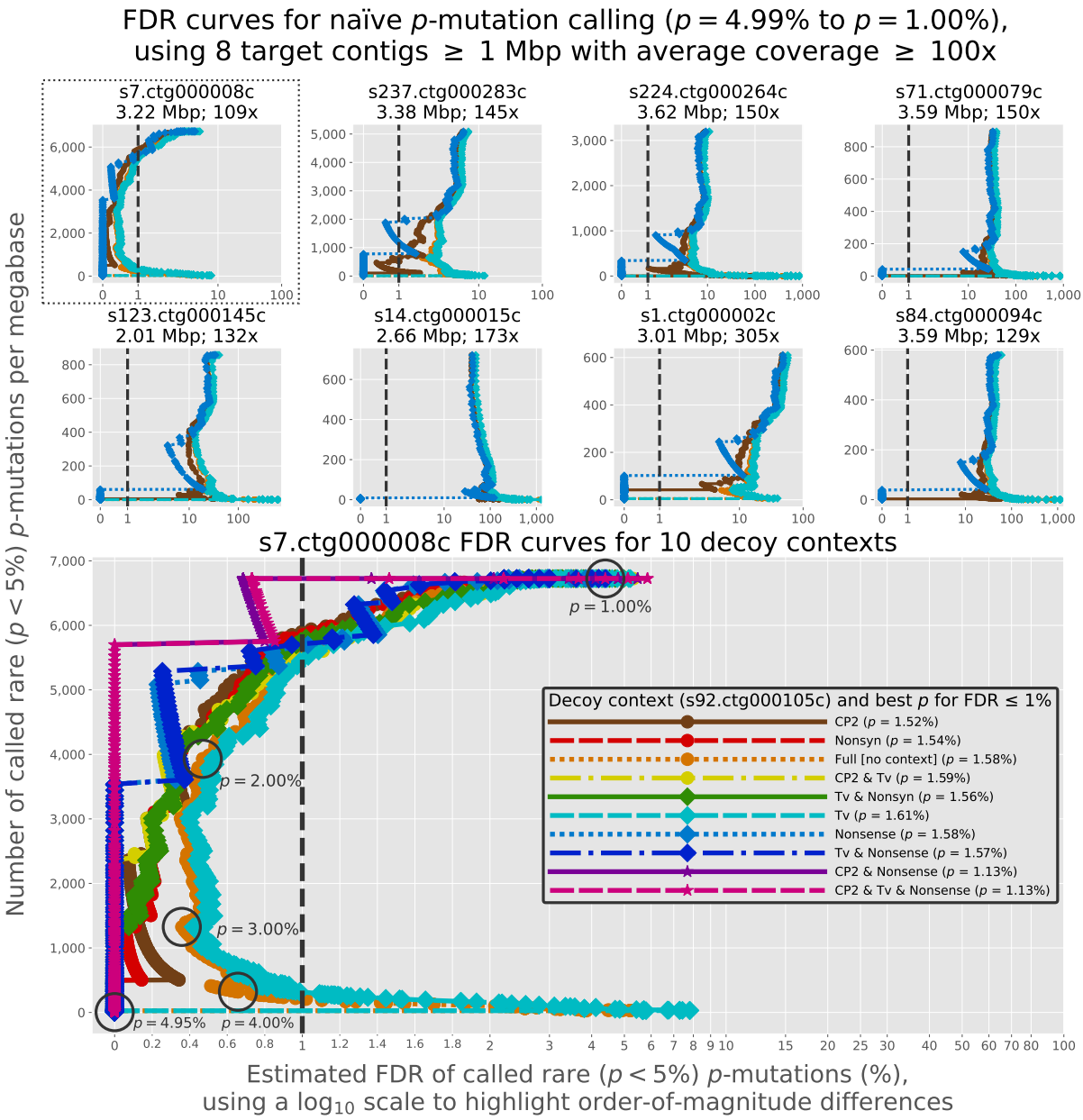
For Figure 3.2, we selected target contigs for which to draw FDR curves by considering contigs with high diversity indices (given $p = 0.5\%$). Here, we took a different approach, and instead chose contigs with high numbers of p-mutations per megabase at $p = 1\%$. This is because the default $minReadNumber = 5$ parameter we used in determining "sufficient coverage" when computing the diversity indices (Appendix B.11) causes many of the diversity indices computed for small values of p to be undefined for the long contigs in ChickenGut, due to these contigs' generally lower coverages. This problem could be circumvented by re-computing diversity indices using a smaller $minReadNumber$ value (we note that this may also result in the selection of a different decoy contig), but for the sake of time we have taken this other approach.

Interestingly, in the SheepGut FDR curve (Figure 3.2), the "Full" decoy context yielded relatively low FDR estimates for many target contigs (at many values of p), and the "CP2 & Tv & Nonsense" decoy context yielded relatively high FDR estimates. In Figure B.6 the situation is reversed; this may be a consequence of the lower coverage in ChickenGut resulting in fewer observed cases of these relatively rare types of mutations.

If we use the (relatively conservative) "CP2" decoy context, the best value of p for target contig `s7.ctg000008c` yielding an estimated FDR $\leq 1\%$ is $p = 1.52\%$. At this threshold, we identify 5,899 rare mutations per megabase in this target contig (19,022 rare mutations total), in addition to 503 "indisputable" mutations (p-mutations at $p = highFrequency = 5\%$). This demonstrates that a large amount of diversity lurks within this dataset.

This brief analysis illustrates the applicability of strainFlye to additional HiFi metagenomic datasets: we have shown that our approach for FDR estimation can be

Figure B.6. FDR curves for eight target contigs in ChickenGut. We use `s92.ctg000105c` as our decoy contig, and chose the target contigs for this figure as the eight contigs (meeting our length and coverage thresholds) with the largest number of p-mutations per megabase at $p = 1\%$. We draw a larger plot for the target contig `s7.ctg000008c`; we circle certain values of p on the "Full" decoy context curve in this plot for clarity. We note that the FDR curves drawn for `s7.ctg000008c` begin at $p = 4.95\%$ rather than $p = 4.99\%$, because the rare mutation rate of this contig is zero for larger values of p—this causes the estimated FDR for this target contig to be undefined. The colors and other visual representations used here for each decoy context (and combinations thereof) match those used in Figure 3.2. In general, the coverages of the "long" contigs in ChickenGut are much smaller than those in SheepGut, resulting in the identification of fewer rare mutations in target and decoy contigs alike.


FDR curves for naïve p-mutation calling (p = 4.99% to p = 1.00%),
using 8 target contigs ≥ 1 Mbp with average coverage ≥ 100x
s7.ctg000008c
3.22 Mbp; 109x
s237.ctg000283c
3.38 Mbp; 145x
s224.ctg000264c
3.62 Mbp; 150x
s71.ctg000079c
3.59 Mbp; 150x
s123.ctg000145c
2.01 Mbp; 132x
s14.ctg000015c
2.66 Mbp; 173x
s1.ctg000002c
3.01 Mbp; 305x
s84.ctg000094c
3.59 Mbp; 129x
s7.ctg000008c FDR curves for 10 decoy contexts
Number of called rare (p < 5%) p-mutations per megabase
p = 1.00%
p = 2.00%
p = 3.00%
p = 4.95%
p = 4.00%
Decoy context (s92.ctg000105c) and best p for FDR ≤ 1%
CP2 (p = 1.52%)
Nonsyn (p = 1.54%)
Full [no context] (p = 1.58%)
CP2 & Tv (p = 1.59%)
Tv & Nonsyn (p = 1.56%)
Tv (p = 1.61%)
Nonsense (p = 1.58%)
Tv & Nonsense (p = 1.57%)
CP2 & Nonsense (p = 1.13%)
CP2 & Tv & Nonsense (p = 1.13%)
Estimated FDR of called rare (p < 5%) p-mutations (%),
using a log10 scale to highlight order-of-magnitude differences

adjusted to work for datasets assembled by tools other than metaFlye, and with lower coverage than SheepGut.

Identifying deletion-rich positions in ChickenGut.

In addition to drawing FDR curves for ChickenGut, we also analyze its alignment in Appendix B.3 in order to show that the pattern of "deletion-rich positions" occurs in this dataset as well as in SheepGut.

B.5 Applying LoFreq to the SheepGut dataset

Running LoFreq.

To compare LoFreq's calls with those of NaiveFreq, we ran LoFreq (version 2.1.3.1, installed using bioconda) on the three selected MAGs. Since LoFreq-NQ did not seem available in this version of LoFreq, we ran `lofreq call` with default parameters.

Running LoFreq on these three MAGs took 474,054.00 seconds (over 5 days and 11 hours); however, we note that there have been recent improvements to LoFreq which, using approximation methods, should decrease this runtime for high-coverage datasets like SheepGut [92].

Throughout this Appendix (and in Figures 3.3 and B.9), we treat positions where LoFreq called multiple single-nucleotide variants (i.e. with distinct alternate nucleotides) as positions with a single $freq(pos)$ value, defining $freq(pos)$ based solely on the second-most-common nucleotide aligned to a position (section 3.3.2, "Computing mutation spectra"). This simplifies the process of comparing LoFreq and NaiveFreq's results, since NaiveFreq only calls at most one mutation per position. For reference, there exist 0 / 66 / 1 positions at which LoFreq called multiple variants in CAMP / BACT1 / BACT2, respectively; since these account for a small fraction of the total variants LoFreq called in these MAGs (Figure 3.3), ignoring them should not make a large difference in the results shown here.

Estimating the FDR of LoFreq's calls.

Section 3.3.4, "Estimating the FDR of identified rare mutations using the TDA" demonstrates how the target-decoy approach (TDA) can be applied to estimate the FDR of a set of mutations; here we illustrate this process using LoFreq outputs. LoFreq called 22, 9,641, and 79 rare mutations in CAMP, BACT1, and BACT2, respectively. At the frequency threshold $p = 2\%$, NaiveFreq called a similar number of rare p-mutations (17, 10,520, and 100 for CAMP, BACT1, and BACT2, respectively). It turns out that the sets of rare mutations identified by LoFreq and by NaiveFreq at $p = 2\%$ are somewhat similar: the numbers of overlapping rare mutations between these groups are 15, 8,033, and 43 for CAMP, BACT1, and BACT2. This suggests that, at least for this dataset, LoFreq primarily detected rare mutations with frequency of at least 2%. Here, we describe an analysis of FDRs which suggests that there exist many more lower-frequency rare mutations.

Using LoFreq's calls, the mutation rates for each MAG are 5.7×10^{-6}, 1.5×10^{-3}, and 9.4×10^{-6} for CAMP, BACT1, and BACT2, respectively. We can estimate the FDR of LoFreq's calls for BACT2 (using CAMP as a decoy) as $\frac{5.7 \times 10^{-6}}{9.4 \times 10^{-6}} \approx 60.6\%$, a very large FDR, indicating that either most identified mutations are false or that selection of CAMP as the decoy results in a highly inflated estimate of the FDR. Although NaiveFreq's calls at the frequency threshold $p = 2\%$ result in a lower estimated FDR of $\frac{4.4 \times 10^{-6}}{1.2 \times 10^{-5}} \approx 37.0\%$ for BACT2, this is still a high FDR that raises concerns about downstream analyses such as phasing.

On the other hand, the estimated FDR of LoFreq's calls for BACT1 (still using CAMP as a decoy) is only $\frac{5.7 \times 10^{-6}}{1.5 \times 10^{-3}} \approx 0.4\%$; NaiveFreq at the frequency threshold of $p = 2\%$ has a slightly lower estimated FDR of $\frac{4.4 \times 10^{-6}}{1.6 \times 10^{-3}} \approx 0.3\%$. Although both LoFreq and NaiveFreq at $p = 2\%$ result in the reliable identification of rare mutations with low FDR, we are still interested in extending the set of identified rare mutations while controlling the FDR. For example, lowering the frequency threshold of NaiveFreq to $p = 0.5\%$ results in

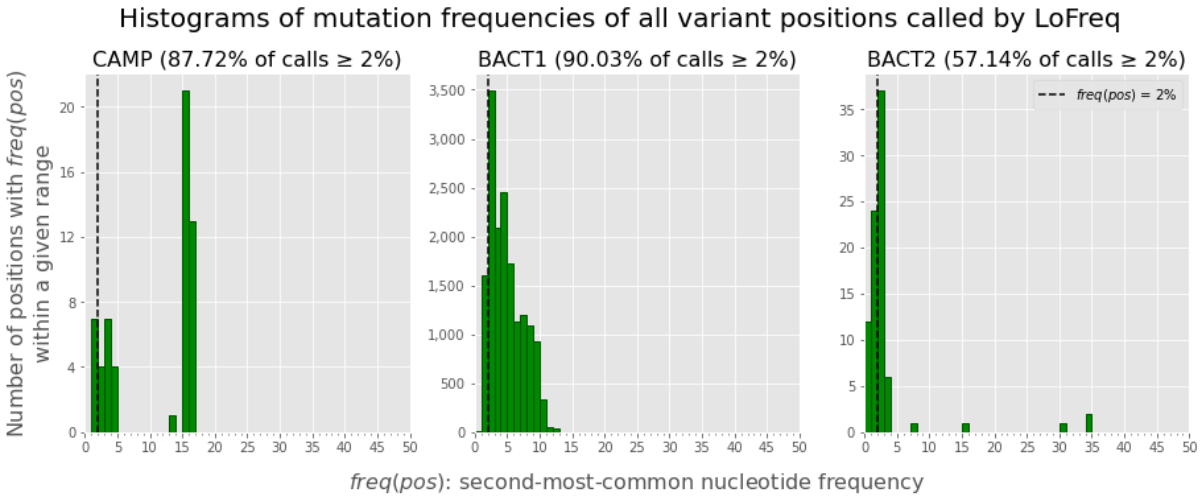


Figure B.7. Histograms of $freq(pos)$ for LoFreq's variant calls across the three selected MAGs. These histograms demonstrate that, at least in the context of these MAGs in this dataset, LoFreq seems to mainly call variants with frequency of at least 2%. The titles of each histogram include the percentage of variants with $freq(pos) \geq 2\%$, and each histogram includes a dashed line showing 2%.

the identification of 17,069 rare mutations in BACT1 (an additional 6,549 rare mutations as compared to $p = 2\%$) with a higher but still relatively low FDR estimate of 2.4%.

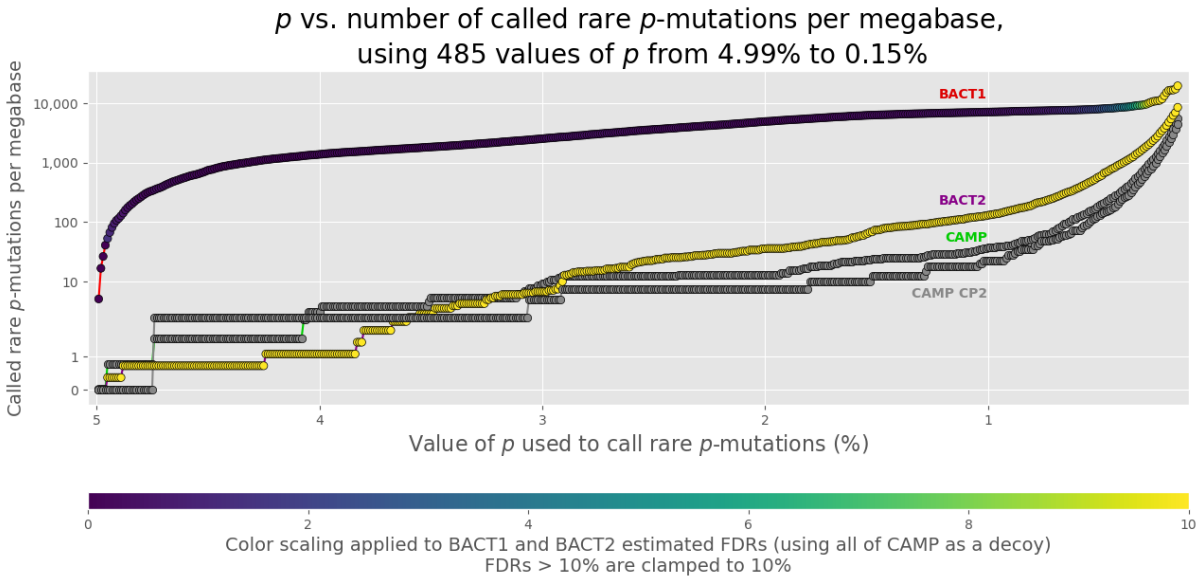
Visualizing mutation frequencies of LoFreq's calls.

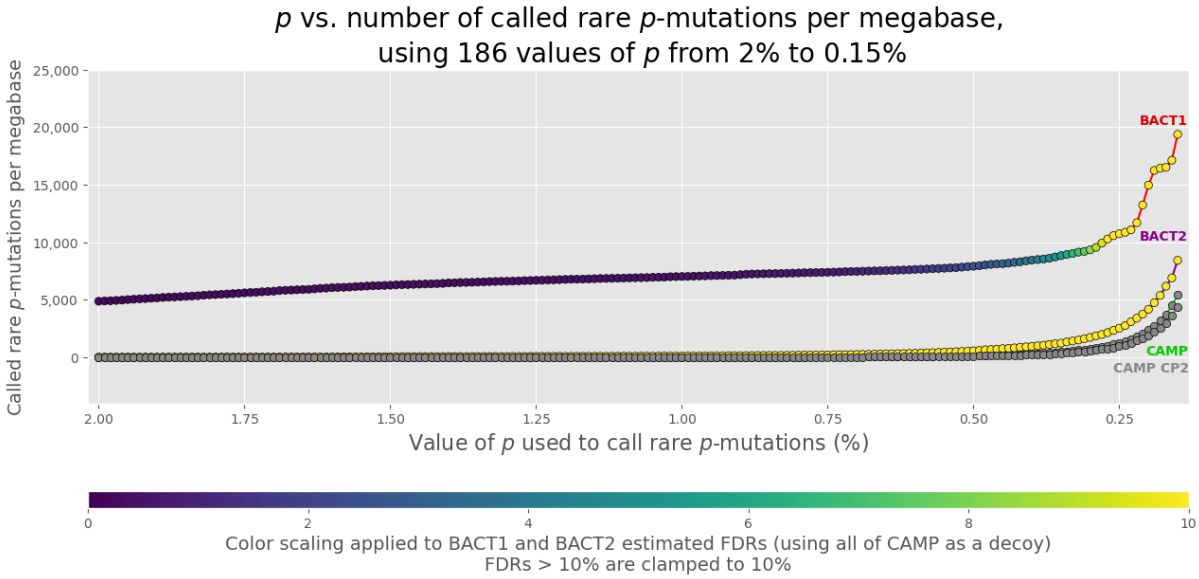
The above results, in addition to Figure 3.3, demonstrate that the variants called by LoFreq are similar to those produced by NaiveFreq at $p = 2\%$. To provide another perspective on this similarity, Figure B.7 shows histograms of $freq(pos)$ for all variants called by LoFreq across the three selected MAGs.

B.6 Growth of the number of p-mutations per megabase as p decreases

Figure 3.2 demonstrates how we can vary p to produce FDR curves for various contigs. To provide additional context to this analysis, Figure B.8 demonstrates how decreasing p monotonically increases the number of (rare) p-mutations identified in the three selected MAGs.

Figure B.8. Decreasing p (x-axis) increases the number of identified rare p-mutations per megabase in the three selected MAGs (y-axis). **(Top)** Visualization of this relationship for 485 values of p, using a logarithmic scale on the y-axis. **(Bottom)** Visualization for a subset of these values of p, using a linear scale on the y-axis. Since BACT1 has the lowest average coverage of 1,415x among the three selected MAGs (Appendix B.3), and since we mandate that NaiveFreq only calls mutations at positions with $alt(pos) > 1$, both plots use a minimum p of $\frac{2}{1{,}415} \approx 0.15\%$; this matches the minimum p used in Figure 3.2. This plot also includes a curve representing a "decoy contig" composed of just the CP2 positions in CAMP located in a single predicted gene. BACT1 and BACT2's curves are colored based on the estimated FDR for each value of p, using all of CAMP as a decoy contig. We limit FDR color variation to within the range $[0\%, 10\%]$ in order to emphasize small differences in FDRs. The gray dots in BACT2's curve in the bottom figure indicate values of p for which no rare mutations were called, which prevented computation of the FDR due to division by zero.


p vs. number of called rare p-mutations per megabase,
using 485 values of p from 4.99% to 0.15%
Called rare p-mutations per megabase
10,000
1,000
100
10
1
0
BACT1
BACT2
CAMP
CAMP CP2
5
4
3
2
1
Value of p used to call rare p-mutations (%)
0
2
4
6
8
10
Color scaling applied to BACT1 and BACT2 estimated FDRs (using all of CAMP as a decoy)
FDRs > 10% are clamped to 10%


p vs. number of called rare p-mutations per megabase,
using 186 values of p from 2% to 0.15%
Called rare p-mutations per megabase
25,000
20,000
15,000
10,000
5,000
0
BACT1
BACT2
CAMP
CAMP CP2
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Value of p used to call rare p-mutations (%)
0
2
4
6
8
10
Color scaling applied to BACT1 and BACT2 estimated FDRs (using all of CAMP as a decoy)
FDRs > 10% are clamped to 10%

B.7 Codon position analysis details

Normalized codon position mutation plots.

The plots shown in Figure 3.3 are useful when identifying relative patterns of mutation rates (for example, that the number of mutations in CP2 in a MAG is less than the number of mutations in CP3); however, since the total amount of positions within protein-coding genes varies across MAGs, comparing the literal values between plots from different MAGs is challenging.

We can account for this by dividing each value (the number of single-gene mutated positions in CP1, CP2, and CP3, as well as the number of mutated positions in non-coding regions) by the total number of positions, mutated and non-mutated, fitting these criteria. This transforms the plots so that the y-axis values are now comparable: a y-axis value of 100% for a given CP indicates that all of the single-gene positions in this CP in this MAG were identified as mutations, while a value of 0% indicates that none of the positions in this CP in this MAG were identified as mutations. Figure B.9 is an analogue of Figure 3.3 adjusted in this way.

Codon position mutation plot computation details.

We also note that our plots only consider each position in a codon in isolation, so "mutations" where multiple positions in a codon change at once might inflate these numbers. For example, as shown in Figure B.12, BACT1 contains a few instances of AGA codons (coding for Arginine) being substituted for CGC (which also codes for Arginine). Although we would currently record this in the figures shown earlier as mutations at both CP1 and CP3, we may instead prefer to discard these mutations entirely. Limiting these analyses to consider codons where the consensus mutated codon differs by a single nucleotide, as in the Appendix B.10, may be preferable (although these mutations should be relatively uncommon).

Although we have not explicitly accounted for the possibility that some of the

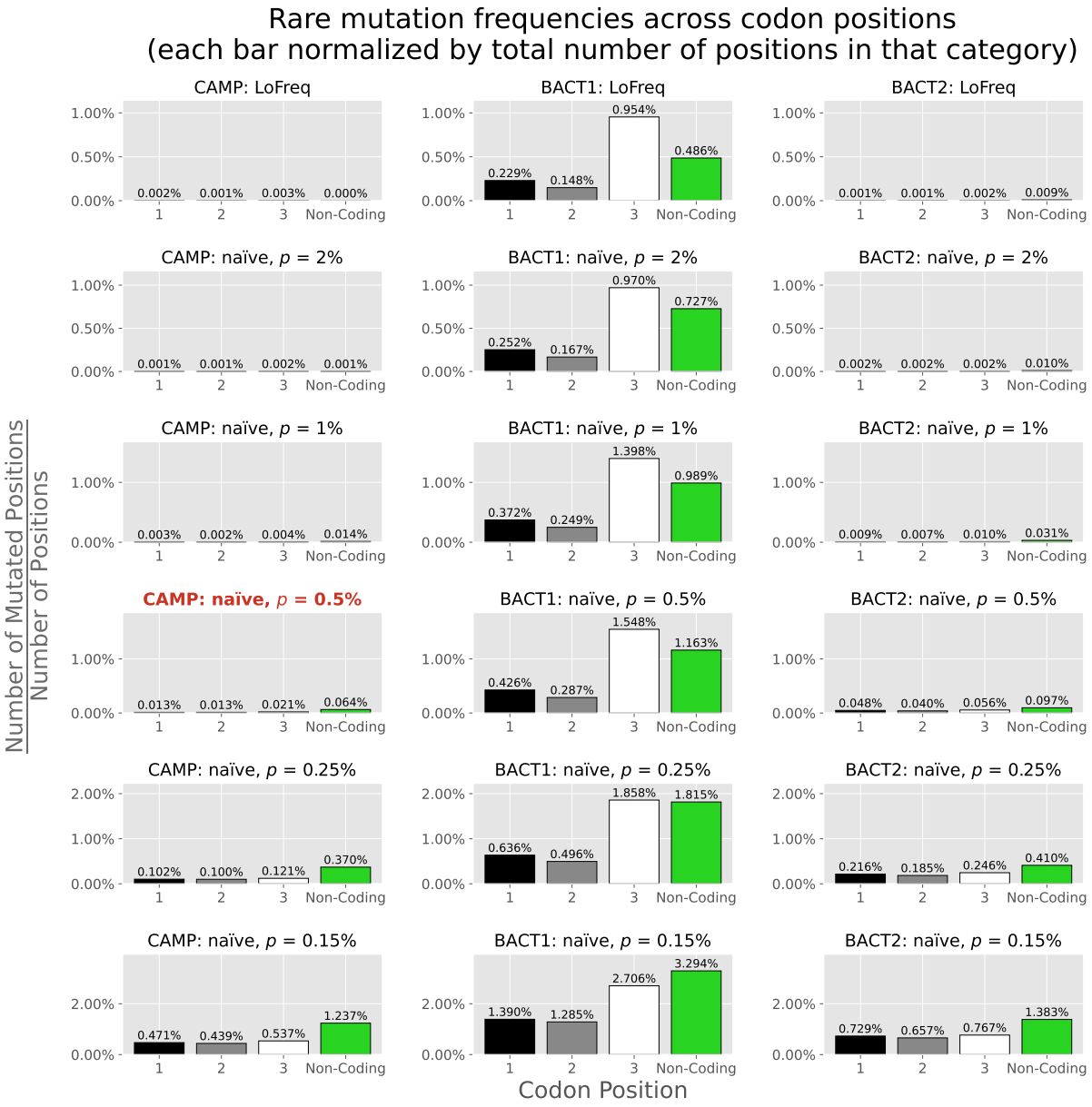


Figure B.9. Rare mutation frequencies across all first, second, and third codon positions (as well as non-coding positions) for each of the three MAGs, using the same variant calling methods as in Figure 3.3 and with y-axis values normalized by the total number of positions considered in each bar. Although the relative patterns are the same as in Figure 3.3, y-axis values are now comparable across MAGs: this makes clear that BACT1 has relatively more mutations than either of the other two MAGs.

predicted genes we consider are nonfunctional pseudogenes that do not currently experience selective pressure [7], our clear recapitulation of these expected patterns demonstrates that these trends hold in practice.

B.8 Nonsynonymous, nonsense, and transversion decoy contexts

From a decoy of positions to a decoy of mutations.

We consider all positions occurring within a predicted protein-coding gene (in CP1/CP2/CP3), ignoring positions that are located within multiple predicted genes due to gene overlap. An arbitrary position i has three possible mutations into another nucleotide, based on this position's CP (three possibilities) and its parent codon (64 possibilities). We note this paper assumes use of the standard genetic code.

For an arbitrary position i, S_i of these three mutations are synonymous and N_i of these three mutations are nonsynonymous. For example, CP3 in the Lysine-coding codon AAA has $S_i = 1$ (AAG still codes for Lysine) and $N_i = 2$ (AAT and AAC both code for Asparagine).

Similarly, if we limit our focus to positions i within the 61 sense codons (ignoring the three stop codons TAA, TAG, and TGA), we can define the values NNS_i and NS_i, corresponding respectively to non-nonsense and nonsense mutations. For example, CP1 in AAA has $NNS_i = 2$ (neither CAA nor GAA are stop codons) and $NS_i = 1$ (TAA is a stop codon).

Analogously to our computation of mutation rates for more traditional decoy contigs (of either an entire MAG, or just the CP2 positions within a MAG), we can compute a mutation rate for the decoy contig of possible nonsynonymous or nonsense mutations within a MAG.

Computing (non)synonymous mutation rates.

We define the rate of nonsynonymous mutations as $R_N = \frac{M_N}{\sum_i N_i}$, where M_N is the total number of mutations observed in the decoy contig that are nonsynonymous with respect to the position's parent codon and $\sum_i N_i$ is the sum of N_i across all candidate mutation positions i located within the decoy contig. R_N can be used in place of $rate_{decoy}$ when estimating the FDR of identified mutations.

For comparison, we can compute a corresponding rate for synonymous mutations as $R_S = \frac{M_S}{\sum_i S_i}$. Like R_N, R_S is interpretable as a ratio of observed to possible mutations. Figure B.10 shows plots of R_S versus R_N, illustrating that generally $R_S > R_N$.

We note that the R_S and R_N values bear resemblance to the commonly used K_a and K_s values [79]; in computing R_S and R_N, we have attempted to adapt these ideas to the aggregate analysis of different strains of a MAG, rather than the analysis of distinct homologous genes.

Computing (non)sense mutation rates.

We define R_{NNS} and R_{NS} as ratios of observed to possible non-nonsense and nonsense single-nucleotide mutations, computed analogously to R_S and R_N (albeit limited to positions located within the 61 sense codons). These values can be interpreted similarly to R_S and R_N, and are also visualized in Figure B.10. We expect to see more non-nonsense than nonsense codon mutations in real sequencing data, since nonsense mutations are usually harmful: and this is generally the case, although this pattern weakens as we reduce p.

Computing transversion mutation rates.

As above, consider an arbitrary position i in a contig (not necessarily located within a predicted gene). The nucleotide located at i has three possible mutations into another nucleotide: of these three possible mutations, two are *transversions* (mutations from a purine into a pyrimidine, or vice versa) and one is a *transition* (a mutation from

one purine into another, or from one pyrimidine into another) [204]. Although there are twice as many possible transversion mutations as transition mutations, transition mutations generally occur more often than transversions [204, 94], to the point that variant callers' outputs are often evaluated using the ratio of transitions to transversions [206, 209]. Transversion mutations are thus an attractive option for constructing a decoy context, similar to nonsynonymous or nonsense mutations. We can compute the ratio of observed to possible transversion mutations analogously to how we compute R_N and R_{NS} above.

Combining decoy contexts.

Figure 3.2 demonstrates decoy contexts using CP2, nonsynonymous, nonsense, and transversion mutations, as well as various combinations of these contexts. In total, this figure shows ten distinct FDR curves: here we provide some details about the combinations of decoy contexts that these curves represent.

These combinations can be thought of as intersections. The decoy context of "CP2" and "Transversion," for example, will consider only possible transversion mutations located within CP2 positions. So, although the decoy context of just transversion mutations considers all positions in the decoy contig, the combination of "CP2" and "Transversion" implies focusing solely on positions located within CP2 of a single predicted gene.

If we consider a set containing the entries {"CP2", "Transversion", "Nonsynonymous", "Nonsense"}, the resulting power set has $2^4 = 16$ subsets, or combinations of these contexts. However, not all of these subsets are distinct from each other in practice: for example, all nonsense mutations are also nonsynonymous, so we can ignore combinations that include both "Nonsynonymous" and "Nonsense."

We can also ignore {"CP2", "Transversion", "Nonsynonymous"} because it is identical to {"CP2", "Transversion"}. In the standard genetic code, there is only one possible synonymous CP2 mutation (TAA $\longleftrightarrow$ TGA, both of which are stop codons): since A $\longleftrightarrow$ G is a transition mutation, the addition of the "Nonsynonymous" context to

{"CP2", "Transversion"} does not change anything.

We are now left with eleven distinct decoy contexts and combinations thereof. The `strainFlye fdr estimate` command can produce up to eleven FDR curves, using any of these contexts. Although strainFlye supports computing it, we note that one of these eleven combinations ({"CP2", "Nonsynonymous"}) is omitted from Figure 3.2. This is because—although this decoy context is technically different from the decoy context of just "CP2"—it is very difficult to visually distinguish these two contexts on a plot, since they only differ in whether or not they count the aforementioned TAA $\longleftrightarrow$ TGA mutation.

Details in the use of nonsynonymous, nonsense, and transversion decoy contexts.

We ignore positions that are "unreasonable" (discussed in Appendix B.12), to simplify our interpretation of mutations. For example, when we identify nonsense mutations, we assume directionality in this mutation—i.e. that the mutation goes from a sense codon to a stop codon, and not the other way. The "Full" and CP2 decoy contigs do allow unreasonable positions; however, there are in total only 169 unreasonable positions across all three selected MAGs (for reference, 155 / 169 of these are in BACT1), so our exclusion or inclusion of them should not make much of a difference in our estimated FDRs. Anecdotally, these positions seem to be mostly artifacts of low-coverage regions.

When computing nonsynonymous and nonsense mutation rates, we note that our consideration of the positions within a codon in isolation ignores the possibility of multiple positions within a codon being mutated simultaneously—for example, if CP1 and CP2 of the codon AGT (coding for Serine) are mutated into the codon TCT, these mutations are ultimately synonymous since TCT still codes for Serine. However, viewed individually, neither of these two mutations would be treated as synonymous, since neither TGT (Cysteine) nor ACT (Threonine) codes for Serine. This is a limitation of our approach.

Relatedly, we do not explicitly account for alternative start codons. If a predicted gene begins with a codon like TTG or GTG, then we will treat this codon as if it coded

for Leucine or Valine, respectively.

Multiallelic positions also pose a challenge, because NaiveFreq only performs "binary" mutation calling: it only classifies a position as mutated or not, based on a single alternate nucleotide. It thus cannot call multiple mutations (into multiple alternate nucleotides) at a single position, although more sophisticated tools like LoFreq [213] can do this. This means that these mutation rates may be slightly lower than they would be if we had accounted for multiple mutations at the same position. (However, as discussed in Appendix B.5, these sorts of mutations should be relatively rare.)

Lastly, across all $61 \cdot 3 = 183$ positions i in CP1, CP2, or CP3 of the 61 sense codons, there are $\sum_i NNS_i = 526$ non-nonsense mutations and only $\sum_i NS_i = 23$ nonsense mutations [131]. We emphasize that, since the number of nonsense mutations is much smaller than the number of non-nonsense mutations, the use of nonsense mutations for FDR estimation should be done with caution. For example, although a decoy contig that happens to have no nonsense mutations at all may imply a FDR estimate of 0%, this is not necessarily a useful estimate.

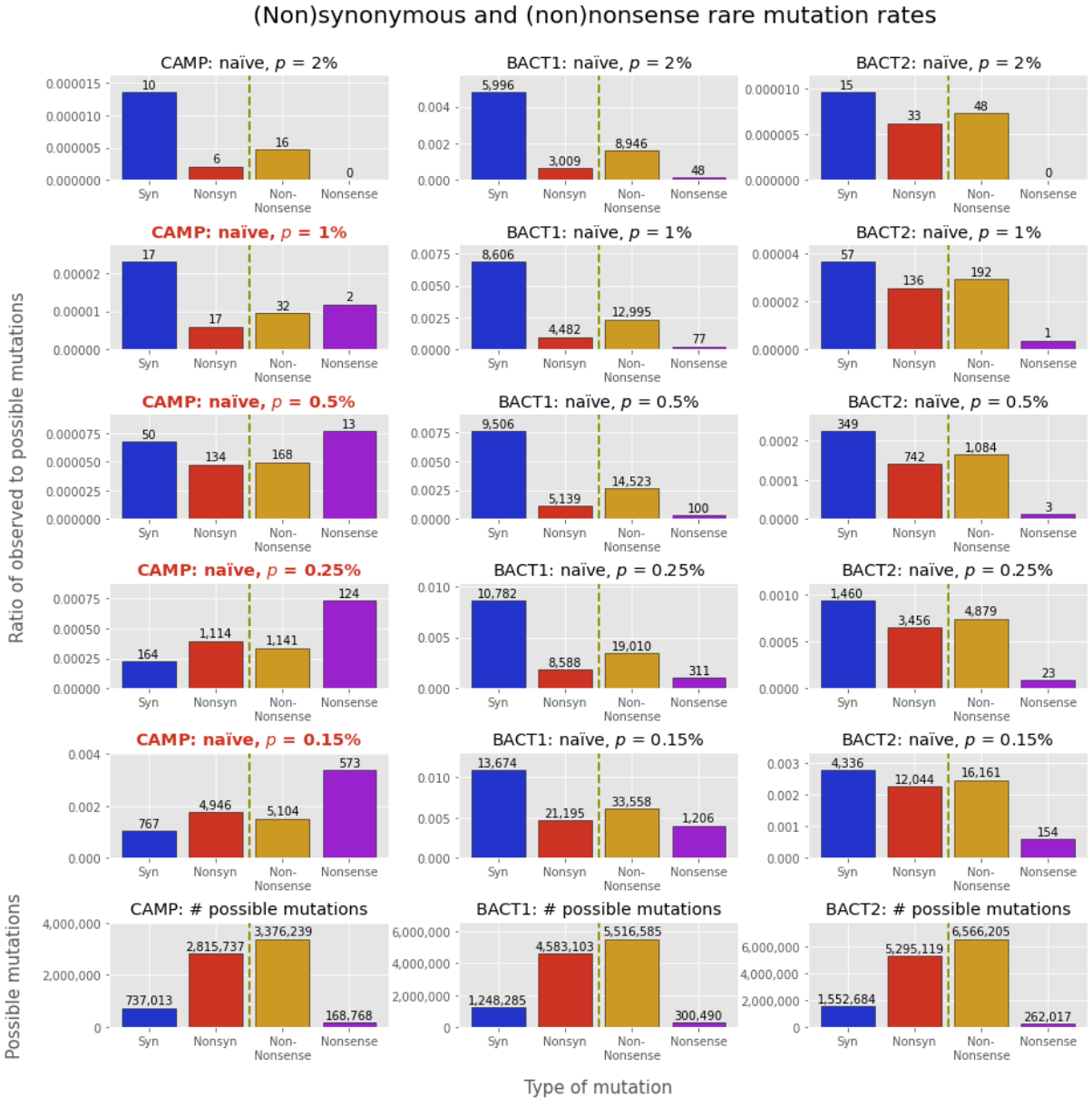
Barplots of (non)synonymous and (non)sense mutation rates.

Figure B.10 shows plots of the rates of synonymous vs. nonsynonymous p-mutations and the rates of non-nonsense vs. nonsense p-mutations as p decreases. This figure is analogous to Figure 3.3 in section 3.3.3, "The target-decoy approach for estimating the FDR of identified mutations": both figures show how expected mutational patterns mostly remain constant across the three selected MAGs as we adjust p, with some exceptions.

B.9 Identifying mutations based solely on read counts

As an alternative to p-mutation calling, given a *read count threshold* $r \geq 1$, NaiveFreq classifies *pos* as an *r-mutation* if $alt(pos) \geq r$. This is implemented in the `strainFlye call`

Figure B.10. Barplots of R_S, R_N, R_{NNS}, and R_{NS} for various values of p. Since these values are all ratios of observed to possible mutations in a MAG, these ratios increase monotonically as p decreases (causing NaiveFreq to call more p-mutations). The text above the bars for all rows above the bottom row indicates the number of called mutations of each type for each MAG at each value of p; the bottom row shows the number of possible mutations of each type for each MAG. Notably, the relative distributions of possible synonymous vs. nonsynonymous and non-nonsense vs. nonsense mutations are very similar across the three selected MAGs. The ratio of R_S to R_N, and the ratio of R_{NNS} to R_{NS}, also vary with p. Due to selection, we would generally expect $R_S > R_N$ (i.e. relatively more synonymous than nonsynonymous mutations), and we would also expect $R_{NNS} > R_{NS}$ (more non-nonsense than nonsense mutations). The titles of plots where either of these expected patterns do not hold are highlighted in red, like in Figure 3.3. This is the case for CAMP for $p \in \{1\%, 0.5\%, 0.25\%, 0.15\%\}$.


(Non)synonymous and (non)nonsense rare mutation rates
CAMP: naïve, p = 2%
BACT1: naïve, p = 2%
BACT2: naïve, p = 2%
CAMP: naïve, p = 1%
BACT1: naïve, p = 1%
BACT2: naïve, p = 1%
CAMP: naïve, p = 0.5%
BACT1: naïve, p = 0.5%
BACT2: naïve, p = 0.5%
CAMP: naïve, p = 0.25%
BACT1: naïve, p = 0.25%
BACT2: naïve, p = 0.25%
CAMP: naïve, p = 0.15%
BACT1: naïve, p = 0.15%
BACT2: naïve, p = 0.15%
CAMP: # possible mutations
BACT1: # possible mutations
BACT2: # possible mutations
Ratio of observed to possible mutations
Possible mutations
Type of mutation
Syn
Nonsyn
Non-Nonsense
Nonsense
10
6
16
0
5,996
3,009
8,946
48
15
33
48
0
17
17
32
2
8,606
4,482
12,995
77
57
136
192
1
50
134
168
13
9,506
5,139
14,523
100
349
742
1,084
3
164
1,114
1,141
124
10,782
8,588
19,010
311
1,460
3,456
4,879
23
767
4,946
5,104
573
13,674
21,195
33,558
1,206
4,336
12,044
16,161
154
737,013
2,815,737
3,376,239
168,768
1,248,285
4,583,103
5,516,585
300,490
1,552,684
5,295,119
6,566,205
262,017

`r-mutation` command, which functions analogously to the `strainFlye call p-mutation` command discussed in the main text.

It is sometimes simpler to think in terms of counts than in frequencies, and in this sense r-mutations can have some use. The set of all r-mutations for a large value of r such as $r = 100$ provides, for example, an easy-to-interpret set of usually indisputable mutations (albeit one that will almost certainly miss many real rarer mutations).

Though r-mutation identification can be useful in certain circumstances, we generally caution against using r-mutations in the context of our FDR estimation methods. r-mutation identification does not explicitly account for coverage: contigs with higher coverages can accumulate many more r-mutations at a given r simply by chance. (For example, if a contig has coverage 1,000,000x, then its set of r-mutations at $r = 100$ becomes much less convincing.) p-mutations, while not perfect, adjust the "burden of proof" needed to call a p-mutation at a position based on this position's coverage—this mitigates the problem somewhat.

B.10 Constructing and visualizing mutation matrices

Here we describe the construction and visualization of codon and amino acid mutation matrices for a given MAG, as discussed in section 3.3.6, "Codon and amino acid mutation matrices". We provide the `strainFlye matrix` module for the construction of these matrices. Similarly to [201], we define a given codon as mutated or not based on considering the frequencies of 3-mers that span this codon in the alignment: this is analogous to the process of classifying individual positions in a MAG as p-mutated or not, as discussed in section 3.3.2, "Computing mutation spectra".

Identifying codon p-mutations.

Given a threshold $p \in (0\%, 50\%]$, we classify a codon as p-mutated analogously to how NaiveFreq classifies a position as p-mutated: in the case of a codon, we consider all three-nucleotide sequences aligned to this codon's three positions in a MAG.

We consider each codon within each predicted gene within a MAG, allowing the consideration of the same position(s) multiple times if they are present in overlapping genes. (Although we considered limiting these matrices to codons only present in a single gene, we elected not to do this because this might bias our analyses against the inclusion of codons located near the ends of genes—such as Stop codons.) We consider reads aligned to the MAG that cover all three positions in this codon within a single linear alignment, where every position in the read's alignment to this codon is a match or a mismatch operation. A read's alignment spells a *mutated 3-mer* in this codon if the read's sequence aligned to the codon differs from the reference codon sequence. For each codon, we compute all matched and mismatched (mutated) 3-mers aligned to the codon from covering read alignments.

After computing this information for every codon, we then ignore codons where the most common aligned 3-mer (or, in the case of a tie, any of the most common aligned 3-mers) at this codon is not exactly the same as the reference codon's 3-mer. This check allows us to ignore low-coverage or otherwise difficult-to-interpret codons where it is unclear how to define a "mutation" from this codon into another, and is analogous to how we also ignore "unreasonable" positions in other analyses in this paper. We implicitly make the assumption that these mutations occur *from* the reference codon.

For all codons that pass the above check, we then classify the codon as mutated or not similarly to how we classified positions as mutated in section 3.3.2, "Computing mutation spectra": if the maximum-frequency mutated 3-mer aligned to the codon has a relative frequency (analogous to $freq(pos)$) greater than or equal to some threshold p and an absolute frequency (analogous to $alt(pos)$) greater than 1, we define the codon as p-mutated. Unlike our use of NaiveFreq in FDR estimation, we consider both rare and

indisputable p-mutated codons.

Our exclusion of codons where the reference 3-mer is not also the most common aligned 3-mer means that, as with the definition of $freq(pos)$ in section 3.3.2, "Computing mutation spectra", the relative frequency of any p-mutated codon is constrained to the range of $(0\%, 50\%]$.

Once we identify codon mutations, we can then compute codon mutation frequencies and construct a mutation matrix.

Computing codon mutation frequencies.

Using these identified codon mutations, we compute the total number of (X, Y)-mutations (the number of times codon X mutates into a different codon Y) and the total number of X-mutations (the number of times X mutates into any of the other 63 codons, including Y). We define the *codon mutation frequency* of X into Y as the number of (X, Y)-mutations divided by the number of X mutations.

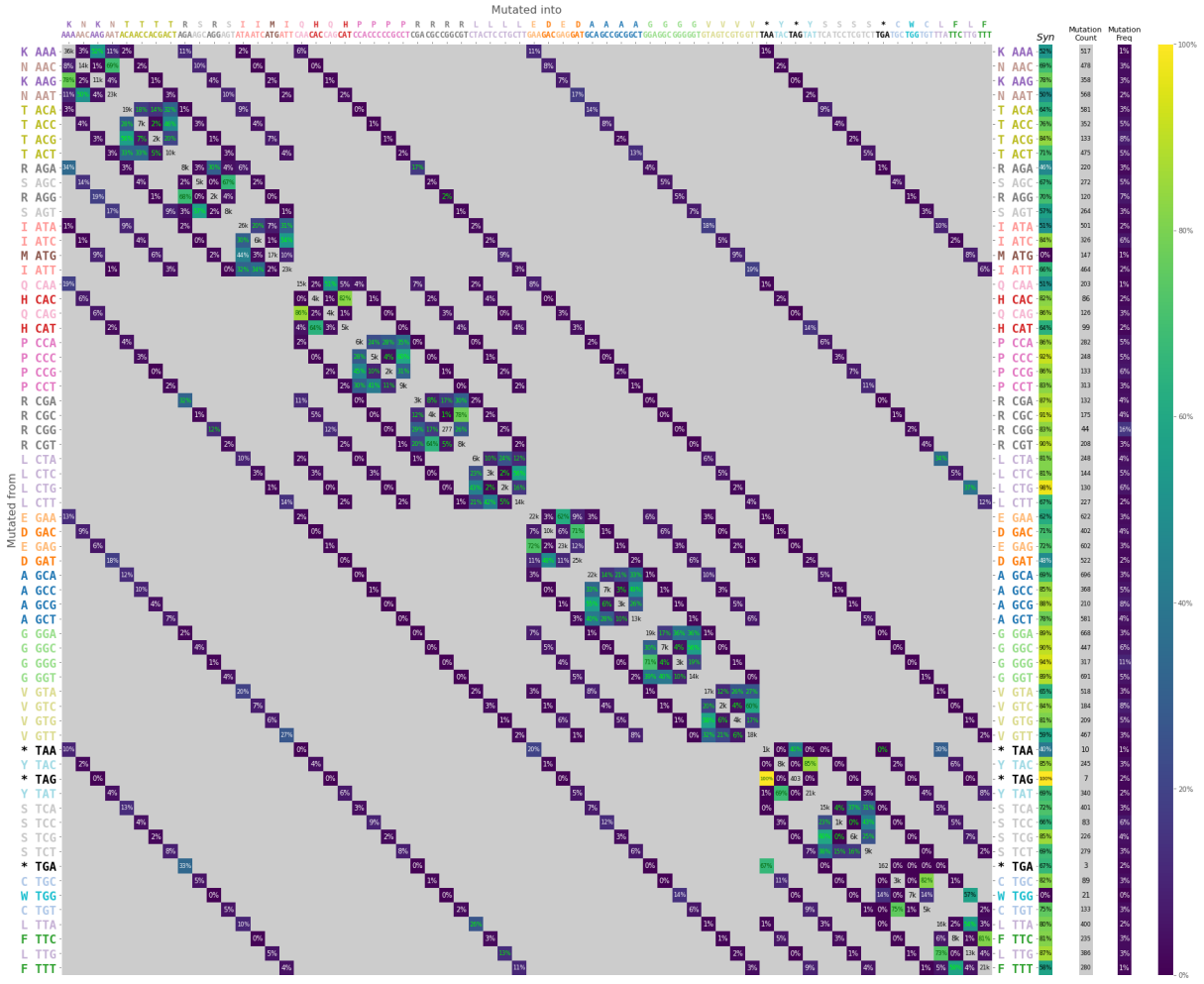
Visualizing codon mutation frequencies.

Figure B.11 is a 64x64 *codon mutation matrix* representing all single-nucleotide codon mutation frequencies throughout BACT1, using the threshold $p = 0.5\%$ to naïvely call codons as p-mutated. Figure B.11 reveals large variations in the mutation rates of various codons encoding the same amino acid, indicating that codon-based matrices may be more sensitive for strain comparison. For example, the Glycine-encoding codons GGA and GGG have mutation rates 3% and 11%, respectively. Interestingly, our analysis reveals that rare codons typically have much higher mutation rates than frequent codons for the same amino acid.

Figure B.12 shows "full" codon mutation matrices. The main differences between these matrices and the matrix shown in Figure B.11 are the inclusion of all codon mutations in each row (not just the 9 single-nucleotide mutations from a given codon); the representation of mutations in the matrix and *Syn* column as raw numbers, rather than

Figure B.11. The 64x64 single-nucleotide codon mutation matrix for BACT1 (using the frequency threshold $p = 0.5\%$). The numbers on the main diagonal show the number of times a codon occurs in all genes in the MAG, rounded to the nearest thousand (for codons that occur less than 1,000 times, the raw number is shown). Percentages off of the main diagonal show the fraction of times a given mutation from codon X into codon Y was observed, relative to the total number of single-nucleotide codon mutations from codon X in this MAG. The sum of the non-diagonal entries in each row of the matrix, then, should be approximately 100%. The text fonts representing numbers in the non-diagonal cells of the matrix are colored white or black if the mutation represented in this cell is nonsynonymous, and light green or dark green if this mutation represented in this cell is synonymous. The labels of the matrix are annotated with both the codon sequence and the corresponding amino acid / stop codon translation, abbreviated as a single letter [32]. The column labelled *Syn* shows the sums of synonymous single-nucleotide mutation percentages across each row in the matrix (these sums are also included as text within these columns). The column labelled "Mutation Count" shows the total number of mutations of this row's codon in BACT1's genes (*including both single-nucleotide and non-single-nucleotide codon mutations*). The column labelled "Mutation Freq" shows "Mutation Count" for each codon's row divided by the total number of occurrences of this codon in BACT1's genes. The gradient on the right is a legend mapping percentages to colors in the viridis colormap [179]. The shown matrix only reflects single-nucleotide codon mutations; however, a "full" version of this matrix is shown in Figure B.12.

BACT1


Mutated into
Mutated from
Syn
Mutation Count
Mutation Freq

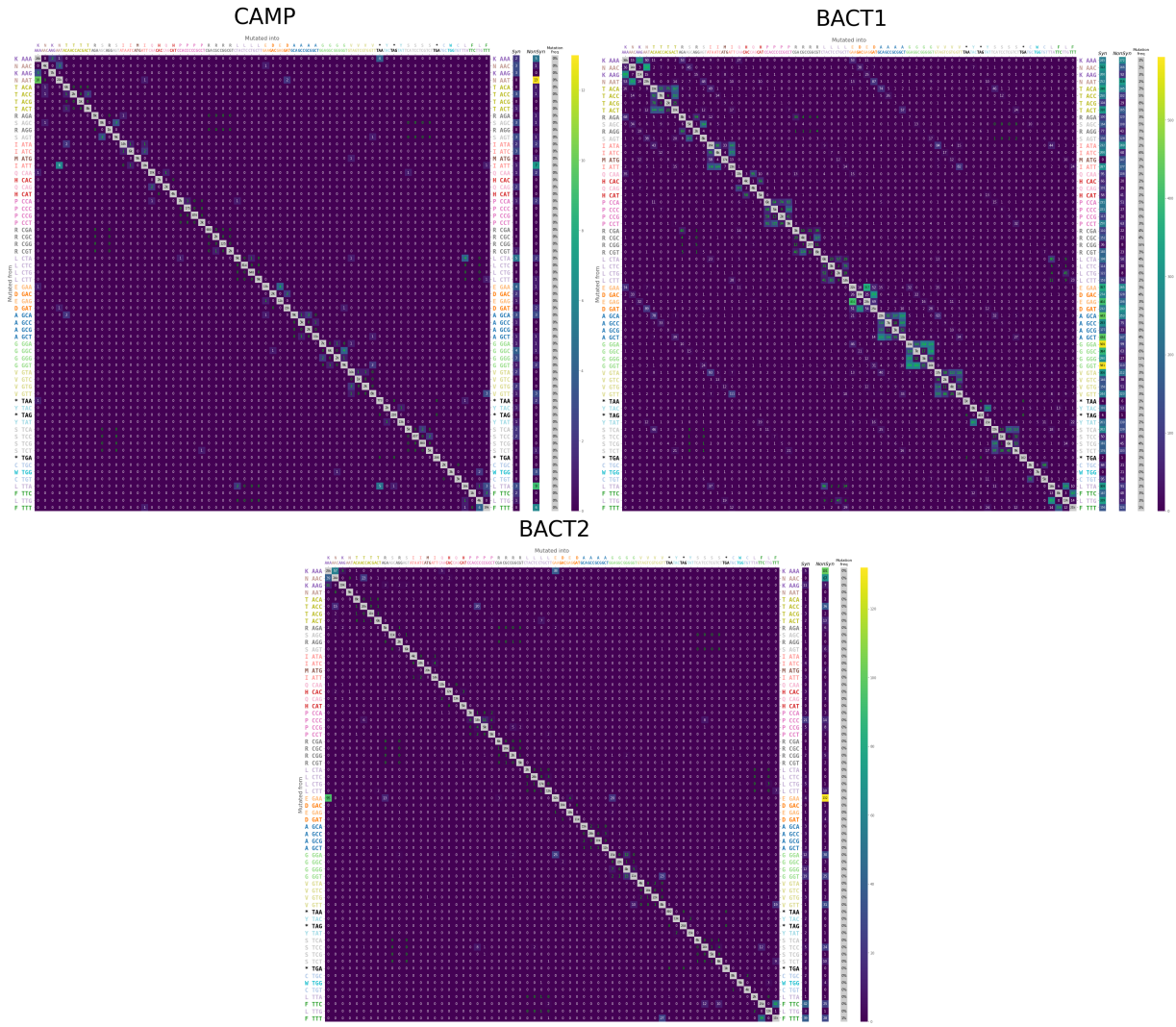


Figure B.12. "Full" codon mutation matrices for the three selected MAGs (using the frequency threshold $p = 0.5\%$).

percentages; the presence of a *NonSyn* column instead of the "Mutation Count" column; and the lack of color in the codon mutation rate column (since coloring percentages using the same scale by which the raw numbers are colored is an "apples-to-oranges" comparison).

Visualizing amino acid mutation counts.

Figure B.13 shows the 21x21 amino acid / stop codon mutation matrices for each of the three selected MAGs, derived from their corresponding codon mutation matrices

shown in Figure B.12. As with Appendix B.8, we do not account for alternative start codons.

B.11 Diversity index details

Here we provide further information on diversity indices introduced in section 3.3.7, "Diversity indices".

Defining "sufficient coverage" for p-mutation calling.

Here we assume that strains of a given organism are sequenced exhaustively, and that the alignments of all reads are correct. Although these assumptions may be broken in practice [123], they should be acceptable in the context of high-coverage HiFi datasets. Given these assumptions we would expect to see, for each strain with population frequency p, $p \cdot C$ copies of this strain's sequence at a given position in the alignment (where C is the coverage at the position). For example, a strain with population frequency $p = 25\%$ would on average be represented at 25x coverage at a given position in the alignment with coverage $C = 100$x; similarly, a strain with population frequency $p = 0.8\%$ would on average be represented at 8x coverage at a position with coverage $C = 1{,}000$x.

We thus define the minimum sufficient coverage for some p as $minSuffCov = \frac{minReadNumber}{p}$, where $minReadNumber$ (default value 5) is a small positive parameter and p is represented as a number in the range $(0, 0.5]$ rather than as a percentage. That is, if we want to find mutations with a given population frequency using solely the sequencing data at hand, without taking into account further biological insights about error rates, we need to have sufficient sequencing coverage to where we would expect to observe $minReadNumber$ copies of these mutations [191]. This relation between $minSuffCov$ and p makes clear that the burden of proof for calling a relatively "common" mutation (e.g. $p = 50\%$) is much lower than the burden of proof for calling a relatively "rare" mutation (e.g. $p = 0.5\%$).

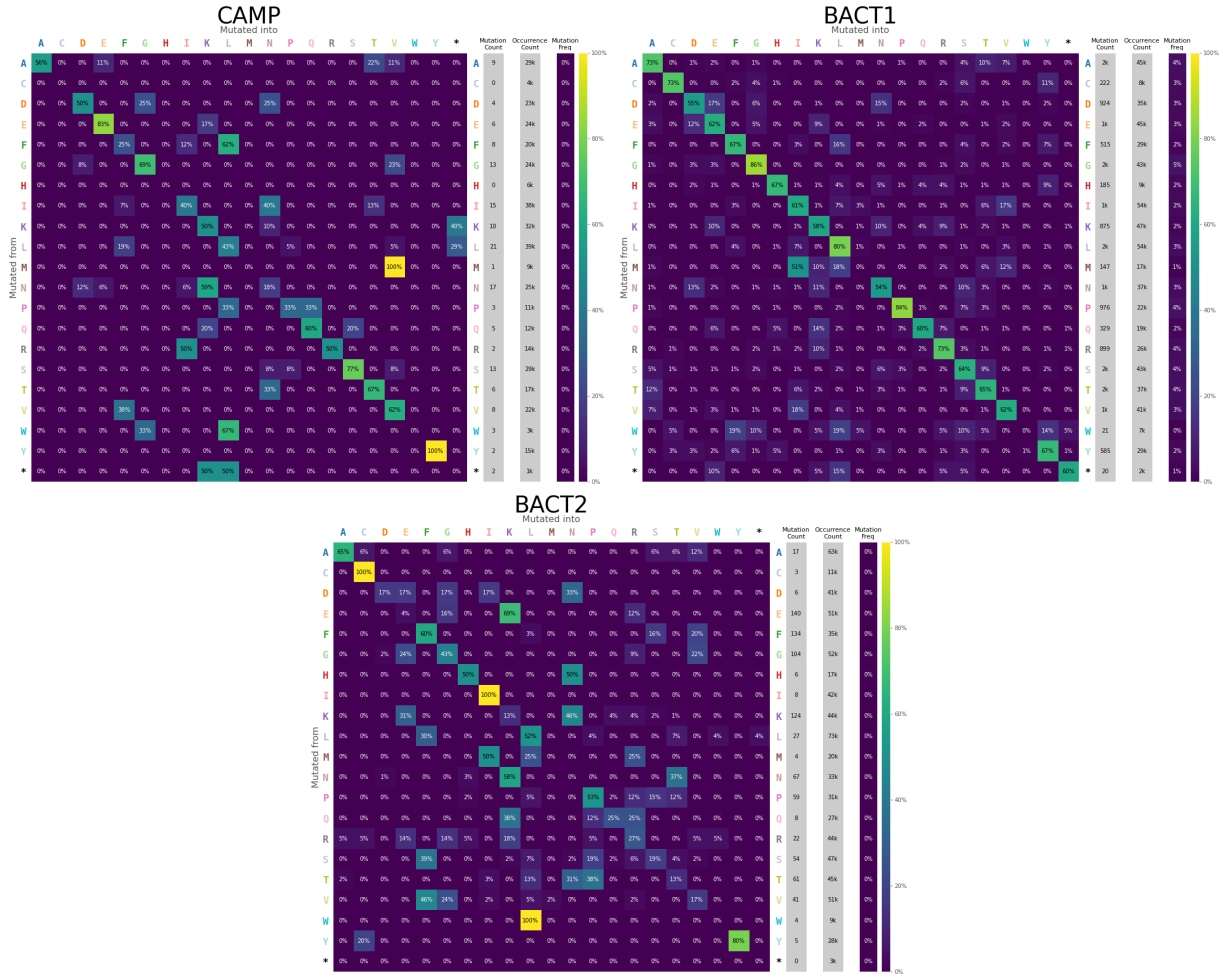


Figure B.13. The 21x21 amino acid / stop codon mutation matrices derived from codon mutation data for the three selected MAGs (using the frequency threshold $p = 0.5\%$). Percentages in each matrix show the number of times a given mutation from one amino acid / stop codon into another was observed based on the codon mutation data (including both single-nucleotide and non-single-nucleotide codon mutations), divided by the total number of mutations (both synonymous and nonsynonymous) seen from this row's amino acid / stop codon. The denominators of these percentages for each row are represented in the "Mutation Count" column. The numbers in the "Occurrence Count" columns show the number of times an amino acid / stop codon occurs in all genes in a MAG. The numbers in the "Mutation Freq" columns show "Mutation Count" divided by "Occurrence Count", analogously to the "Mutation Freq" column in Figure B.12. Numbers are colored white or black arbitrarily, in order to improve legibility on dark or light cells. Synonymous codon mutations (e.g. GCA into GCC, both of which code for Alanine) are represented as values on the main diagonals. Amino acids are referred to by their single-letter abbreviations [32], with the $*$ character (located in the final row and column of each matrix) representing stop codons. The gradients on the right are legends mapping percentages to colors in the viridis colormap [179].

Defining "sufficient coverage" for r-mutation calling.

Appendix B.9 details the identification of r-mutations, which are called using read counts rather than frequencies. There is less need to impose an additional requirement of sufficient coverage for these particular mutations.

However, knowledge of which positions are or are not sufficiently-covered has other uses—for example, as shown below, when defining entire contigs as sufficiently-covered or not. We thus define a position as sufficiently-covered to call a given r-mutation using a quantity named $minCovFactor$ (default value 2): the minimum sufficient coverage for some r is defined as $minSuffCov = minCovFactor \cdot r$. Since r corresponds to the count of the second-most-common nucleotide at a position, the default value of $minCovFactor = 2$ means that this condition will always hold if we have called an r-mutation in the first place (although it will also hold for many positions at which there exist no r-mutations).

Computing the diversity index for sufficiently-covered contigs.

We have shown above how to compute the minimum sufficient coverage for calling p- or r-mutations for arbitrary positions in a contig, when computing diversity indices. We (somewhat crudely) define a contig G itself as sufficiently-covered if at least 50% of its positions are sufficiently-covered.

We define the diversity index for a sufficiently-covered contig G as the number of called p- or r- mutations in the sufficiently-covered positions in G, divided by the total number of sufficiently-covered positions in G. If a contig is not sufficiently-covered, we do not define a diversity index for it.

Information about high-diversity-index edges.

Figure 3.4 reveals a number of edges in SheepGut with high diversity indices. Here we provide further information about these edges in order to help in the interpretation of this figure.

For reference, the most diverse edge at $p = 50\%$ is edge 3091 in the graph, which

has a length of 1 Mbp and a coverage (reported by metaFlye) of 40x. Kaiju [124] classified this edge as *Dorea formicigenerans*.

The most diverse edge at $p = 25\%$ is edge 15931, which has a length of 1.1 Mbp and a coverage of 33x. Kaiju classified this edge as an uncultured *Flavonifractor* sp.

The most diverse edge for $p \in \{10\%, 5\%, 2\%, 1\%, 0.5\%\}$ is edge 3030, which has a length of 1.9 Mbp and a coverage of 1,011x. Kaiju classified this edge in the phylum *Firmicutes*.

The most diverse edge at $p = 0.25\%$ is edge 23917, which has a length of 1 Mbp and a coverage of 1,973x. Kaiju classified this edge as *Phocaeicola vulgatus*.

Although edges 3030 and 23917 both have very high coverage, neither were selected for the analyses in the rest of this paper because these edges were located in the largest "hairball" component of the assembly graph. (Edges 3091 and 15931 are also located in the "hairball," for reference.)

B.12 Hotspot genes in the three selected MAGs

For each gene in each MAG, we can compute values for each gene describing the mutations observed for the positions in this gene. These values include the *mutation rate* as well as more specific values such as the *nonsynonymous mutation rate* (defined as the fraction of mutated positions in the gene where the highest-frequency alternate nucleotide causes a nonsynonymous mutation in the position's codon within the gene; more details are described later in this Appendix).

For convenience's sake, here we ignore positions where the reference nucleotide is not also the consensus (i.e. most frequently seen in the alignment at this position) nucleotide. Positions that do not meet this criteria, termed "unreasonable" positions, are not included as mutations when computing the mutation rates shown in Table B.2. This simplifies the computation of the nonsynonymous mutation rate, since it becomes easier to

say that a given nucleotide represents a mutation "from the reference"; this is analogous to how certain codons are ignored in the Appendix B.10.

Out of a total of 1,297 / 1,761 / 2,567 genes in CAMP / BACT1 / BACT2, respectively, 110 / 1,511 / 677 genes have nonzero p-mutation rates (using the threshold $p = 0.5\%$). Of these genes, 98 / 1,171 / 566 genes have nonzero nonsynonymous mutation rates. (The rounded average gene lengths for each MAG are 919 / 1,106 / 897 bp, respectively.) To assist in identifying specific genes of interest, Table B.2 presents information about the ten genes with the highest mutation rates for each of the selected MAGs. Given this information, we can now investigate the mutation spectra of these genes in an attempt to learn more about the strain(s) of these MAGs present in the dataset, for example as shown in Figure 3.5.

Computing the nonsynonymous mutation rate of a gene.

Table B.2 makes use of the *nonsynonymous mutation rate*, a value we defined for each gene. Here we describe some minor details of the computation of this value.

When computing this metric we consider all mutated positions from all codons in a gene. An example gene containing 99 positions—with two mutated positions, both in the same codon within the gene, and each causing a nonsynonymous mutation by themselves—would have a nonsynonymous mutation rate of $2/99 \approx 2.02\%$.

We note that our definition of a mutation as "nonsynonymous" assumes that the contig in which this mutation occurs follows the standard genetic code, and that we do not explicitly account for alternative start codons. These assumptions match those used in Appendices B.8, "Nonsynonymous, nonsense, and transversion decoy contexts" and B.10, "Constructing and visualizing mutation matrices".

A rare corner-case in this definition occurs when a given mutated position has more than one highest-frequency alternate nucleotide. For example, we could imagine CP3 of the codon CAA. This codon codes for the amino acid Glutamine. If, in CP3 of this codon,

Table B.2. The ten most mutated genes in the three selected MAGs (using the threshold $p = 0.5\%$ to classify a position as mutated or not). The "#" column shows the number assigned to each gene in the SCO files output by Prodigal for each MAG. We sort genes by their total mutation rate (the rightmost column); the mutation rates for the top ten genes in BACT1 are much higher than those from the other two MAGs.

CAMP

Left	Right	Length (bp)	#	Nonsyn. Mutation Rate	Mutation Rate
1,208,927	1,210,075	1,149	1217	0.26%	2.96%
1,053,365	1,055,938	2,574	1056	0.31%	0.97%
1,197,031	1,197,267	237	1205	0.84%	0.84%
1,287,424	1,287,777	354	1295	0.56%	0.56%
890,033	890,569	537	886	0.37%	0.56%
223,065	223,976	912	229	0.55%	0.55%
1,056,052	1,059,237	3,186	1057	0.16%	0.50%
493,154	493,558	405	477	0.49%	0.49%
1,231,190	1,231,402	213	1243	0.00%	0.47%
684,152	684,607	456	674	0.44%	0.44%

BACT1

Left	Right	Length (bp)	#	Nonsyn. Mutation Rate	Mutation Rate
1,041,656	1,042,084	429	868	16.32%	20.05%
206,606	207,304	699	184	12.73%	18.74%
1,402,353	1,402,718	366	1185	9.29%	16.94%
266,551	267,051	501	241	7.78%	15.77%
1,326,288	1,327,073	786	1118	7.00%	15.27%
724,871	725,443	573	586	10.65%	15.18%
1,458,950	1,460,104	1,155	1233	9.09%	15.06%
1,041,043	1,041,654	612	867	9.64%	14.38%
1,156,180	1,157,316	1,137	966	9.15%	14.16%
601,203	602,750	1,548	486	8.07%	13.95%

BACT2

Left	Right	Length (bp)	#	Nonsyn. Mutation Rate	Mutation Rate
2,797,869	2,798,756	888	2561	0.90%	4.84%
2,512,322	2,513,143	822	2288	0.49%	3.77%
1,696,990	1,697,364	375	1575	1.33%	2.13%
2,216,402	2,216,542	141	2030	1.42%	2.13%
2,626,301	2,626,774	474	2420	1.48%	1.90%
177,371	177,700	330	155	1.21%	1.82%
660,412	660,522	111	646	1.80%	1.80%
2,510,205	2,512,007	1,803	2286	0.50%	1.72%
2,101,831	2,102,010	180	1926	1.11%	1.67%
269,131	269,250	120	249	1.67%	1.67%

100 aligned reads have an A, 10 aligned reads have a C, 10 aligned reads have a G, and 1 aligned read has a T, then this position will be labelled as a p-mutation for $p = 0.5\%$, since $10/121 \approx 8.26\%$, which is greater than 0.5%. However, it is ambiguous whether or not this mutated position causes a nonsynonymous mutation: C and G are tied as the highest-frequency alternate nucleotide, and CAC causes a nonsynonymous mutation (coding for Histidine) but CAG does not cause a nonsynonymous mutation (coding for Glutamine).

In the case of this sort of tie, the alternate nucleotide selected for determining whether or not this mutation is nonsynonymous or not is arbitrary. However, we expect that this case should generally be rare.

Coverages of highly-mutated genes.

Figure B.14 shows the coverages of the highest-mutation-rate genes for each of the three selected MAGs. This figure demonstrates that the coverages these genes are uniformly high, and that the mutation spectra of these genes (shown in Figure 3.5 in the main text) are thus reliable.

B.13 Identifying strains in the most mutated gene of BACT1

The most mutated gene in BACT1 (gene 868 of length 429 bp) exhibits a complex pattern of mutations illustrated in Figure 3.5 (middle subfigure) with three main levels of mutated positions, at approximately 1.5%, 4.5%, and 6.5%. Below we attempted to group all 1,273 reads spanning this gene into clusters representing putative strains.

We transformed each read into a 429-dimensional binary vector, where the i-th value of a read is set to 1 if this read differs from the BACT1 reference MAG (with a mismatch or deletion/skip) at the i-th position of gene 868, and is set to 0 otherwise. We then performed k-means clustering on these 429-dimensional binary vectors for $k \in [1, 20]$

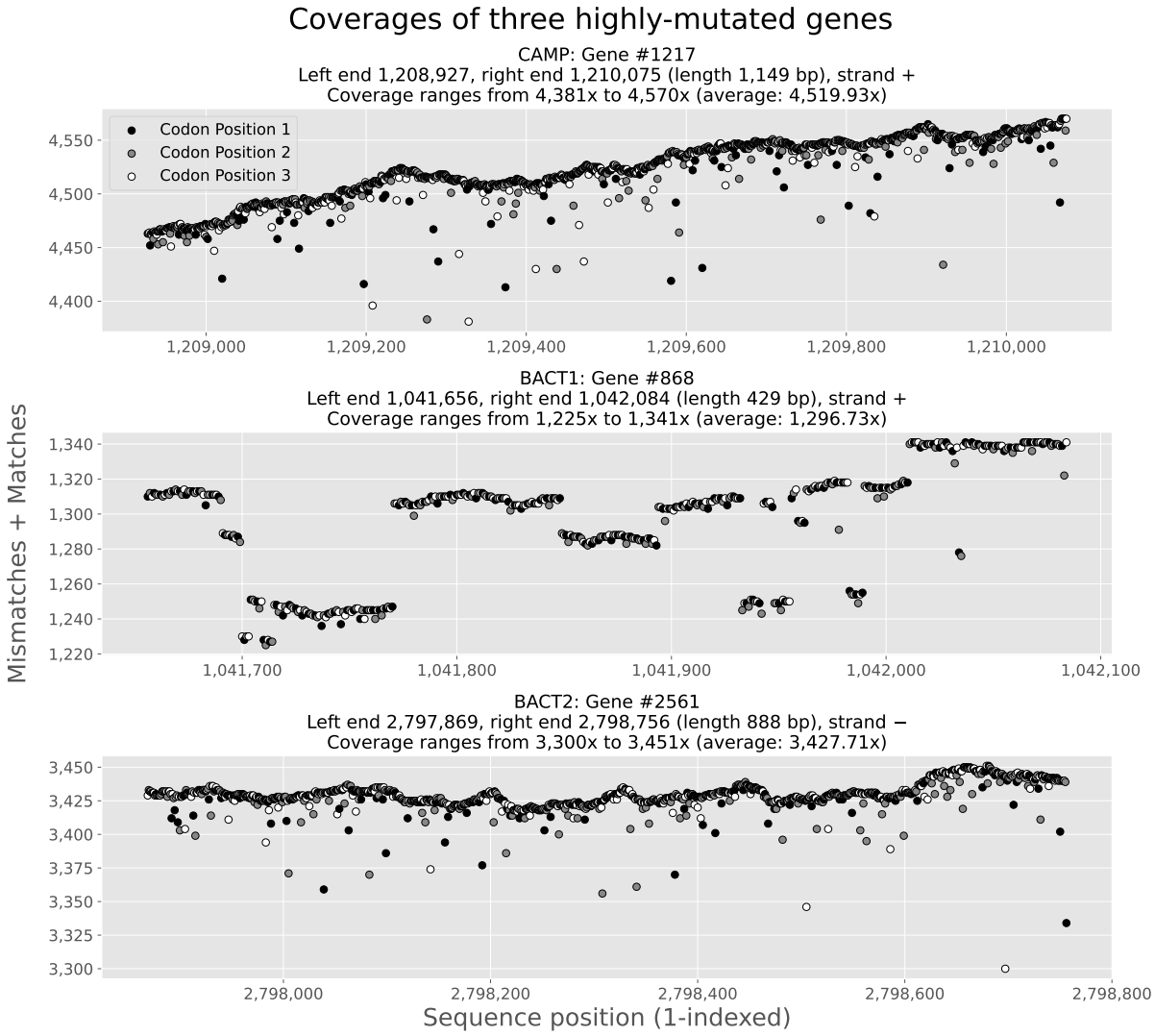


Figure B.14. Scatterplot showing $mismatches(pos) + matches(pos)$, also known as coverage, for each position *pos* within the highest-mutation-rate genes in CAMP, BACT1, and BACT2. As in Figure 3.5, positions are colored by their codon position within the parent genes shown here. This plot demonstrates that the coverages of these genes are all high ($\geq$ 1,000x) and somewhat uniform. Strangely, coverage increases slightly over the length of the highest-mutation-rate gene in CAMP; although this may just be a sequencing artifact, it may indicate that this genome is being actively replicated, since such genomes have higher coverage near the origin of replication [99]. (We thus highlight the location of this gene in Figure B.17 in the Appendix B.16.) The coverage of the highest-mutation-rate gene in BACT1 contains more dramatic "jumps," although these are primarily due to this plot not counting deletions towards coverage.

using scikit-learn [147]. A plot of the performance of each of these runs of k-means clustering, assessed using scikit-learn's "intertia" measurement, is shown in Figure B.15 (left). This plot contains a clear "elbow" at $k = 4$ after which improvements in classification begin to plateau, indicating that 4 is likely an acceptable number of clusters [69]. This corroborates the mutation spectrum of this gene in Figure 3.5 (middle), which contains the aforementioned three rough levels of mutated positions' frequencies (in addition to a fourth level of mostly un-mutated positions). In order to provide an additional view of how these reads' vectors vary, Figure B.15 (right) shows a principal component analysis plot of a matrix of the binary vectors, with points colored by their assigned cluster from the k-means clustering algorithm at $k = 4$. The majority of reads belong to the first cluster, which contains relatively "un-mutated" reads.

We note that the first draft of this analysis did not consider deletions/skips in a read at a given position in the gene to result in a 1 in that read's binary vector at this position (in these cases, the vector would contain a 0 at this position). This version of the analysis resulted in the selection of an "elbow" at $k = 3$ clusters. From manual inspection in IGV [190], the reads aligned to this gene contain many deletions; this is why we have presented this particular analysis in a way that considers deletions as mutations.

B.14 Plots of mutation locations

Figure B.16 visualizes the locations of mutations in CAMP, BACT1, and BACT2. Although mutations are spread throughout these MAGs, especially for small values of p, there are obvious coldspots. For example, there are two particularly large regions without any identified p-mutations located near 1.6–1.8 Mbp in BACT1: these regions have no p-mutations even when $p = 0.5\%$, a surprising result. Appendix B.15 further investigates the coldspots in BACT1.

Hotspots can also be identified from Figure B.16, although they are less easy than

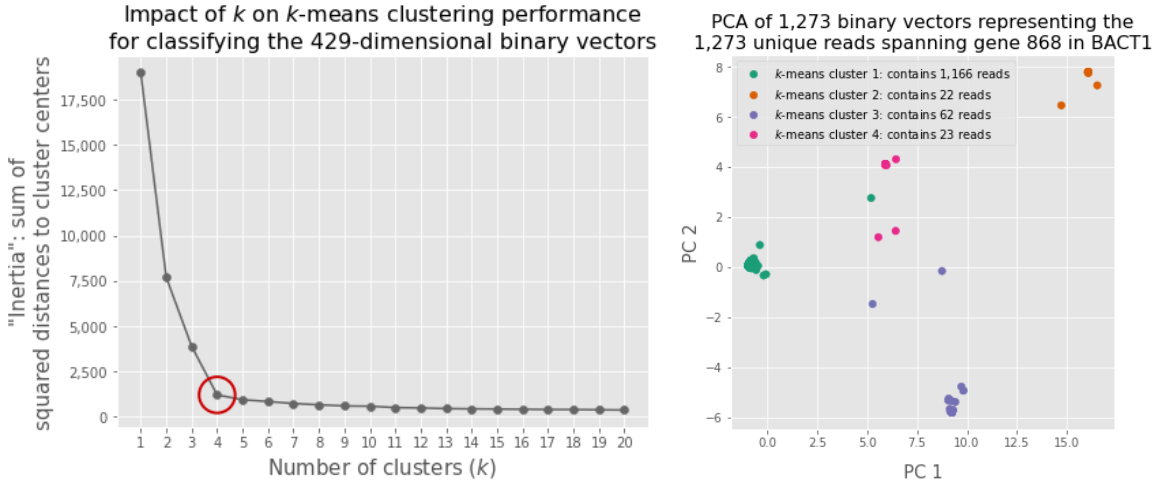


Figure B.15. **(Left)** Plot of k versus k-means clustering performance for the 1,273 429-dimensional binary vectors representing reads spanning gene 868 in BACT1. Per the scikit-learn documentation at https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, the "inertia" measurement describes the "sum of squared distances of samples to their closest cluster center" [147]; the use of inertia as the clustering evaluation metric is based on the tutorial given at https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/. We highlight $k = 4$ here, which represents a clear "elbow" in this plot. **(Right)** Principal component analysis (PCA) plot of the 1,273 429-dimensional binary vectors. Each vector's point is colored by its assigned cluster from k-means clustering with $k = 4$, using the "Dark2" color map from ColorBrewer [19]. The two-dimensional visualization provided by PCA mostly separates the four clusters. We performed PCA using scikit-learn [147].

Figure B.16. Locations of p-mutations throughout the three selected MAGs. Black or green vertical lines indicate p-mutations; black lines correspond to mutations located within at least one gene, while green lines correspond to mutations located within intergenic regions. The values of p shown here are the same as those shown in the context of the diversity index plots in Figure 3.4. p-mutations are only identified for sufficiently-covered positions (represented by gray horizontal bars); insufficiently-covered positions are marked with red horizontal bars instead. We determine "sufficient coverage" as described in Appendix B.11 for p-mutations, using $minReadNumber = 5$. As was also the case in Figure 3.4, BACT1 is not sufficiently covered for $p = 0.25\%$ (the minimum sufficient coverage for this value of p is $\frac{5}{0.0025} = 2{,}000$x). These plots are a very coarse, high-level way of visualizing mutation locations across these MAGs, and identifying small single-nucleotide-level details is challenging due to the limited resolution available. However, these plots are nonetheless useful for making high-level comparisons between the distributions of p-mutations across these MAGs. We note that MAGs are represented here linearly rather than circularly, a distinction from the figures in [175, 213]. This representation is used both to simplify the comparison of many different sequences and values of p, and because the sequence corresponding to CAMP is not completely assembled into a single circular sequence (as discussed in Appendix B.2).


Locations of p-mutations
Mutations within one or more genes
Mutations within intergenic regions
Sufficiently-covered positions
Insufficiently-covered positions
CAMP
BACT1
BACT2
p = 50%
p = 25%
p = 10%
p = 5%
p = 2%
p = 1%
p = 0.5%
p = 0.25%
p-mutations
Sequence position (1-indexed)

coldspots to distinguish in this visualization. For example, when $p = 10\%$ and when $p = 5\%$, 34 of the 35 p-mutations in CAMP are represented by the single black bar located near the 1.2 Mbp point in the MAG: these 34 mutations, it turns out, are all contained within a single highly-mutated gene in CAMP (shown in Figure 3.5).

B.15 Investigating coldspots

Taxonomic classifications of long coldspots in BACT1.

At $p = 0.5\%$, and without attempting to call p-mutations at positions with coverage less than 1,000x (Figure B.16), BACT1 contains 7 "coldspots" corresponding to regions of length $\geq$ 5,000 bp without any p-mutations. Table B.3 describes the location, length, average coverage, and taxonomic classifications of each of these coldspots.

We suspected that some of these coldspots might correspond to conserved sequences (e.g. parts of 16S rRNA genes) that would not vary much across genomes. To check this, we ran barrnap [173] on BACT1; although barrnap predicted multiple rRNA genes, none of these predicted genes' coordinates intersected with any of the seven coldspots investigated here. The cause of these coldspots is thus still unclear. The two coldspots for which BLASTN identified matches to viral genomes may indicate recent prophage insertions that could justify the lack of mutations in these regions [114], although additional work would be needed to prove this.

Evaluating statistical significance of coldspots.

Although it is tempting to assume that long "gaps" between mutations in a contig have some biological meaning, gaps between mutations could arise by chance. To provide additional context for the gaps identified in a dataset, the `strainFlye spot cold-gaps` command can compute a simple *p-value* describing the probability of the longest gap in a contig being at least a certain length. The computation of this p-value relies on the null hypothesis that each position in the contig is defined as mutated or not independently,

using a fixed contig-specific mutation rate. This assumption reduces the problem of computing this p-value to the classical problem of analyzing the distribution of the length of the longest run of heads in a sequence of coin tosses [10].

For each contig with at least one mutation and at least one gap meeting a user-set minimum length threshold, we define the *mutation rate* of this contig, q, as the ratio of mutated positions in this contig (m) to the total number of positions in this contig (n) [56]. Under our null hypothesis, the mutation status of each position in a contig can be thought of as the result of a Bernoulli trial with probability of "failure" (the position is mutated) set to $q = \frac{m}{n}$, and probability of "success" (the position is not mutated) set to $p = 1 - q$. If the null hypothesis were true, then the number of non-mutated positions in a contig should thus follow the Binomial distribution with expected value $np = n\left(1 - \frac{m}{n}\right) = n - m$.

This null hypothesis is an obvious oversimplification that will be broken in practice: for example, Figure 3.3 makes clear that the probability of a position being mutated is not the same for all positions in a contig. However, we submit that it is an acceptable approximation, especially since these p-values are intended only as a starting point for further analysis of these gaps.

For each contig, we can compute the probability of the longest gap in the contig (equivalently, the longest run of "successes" in a sequence of Bernoulli trials) using the final equation shown in [10]. We allow the user to choose between attempting to compute exact p-values using the equation from [10], or using the approximation method shown in [137]. For example: using NaiveFreq p-mutation calling at $p = 0.5\%$ (and ignoring whether or not sufficient coverage exists, unlike in Table B.3 and Figure B.16), the longest coldspot in BACT1 has length 19,167 bp (corresponding to the fifth coldspot listed in Table B.3). The probability of seeing a gap at least this long by chance, considering BACT1's length and the amount of $p = 0.5\%$ mutations contained therein, is 6.0498×10^{-91} (computed using the exact method from [10]).

We note that these p-values come with some caveats in addition to the obvious

Table B.3. Information about all coldspots of length $\geq$ 5,000 bp in BACT1. Coldspots are listed from left to right in the MAG. For the purposes of this table we define coldspots as regions of BACT1 containing no p-mutations at $p = 0.5\%$, and we only attempt to call p-mutations at sufficiently-covered positions (where the coverage is at least $\frac{5}{0.005} = 1{,}000$x, as discussed in Appendix B.11). As in Table B.1, the "Coverage" column describes the rounded average coverage of each region: most coldspots have coverages consistent with the BACT1 average of 1,415x, although the third coldspot (containing the large region of insufficiently-covered positions in the middle of BACT1 shown in Figure B.16) has a much lower coverage. We provide taxonomic classifications to provide a basic analysis of why these parts of BACT1 are "cold"—due, for example, to having evolutionarily conserved functions or arising from prophage insertions. The "Kaiju" column describes the taxonomic assignments from Kaiju [124] of each region, computed as described in Table B.1. Since Kaiju was only able to classify the entirety of BACT1 as *Bacteria*, the diversity of taxonomic classifications shown for different regions of this MAG is not necessarily surprising. This diversity could be due to a variety of factors: for example, this MAG could correspond to a novel species, or it could contain multiple strain-level genomes collapsed into a single sequence. This latter situation is common for many MAGs, for many metagenomic assemblers; this is in part a consequence of us running metaFlye without the `--keep-haplotypes` flag, as well (Appendix B.2). Since some of these coldspots are relatively short (Kaiju relies on protein-level matches), we also used BLASTN 2.12.0+ [218] on the NCBI nucleotide collection database [130] with default parameters to attempt to assign matches to these regions. The "BLASTN" column describes the scientific name of the sequence listed first in the BLASTN results for each coldspot region, sorting sequence matches by ascending E-values; BLASTN did not identify any matches for the leftmost coldspot.

Left	Right	Length (bp)	Coverage	Kaiju	BLASTN
740,701	750,399	9,699	1,261x	*Clostridium* sp. CAG:1013	"No significant similarity found"
974,177	979,570	5,394	1,226x	*Bacteroides fragilis*	*Emticicia oligotrophica* DSM 17448
1,183,798	1,229,941	46,144	432x	*Tannerella* sp. CAG:118	*Podoviridae* sp.
1,229,943	1,239,536	9,594	1,384x	*Prevotella disiens*	*Podoviridae* sp.
1,618,448	1,637,614	19,167	1,454x	*Bacteroides uniformis*	*Phocaeicola salanitronis* DSM 18170
1,798,600	1,813,374	14,775	1,374x	*Alistipes* sp. CHKCI003	Uncultured organism clone VC1D523TF
2,140,896	2,147,770	6,875	1,345x	*Bacteroidales*	*Ruminococcus* sp. JE7A12

simplicity of the null hypothesis used. For one, we do not correct for multiple testing (so blindly classifying all longest-gap p-values less than 0.05, for example, as "significant" may be problematic when considering datasets with many contigs). Additionally, the equation for computing the p-value of a gap is given only for a sequence of Bernoulli trials; in the rare case where the longest coldspot gap in a contig is the "loop-around" gap from the rightmost mutation to the leftmost mutation, we implicitly ignore that this break in the contig exists, and act as if this gap were instead in the middle of the contig. This may technically violate the assumptions made by the equation, but we do not imagine it should complicate the interpretation of these p-values.

B.16 Growth dynamics

Here we demonstrate simple analyses of the three selected MAGs' growth dynamics by comparing their coverages (Appendix B.3) and GC skews. We emphasize that strainFlye performs these analyses without reliance on reference databases.

A global maximum in a coverage plot usually indicates the origin of replication for a bacterial genome that is undergoing replication, and the ratio of this global maximum's coverage to a global minimum's coverage (the *peak-to-trough ratio*, or *PTR*) can measure *in vitro* bacterial growth rates [99]. Similarly, the global minimum of *GC skew* of a bacterial genome can also be indicative of its origin of replication [108, 63]. Figure B.17 shows plots of coverage and GC skew for the three selected MAGs, demonstrating a clear anticorrelation between coverage and skew. To simplify these plots, we bin each MAG's coverage and skew; the `strainFlye dynam covskew` command can compute these binned quantities.

We can compute the PTR [99] for each MAG by computing the ratio of normalized coverage at the bin with the global minimum of the skew to the ratio of normalized coverage at the bin with the global maximum of the skew; both extrema are highlighted

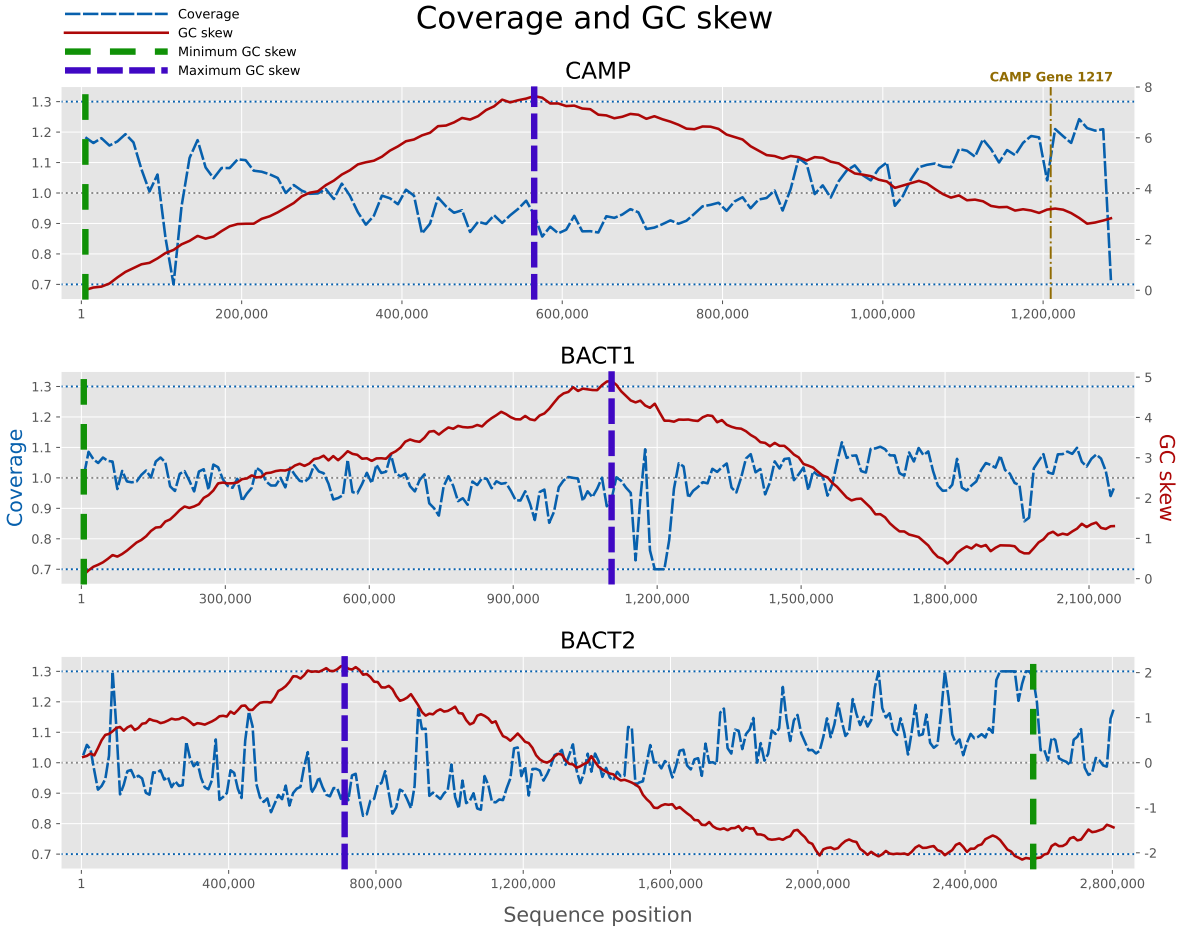
in Figure B.17. CAMP's PTR is $\frac{1.18}{0.93} \approx 1.27$; BACT1's PTR is $\frac{1.01}{0.93} \approx 1.08$; and BACT2's PTR is $\frac{1.28}{0.91} \approx 1.40$. All of these values are consistent with the distributions of PTRs shown in Figure 1(B) of [99], and they indicate that—if the organisms represented by these MAGs are indeed undergoing replication—the replication rate is higher for BACT2 and CAMP than for BACT1. Although further work would be needed to validate these claims, our ability to make these observations shows that HiFi-based metagenomics enables the evaluation of the growth dynamics of a metagenome.

We close by noting that, beyond serving as an example of the utility of complete metagenomics, information about PTRs can be useful in a variety of contexts when studying human-associated microbiota [85]. Prior work has also linked mutation rates and replication timing within a single yeast chromosome [103]; so, information about putative replication origins has potential to inform a "prior" about how likely mutations in certain regions of a MAG are, extending on the approaches for identifying rare mutations described in this paper.

B.17 The link graph structure for haplotype visualization

Here we define *link graphs* for representing mutations within a MAG that likely co-occur on the same strain. Nodes in the link graph represent *alleles* of mutated positions, and edges connect alleles that are observed together in the reads. The link graph resembles the two graph structures (the "simple graph" and "probabilistically weighted graph") for metagenomic haplotyping proposed in [138]. The link graph differs from the graphs proposed in that paper in that non-adjacent pairs of mutations can be connected; the edge weighting methods are also somewhat simpler than those defined for the probabilistically weighted graph [138]. That said, we expect that for many cases these graphs should be fairly similar. We provide the `strainFlye link` module for the construction of these

Figure B.17. Coverage and GC skew throughout the three selected MAGs. We bin and normalize coverages in order to simplify comparisons (both between coverage and skew for the same MAG, and between coverages of different MAGs). Starting at the left end of each MAG, each 10 kbp of positions are combined into a single bin. (The rightmost bin for each MAG can contain less than 10 kbp positions.) We compute the median coverage of each bin and then compute the entire MAG's median coverage (M) by taking the median of all bins' median coverages. We then divide each bin's median coverage by M in order to normalize each bin's coverage. These normalized coverages are clamped to the range $[0.7, 1.3]$ to limit the visual impact of outliers that likely represent sequencing artifacts: these boundaries, in addition to $y = 1$, are represented as horizontal dotted lines on each plot. We compute GC skew for the same bins of 10 kbp used for coverage. The GC skew of a bin containing G instances of G and C instances of C is defined as $\frac{G-C}{G+C}$, plus—for all bins after the leftmost—the GC skew of the bin immediately to the left of the current bin [63]. The midpoints of the bins with the global minimum and maximum GC skew in each MAG are highlighted with vertical lines, a choice inspired by Figure 1 of [99]. We also highlight the midpoint of gene 1217, the highest-mutation-rate gene in CAMP, as another vertical line in the CAMP plot. This enables us to follow up on our observation in Figure B.14 in Appendix B.12 that this gene's coverage is increasing. Here we can confirm that this gene is located in a region of the MAG where, in general, coverage is increasing and GC skew is decreasing; this may be indicative that this region of the CAMP is being actively replicated. Although all three MAGs' skew plots have clear global maxima, only BACT2's skew plot has an "obvious" global minimum. The fact that CAMP is not completely assembled, as discussed in Appendix B.2, may be a causal factor for its lack of a clear global minimum skew.


Coverage and GC skew
Coverage
GC skew
Minimum GC skew
Maximum GC skew
CAMP
CAMP Gene 1217
BACT1
BACT2
Coverage
GC skew
Sequence position

graphs.

Defining a link graph.

We define one link graph per MAG. For each position i at which a mutation was called, we define reads(i, N) as the number of reads with nucleotide $N \in \{A, C, G, T\}$ at position i. We add up to four nodes for each mutated position: each node (i, N) represents the occurrence of a specific nucleotide at position i, and is only added if reads$(i, N) > 1$.

We then consider all pairs of nodes (i, N_i) and (j, N_j), where $i \neq j$. We define a read as an (i, j, N_i, N_j)-*read* if its aligned nucleotide at position i is N_i and its aligned nucleotide at position j is N_j. We define reads(i, j, N_i, N_j) as the number of (i, j, N_i, N_j)-reads. We can now define the *link weight* between two alleles:

$$\text{link}(i, j, N_i, N_j) = \frac{\text{reads}(i, j, N_i, N_j)}{\max(\text{reads}(i, N_i), \text{reads}(j, N_j))}$$

We also define

$$\text{spanCount}(i, j) = \sum_{N_i \in \{A,C,G,T\}} \left(\sum_{N_j \in \{A,C,G,T\}} \Big(\text{reads}(i, j, N_i, N_j) \Big) \right),$$

which is the total number of reads spanning two positions i and j (summing across all combinations of alleles).

Finally, we connect two nodes (i, N_i) and (j, N_j) by an undirected edge of weight link(i, j, N_i, N_j) if (i) spanCount$(i, j) \geq minSpan$, and (ii) link$(i, j, N_i, N_j) > 0$. In the context of the high-coverage MAGs in SheepGut, we set the *minSpan* parameter to 501; lower values of this parameter will be required for lower-coverage MAGs, or for lower-coverage sequencing projects. Visualizations of the distributions of reads(i, j, N_i, N_j) and link(i, j, N_i, N_j) across the link graphs of the three selected MAGs in SheepGut are shown in Figure B.18.

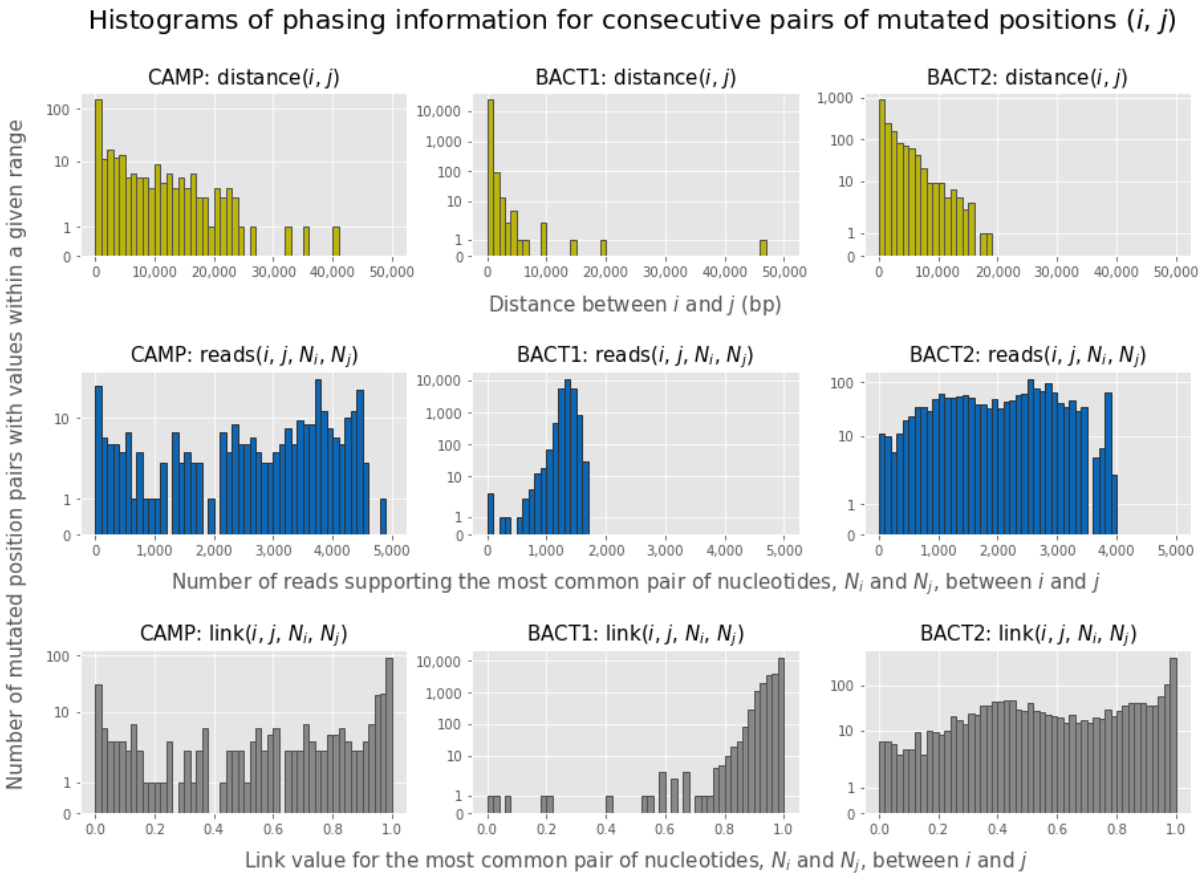


Figure B.18. Histograms showing various phasing statistics for each pair of consecutive mutated positions (i, j) in the three selected MAGs. Mutated positions were identified by NaiveFreq at $p = 0.5\%$, and filtered to just positions with a coverage of at least 1,000x. Two mutated positions i and j are considered as a pair of *consecutive* mutated positions if no other mutated positions (using the same criteria of $p = 0.5\%$ and minimum coverage $\geq$ 1,000x) occur in between i and j. (For BACT1 and BACT2, which are represented as connected components of a single circular sequence in the assembly graph, we consider the rightmost mutation r and the leftmost mutation ℓ as a pair of consecutive mutated positions, since reads can "loop around" these two MAGs' sequences in the alignment.) The top row shows a histogram of the distance between i and j. The middle row shows reads(i, j, N_i, N_j): for each pair (i, j), only the most frequent pair of nucleotide positions (N_i, N_j) is included here. The bottom row shows link(i, j, N_i, N_j), selecting only the most frequent shared haplotype in the same way as the second row. We note that, if a pair of consecutive mutated positions (i, j) are not spanned by at least one read, then reads(i, j, N_i, N_j) = link$(i, j, N_i, N_j) = 0$ for this pair. (This is the case for all pairs with distances above 25,000 bp shown in the first row of plots.)

Visualizing link graphs.

Connected components of the link graph represent groups of mutations that tend to co-occur. Here we visualize components of two link graphs from the SheepGut dataset, produced using the set of mutated positions identified by NaiveFreq at $p = 0.5\%$ with coverage of at least $minCov = 1{,}000$x.

As expected, the largest few connected components (sorted by number of nodes per component) in the link graph of BACT1, which contains many rare mutations, are much larger than the largest components of CAMP and BACT2. Figure B.19 shows the largest connected component of the link graphs for BACT1 and CAMP. Nodes are colored according to the gene(s) in which the node's position is contained. This indicates that the BACT1 component spans a large region of BACT1 and that the CAMP component spans a comparatively small region of CAMP. This is a consequence of highly-mutated MAGs being easier to phase than sparsely mutated ones [138].

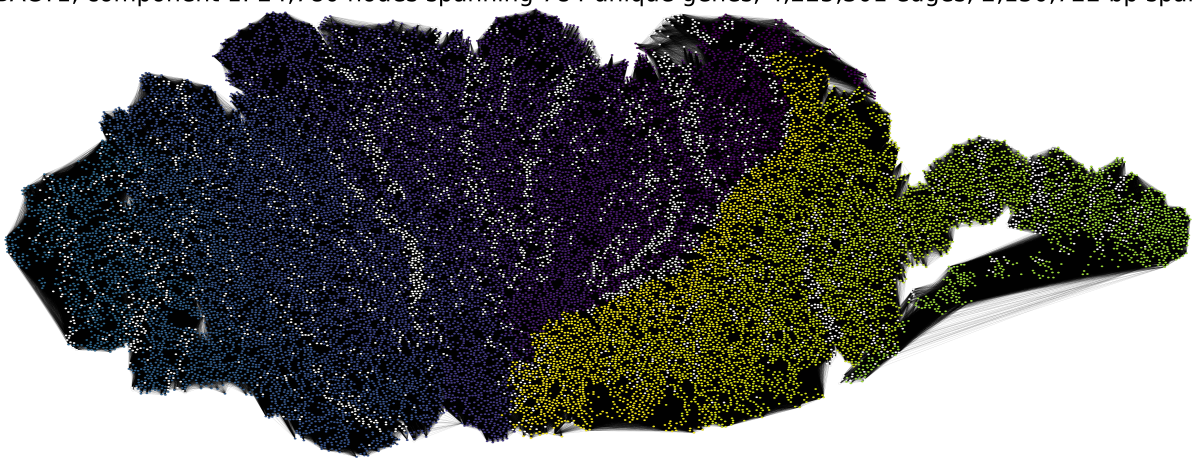
Link graphs can be helpful for visualizing what portions of a MAG are possible to phase, but their general usefulness is limited. strainFlye thus also supports the ability to generate smoothed haplotypes directly, as discussed in section 3.3.10, "Phasing identified mutations". These analyses and visualizations demonstrate the ability of HiFi reads to span long regions of highly-mutated MAGs, and thus indicate putative haplotypes.

B.18 Haplotypes of the most mutated gene in CAMP

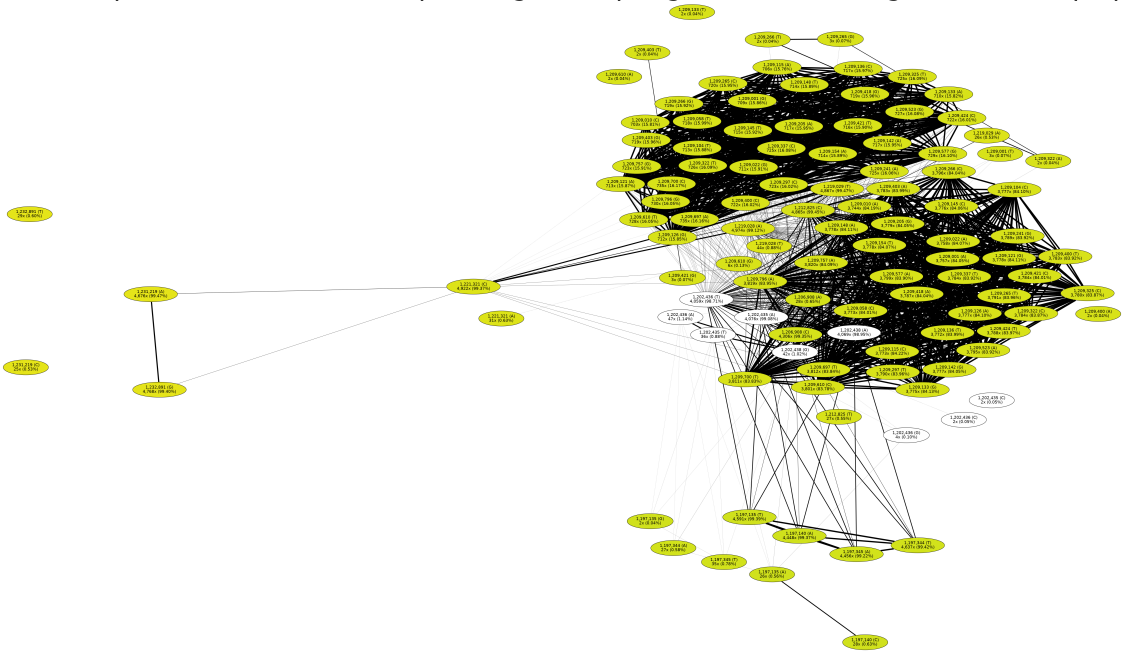
During the development of strainFlye's `smooth` module, we tested our pipeline against a simple example—the region surrounding the highest-mutation-rate gene in CAMP (Figure 3.5 (top))—in order to provide a simple benchmark for our ability to recover strain-level haplotypes from a MAG. We found that using the jumboDBG module of LJA [8] by itself, or using the jumboDBG and multiplexDBG modules of LJA without

Figure B.19. Largest components in the link graphs of BACT1 (top) and CAMP (bottom). Nodes (alleles) are colored by the gene(s) in which their corresponding position is located. Nodes present in a single gene are colored using the viridis gradient [179]: darker (lighter) colors correspond to genes closer to the start (end) of the MAG. Positions present in multiple genes are colored gray, and positions located in intergenic regions are colored white. Nodes in the BACT1 (CAMP) component span a large (small) region of BACT1 (CAMP). The smaller CAMP component includes additional labels for each node: (i) position in the MAG, (ii) nucleotide at this position, (iii) the frequency with which this nucleotide was observed at this position (reads(i, N)), and (iv) the relative frequency of this nucleotide at this position. Many of the mutated positions in this component are located within the gene shown in Figure 3.5 (top). Edges' thicknesses in the CAMP component's visualization are scaled by their link values, so that thicker edges roughly correspond to higher link values. Interestingly, this component contains two large "clique"-like structures including many nodes that have high-link edges between each other. These two structures correspond to the two different haplotypes of CAMP within this gene shown in Figure 3.5 (top), with one haplotype at roughly 16% relative frequency and another haplotype at roughly 84% relative frequency. We visualized these graphs using Graphviz [54], using the sfdp [76] layout method.

BACT1, component 1: 24,780 nodes spanning 784 unique genes, 4,225,301 edges, 2,150,722 bp span



CAMP, component 1: 110 nodes spanning 9 unique genes, 3,473 edges, 35,757 bp span



any error correction step, resulted in assembly graphs more complex than we expected. We wanted to investigate methods for simplifying this graph while limiting the removal of real strain-level variation; here we document these tests.

Analysis of haplotypes of this gene.

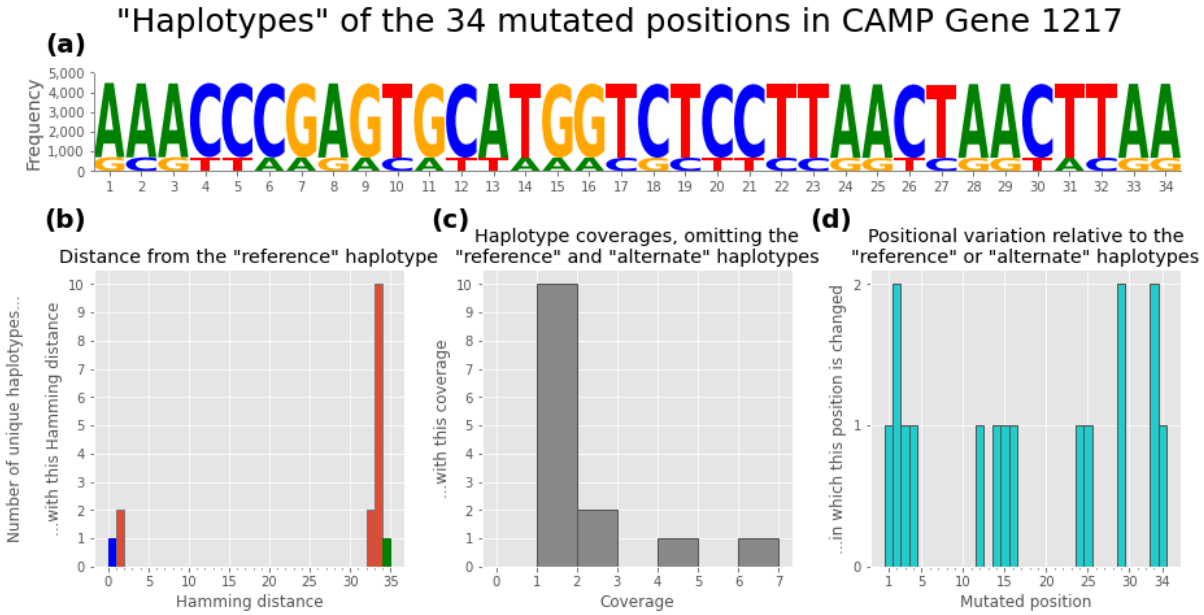
We limit our focus only to reads that surround the highest-mutation-rate gene (gene 1217) in CAMP. To do this, we define a region of positions in CAMP that includes the entirety of gene 1217, as well as 25 kbp before and after this gene. We will only consider smoothed reads whose corresponding linear alignments to CAMP overlap with this region (we do not generate any virtual reads for this particular analysis).

We define M as the set of all 34 mutated positions (given $p = 10\%$, based on Figure 3.5 (top)) located in gene 1217 in CAMP. There are 4,054 smoothed reads originating from linear alignments that span all positions in M without deletions or skips. We extracted the haplotypes from these smoothed reads at all positions in M. For the purposes of this Appendix, we now define a *haplotype* of a smoothed read as a string of 34 characters, such that the n-th character of this haplotype corresponds to this smoothed read's nucleotide at the n-th mutated position in gene 1217.

Figure B.20(a) shows a sequence logo that confirms that these mutated positions are consistently dominated by a "reference" ($\sim$84% frequency, at 3,430x coverage) and "alternate" ($\sim$16% frequency, at 600x coverage) haplotype. These two haplotypes are herein referred to as the "primary" haplotypes of this gene.

There are, however, 14 unique haplotypes present in the smoothed reads which differ from the primary haplotypes. The first question we had about these "non-primary" haplotypes was how similar they were to the two primary haplotypes. We computed the Hamming distance [68] between all unique haplotypes and the "reference" haplotype; a histogram of these Hamming distances is shown in Figure B.20(b). This histogram clarifies that all non-primary haplotypes closely resemble the reference or alternate haplotypes.

Figure B.20. Haplotypes of the 34 mutated positions ($p = 10\%$) located within gene 1217 of CAMP. **(a)** Sequence logo of these mutated positions' aligned nucleotides' frequencies. Since we created this plot based on the haplotypes represented in the smoothed reads, it thus only includes the two most common aligned nucleotides at each of these mutations (as discussed in section 3.5, "Methods"). However, this caveat should not make a large difference. This logo was visualized using Logomaker [187]. **(b)** Histogram of Hamming distances for all of the 16 unique haplotypes spanning gene 1217 in CAMP, relative to the reference haplotype. Since these haplotypes are 34-character strings, the Hamming distance of a given haplotype is constrained to within the range $[0, 34]$. This histogram includes the reference and alternate haplotypes for illustrative purposes: the reference haplotype (which has a Hamming distance of 0 to itself) is located in the leftmost bin and colored blue, while the alternate haplotype (which has the maximum possible Hamming distance of 34, since it disagrees with the reference haplotype at every mutated position) is located in the rightmost bin and colored green. This plot shows that most haplotypes' Hamming distances are similar to that of either the reference or alternate haplotype, implying that most of these haplotypes represent small deviations from these two "primary" haplotypes. **(c)** Histogram of haplotype coverages for all unique non-primary haplotypes. The two primary haplotypes are omitted from this histogram due to their relatively large coverages (3,430x and 600x); the non-primary haplotypes all have coverages of less than 10x, indicating that these haplotypes may correspond to sequencing errors or extremely rare mutations. For reference, the highest-coverage (6x) non-primary haplotype, `GAGTTAAGACATTAAACGCTTCCGGTCGGTACGG`, has a Hamming distance to the reference haplotype of 33, meaning that this haplotype disagrees with the alternate haplotype at only one mutated position (the second, at which it has an A instead of a C). **(d)** Chart of positional variation for the non-primary haplotypes. The x-axis corresponds to the 34 mutated positions in gene 1217. For each of the non-primary haplotypes, we compute two Hamming distances: D_R is the Hamming distance to the reference haplotype, and D_A is the Hamming distance to the alternate haplotype. If $D_R > D_A$, we add "+1" for all differing positions relative to the alternate haplotype; if $D_R < D_A$, we add "+1" for all differing positions relative to the reference haplotype. ($D_R \neq D_A$ for all haplotypes, and—as indicated in panel (b)—all but two of the non-primary haplotypes have $D_R > D_A$.) This plot indicates that the distribution of differing positions is roughly uniform across the non-primary haplotypes.


"Haplotypes" of the 34 mutated positions in CAMP Gene 1217
(a)
Frequency
(b)
Distance from the "reference" haplotype
Number of unique haplotypes...
...with this Hamming distance
Hamming distance
(c)
Haplotype coverages, omitting the "reference" and "alternate" haplotypes
...with this coverage
Coverage
(d)
Positional variation relative to the "reference" or "alternate" haplotypes
...in which this position is changed
Mutated position

To get a sense of how common the non-primary haplotypes are compared to each other (e.g. "does one of these haplotypes occur frequently?"), we plot a histogram of these haplotypes' coverages in Figure B.20(c). This histogram reassures us that all non-primary haplotypes are extremely uncommon, considering that the total coverage of smoothed reads spanning all mutated positions in gene 1217 in CAMP is 4,054x.

As we consider whether these non-primary haplotypes represent real or technical variation, we next investigate whether these non-primary haplotypes consistently vary at certain mutated positions. The 34 mutated positions from which we have constructed these haplotypes are all already "hotspots," but it may turn out that some of these positions may vary in the same way for many unique haplotypes—this could indicate that these haplotypes are evolutionarily related to each other. (A more formal way of conducting this analysis might involve constructing a phylogenetic tree.) As a simple way to check this, we plot the number of times each of the 34 mutated positions varies from either the reference or alternate haplotypes in Figure B.20(d).

Figure B.20(d) provides evidence against the idea that these non-primary haplotypes consistently vary from the reference or alternate haplotypes at the same mutated positions, since—rather than being biased toward changes at a certain mutated position—the distribution of differences across each mutated position is roughly uniform (albeit with many zeroes, since there are only 14 unique non-primary haplotypes and 34 possible mutated positions).

Taken together, these results thus serve as (imperfect) justification for ignoring these non-primary haplotypes of gene 1217: in general these haplotypes are low-coverage (Figure B.20(c)), only differ subtly from the primary haplotypes (Figure B.20(b)), and do not seem to differ from these primary haplotypes in consistent ways (Figure B.20(d)). For a first-pass analysis, we thus propose removing low-coverage edges in the graph that likely arise due to these haplotypes.

Assembly of haplotypes of this gene.

Based on our analysis of the haplotypes of gene 1217 in CAMP (Figure B.20), we restructured our use of LJA to include a simple coverage filter applied after running jumboDBG but before running multiplexDBG. This coverage filter removes edges from the graph with k-mer coverages below a user-configurable threshold $minKmerCov$ (default value 10; we note that "k-mer coverage" is not exactly comparable to the conventional definition of coverage).

Running LJA in this way on the smoothed reads intersecting gene 1217 in CAMP (using $minKmerCov = 10$) results in a simple assembly graph containing two isolated edges. We can determine the haplotype of gene 1217 represented by an edge by aligning this edge's sequence to CAMP using minimap2 [106] and then extracting the nucleotides of this edge that have been matched to all mutated positions. For the sake of brevity, in the remainder of this Appendix we refer to this process as *haplotype extraction*.

Using haplotype extraction, we confirmed that one of the edges produced by LJA with $minKmerCov = 10$ represents the reference haplotype, while the other edge represents the alternate haplotype. From our perspective, this is the "ideal" result for this simple example, and this bodes well for the application of this approach to other, more complex read-sets (although more thorough testing on different types of communities would be ideal).

Benchmarking assembly of haplotypes of this gene.

For comparison's sake, we also ran six other assembly methods in order to determine which haplotypes of gene 1217 in CAMP were recovered by these methods. We ran metaFlye (with default parameters), metaFlye (with the `--keep-haplotypes` flag), jumboDBG (with $k = 5{,}001$), LJA (with no error correction), LJA (with its default "mowerDBG" error correction), and LJA (with the simple k-mer coverage filter described above, but using $minKmerCov = 1$ instead of 10).

Three of these methods (both metaFlye runs, and LJA with mowerDBG error correction) produced assemblies containing only one edge. Using haplotype extraction, we confirmed that each of these lone edges corresponded to the reference haplotype. These assemblies are thus "over-corrected," since they do not include the alternate haplotype (which clearly corresponds to a real alternate strain).

The other three of these methods (jumboDBG, LJA with no error correction, and LJA with $minKmerCov = 1$) produced assemblies containing more than two edges. jumboDBG's assembly contained 32 edges, while the two LJA runs' assemblies both contained five edges. Although all three assemblers recovered the reference and alternate haplotypes (as determined by haplotype extraction of the assembled edges), the fact that they recovered other haplotypes is undesirable—since, as discussed earlier, these other haplotypes likely represent noise. These assemblies are thus "under-corrected" for this particular example, relative to the run of LJA with $minKmerCov = 10$.

B.19 Smoothed haplotype assembly graphs

Here we present additional visualizations of the assembly graphs produced by LJA for MAGs' smoothed and virtual reads, as described in section 3.3.10, "Phasing identified mutations". Figure B.21 shows MetagenomeScope [46] visualizations of the multiplex de Bruijn graphs for BACT1 and BACT2, analogous to the visualization for CAMP shown in Figure 3.6 in section 3.3.10, "Phasing identified mutations".

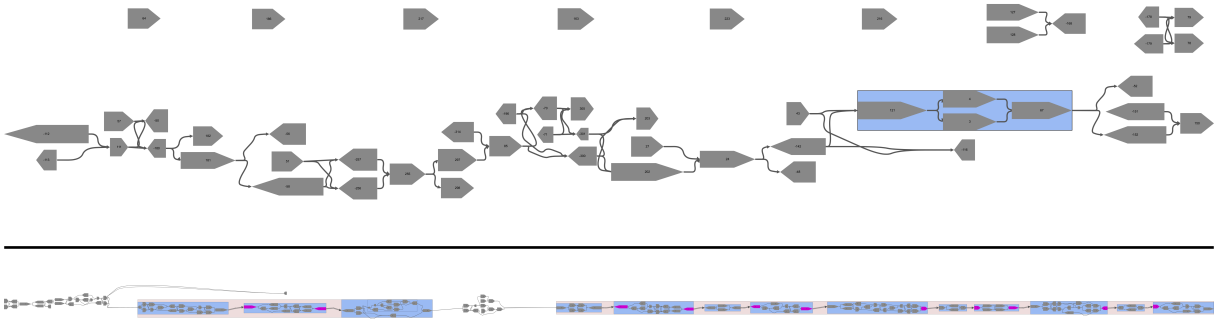


Figure B.21. Multiplex de Bruijn graphs produced by LJA for the BACT1 (top) and BACT2 (bottom) MAGs' smoothed and virtual reads. The BACT1 graph is surprisingly simple; this is likely a consequence of many potential smoothed reads being discarded, as discussed in section 3.5, "Methods". The BACT2 graph, however, resembles CAMP's (Figure 3.6) in that it primarily consists of a series of consecutive "bubble" patterns. We produced these visualizations using MetagenomeScope [46].

Bibliography

[1] Louis Abraham. pydivsufsort. https://github.com/louisabraham/pydivsufsort, 2023.

[2] John Aitchison and Michael Greenacre. Biplots of compositional data. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 51(4):375–392, October 2002.

[3] Amnon Amir, Daniel McDonald, Jose A. Navas-Molina, Evguenia Kopylova, James T. Morton, Zhenjiang Zech Xu, Eric P. Kightley, Luke R. Thompson, Embriette R. Hyde, Antonio Gonzalez, and Rob Knight. Deblur Rapidly Resolves Single-Nucleotide Community Sequence Patterns. *mSystems*, 2(2):e00191–16, April 2017.

[4] Noemi Andor, Trevor A. Graham, Marnix Jansen, Li C. Xia, C. Athena Aktipis, Claudia Petritsch, Hanlee P. Ji, and Carlo C. Maley. Pan-cancer analysis of the extent and consequences of intratumor heterogeneity. *Nature Medicine*, 22(1):105–113, 2016.

[5] Dmitry Antipov, Mikhail Rayko, Mikhail Kolmogorov, and Pavel A. Pevzner. viralFlye: assembling viruses and identifying their hosts from long-read metagenomics data. *Genome Biology*, 23(1):1–21, 2022.

[6] Amy Apprill, Sean McNally, Rachel Parsons, and Laura Weber. Minor revision to V4 region SSU rRNA 806R gene primer greatly increases detection of SAR11 bacterioplankton. *Aquatic Microbial Ecology*, 75(2):129–137, 2015.

[7] Evgeniy S. Balakirev and Francisco J. Ayala. Pseudogenes: are they "junk" or functional DNA? *Annual review of genetics*, 37(1):123–151, 2003.

[8] Anton Bankevich, Andrey V. Bzikadze, Mikhail Kolmogorov, Dmitry Antipov, and Pavel A. Pevzner. Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature Biotechnology*, 40(7):1075–1081, 2022.

[9] Jeffrey E. Barrick, Dong Su Yu, Sung Ho Yoon, Haeyoung Jeong, Tae Kwang Oh, Dominique Schneider, Richard E. Lenski, and Jihyun F. Kim. Genome evolution and adaptation in a long-term experiment with *Escherichia coli*. *Nature*, 461(7268):1243–1247, 2009.

[10] G. Bateman. On the power function of the longest run as a test for randomness in a sequence of alternatives. *Biometrika*, 35(1/2):97–112, 1948.

[11] Richard A. Becker and William S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[12] Richard A. Becker, William S. Cleveland, and Allan R. Wilks. Dynamic graphics for data analysis. *Statistical Science*, 2(4):355–383, 1987.

[13] Gaëtan Benoit, Sébastien Raguideau, Robert James, Adam M. Phillippy, Rayan Chikhi, and Christopher Quince. High-quality metagenome assembly from long accurate reads with metaMDBG. *Nature Biotechnology*, 42(9):1378–1383, 2024.

[14] Denis Bertrand, Jim Shaw, Manesh Kalathiyappan, Amanda Hui Qi Ng, M. Senthil Kumar, Chenhao Li, Mirta Dvornicic, Janja Paliska Soldo, Jia Yu Koh, Chengxuan Tong, Oon Tek Ng, Timothy Barkham, Barnaby Young, Kalisvar Marimuthu, Kern Rei Chng, Mile Sikic, and Niranjan Nagarajan. Hybrid metagenomic assembly enables high-resolution analysis of resistance determinants and mobile elements in human microbiomes. *Nature Biotechnology*, 37(8):937–944, 2019.

[15] Derek M. Bickhart, Mikhail Kolmogorov, Elizabeth Tseng, Daniel M. Portik, Anton Korobeynikov, Ivan Tolstoganov, Gherman Uritskiy, Ivan Liachko, Shawn T. Sullivan, Sung Bong Shin, Alvah Zorea, Victòria Pascal Andreu, Kevin Panke-Buisse, Marnix H. Medema, Itzhak Mizrahi, Pavel A. Pevzner, and Timothy P. L. Smith. Generating lineage-resolved, complete metagenome-assembled genomes from complex microbial communities. *Nature Biotechnology*, 40(5):711–719, 2022.

[16] Lee Bofkin and Nick Goldman. Variation in evolutionary processes at different codon positions. *Molecular Biology and Evolution*, 24(2):513–521, 2007.

[17] Nicholas A. Bokulich, Benjamin D. Kaehler, Jai Ram Rideout, Matthew Dillon, Evan Bolyen, Rob Knight, Gavin A. Huttley, and J. Gregory Caporaso. Optimizing taxonomic classification of marker-gene amplicon sequences with QIIME 2's q2-feature-classifier plugin. *Microbiome*, 6(1):90, 2018.

[18] Evan Bolyen, Jai Ram Rideout, Matthew R. Dillon, Nicholas A. Bokulich, Christian C. Abnet, Gabriel A. Al-Ghalith, Harriet Alexander, Eric J. Alm, Manimozhiyan Arumugam, Francesco Asnicar, Yang Bai, Jordan E. Bisanz, Kyle Bittinger, Asker Brejnrod, Colin J. Brislawn, C. Titus Brown, Benjamin J. Callahan, Andrés Mauricio Caraballo-Rodríguez, John Chase, Emily K. Cope, Ricardo Da Silva, Christian Diener, Pieter C. Dorrestein, Gavin M. Douglas, Daniel M. Durall, Claire Duvallet, Christian F. Edwardson, Madeleine Ernst, Mehrbod Estaki, Jennifer Fouquier, Julia M. Gauglitz, Sean M. Gibbons, Deanna L. Gibson, Antonio Gonzalez, Kestrel Gorlick, Jiarong Guo, Benjamin Hillmann, Susan Holmes, Hannes Holste, Curtis Huttenhower, Gavin A. Huttley, Stefan Janssen, Alan K. Jarmusch, Lingjing Jiang, Benjamin D. Kaehler, Kyo Bin Kang, Christopher R. Keefe, Paul Keim, Scott T. Kelley, Dan Knights, Irina Koester, Tomasz Kosciolek, Jorden Kreps, Morgan G. I. Langille, Joslynn Lee, Ruth Ley, Yong-Xin Liu, Erikka Loftfield, Catherine Lozupone, Massoud Maher, Clarisse Marotz, Bryan D. Martin, Daniel McDonald, Lauren J.

McIver, Alexey V. Melnik, Jessica L. Metcalf, Sydney C. Morgan, Jamie T. Morton, Ahmad Turan Naimey, Jose A. Navas-Molina, Louis Felix Nothias, Stephanie B. Orchanian, Talima Pearson, Samuel L. Peoples, Daniel Petras, Mary Lai Preuss, Elmar Pruesse, Lasse Buur Rasmussen, Adam Rivers, Michael S. Robeson, Patrick Rosenthal, Nicola Segata, Michael Shaffer, Arron Shiffer, Rashmi Sinha, Se Jin Song, John R. Spear, Austin D. Swafford, Luke R. Thompson, Pedro J. Torres, Pauline Trinh, Anupriya Tripathi, Peter J. Turnbaugh, Sabah Ul-Hasan, Justin J. J. van der Hooft, Fernando Vargas, Yoshiki Vázquez-Baeza, Emily Vogtmann, Max von Hippel, William Walters, Yunhu Wan, Mingxun Wang, Jonathan Warren, Kyle C. Weber, Charles H. D. Williamson, Amy D. Willis, Zhenjiang Zech Xu, Jesse R. Zaneveld, Yilong Zhang, Qiyun Zhu, Rob Knight, and J. Gregory Caporaso. Reproducible, interactive, scalable and extensible microbiome data science using QIIME 2. *Nature Biotechnology*, 37(8):852–857, July 2019.

[19] Cynthia Brewer, Mark Harrower, Ben Sheesley, Andy Woodruff, and David Heyman. ColorBrewer: Color advice for maps, 2021. https://colorbrewer2.org.

[20] Joshua N. Burton, Ivan Liachko, Maitreya J. Dunham, and Jay Shendure. Species-level deconvolution of metagenome assemblies with Hi-C–based contact probability maps. *G3: Genes, Genomes, Genetics*, 4(7):1339–1346, 2014.

[21] Andrey V. Bzikadze and Pavel A. Pevzner. UniAligner: a parameter-free framework for fast sequence alignment. *Nature Methods*, pages 1–9, 2023.

[22] J. Gregory Caporaso, Justin Kuczynski, Jesse Stombaugh, Kyle Bittinger, Frederic D. Bushman, Elizabeth K. Costello, Noah Fierer, Antonio Gonzalez Peña, Julia K. Goodrich, Jeffrey I. Gordon, Gavin A. Huttley, Scott T. Kelley, Dan Knights, Jeremy E. Koenig, Ruth E. Ley, Catherine A. Lozupone, Daniel McDonald, Brian D. Muegge, Meg Pirrung, Jens Reeder, Joel R. Sevinsky, Peter J. Turnbaugh, William A. Walters, Jeremy Widmann, Tanya Yatsunenko, Jesse Zaneveld, and Rob Knight. QIIME allows analysis of high-throughput community sequencing data. *Nature Methods*, 7(5):335, 2010.

[23] Sherwood Casjens. The diverse and dynamic structure of bacterial genomes. *Annual review of genetics*, 32(1):339–377, 1998.

[24] Thomas A. Caswell, Michael Droettboom, Antony Lee, John Hunter, Eric Firing, David Stansby, Jody Klymak, Tim Hoffmann, Elliott Sales de Andrade, Nelle Varoquaux, Jens Hedegaard Nielsen, Benjamin Root, Phil Elson, Ryan May, Darren Dale, Jae-Joon Lee, Jouni K. Seppänen, Damon McDougall, Andrew Straw, Paul Hobson, Christoph Gohlke, Tony S. Yu, Eric Ma, Adrien F. Vincent, Steven Silvester, Charlie Moad, Nikita Kniazev, Paul Ivanov, Elan Ernest, and Jan Katins. matplotlib/matplotlib v3.1.3, February 2020.

[25] Mark J. Chaisson, Benjamin J. Raphael, and Pavel A. Pevzner. Microinversions in mammalian evolution. *Proceedings of the National Academy of Sciences*, 103(52):19824–19829, 2006.

[26] Zhoutao Chen, Long Pham, Tsai-Chin Wu, Guoya Mo, Yu Xia, Peter L. Chang, Devin Porter, Tan Phan, Huu Che, Hao Tran, Vikas Bansal, Justin Shaffer, Pedro Belda-Ferre, Greg Humphrey, Rob Knight, Pavel Pevzner, Son Pham, Yong Wang, and Ming Lei. Ultralow-input single-tube linked-read library method enables short-read second-generation sequencing systems to routinely generate highly accurate and economical long-range sequencing information. *Genome Research*, 30(6):898–909, 2020.

[27] Brian Cleary, Ilana Lauren Brito, Katherine Huang, Dirk Gevers, Terrance Shea, Sarah Young, and Eric J. Alm. Detection of low-abundance bacterial strains in metagenomic datasets by eigengenome partitioning. *Nature Biotechnology*, 33(10):1053–1060, 2015.

[28] Phillip Compeau and Pavel Pevzner. *Bioinformatics Algorithms: An Active Learning Approach*. Active Learning Publishers La Jolla, California, 3 edition, 2018.

[29] Phillip Compeau and Pavel Pevzner. Bioinformatics Algorithms FAQ: Chapter 6. https://www.bioinformaticsalgorithms.org/faq-chapter-6, 2018.

[30] Phillip E. C. Compeau, Pavel A. Pevzner, and Glenn Tesler. How to apply de Bruijn graphs to genome assembly. *Nature Biotechnology*, 29(11):987–991, 2011.

[31] Joshimar Cordova and Gonzalo Navarro. Simple and efficient fully-functional succinct trees. *Theoretical Computer Science*, 656:135–145, 2016.

[32] Athel Cornish-Bowden. Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984. *Nucleic Acids Research*, 13(9):3021–3030, 1985.

[33] Wilfred R. Cuff, Venkata R. S. K. Duvvuri, Binhua Liang, Bhargavi Duvvuri, Gillian E. Wu, Jianhong Wu, and Raymond S. W. Tsang. A novel interpretation of structural dot plots of genomes derived from the analysis of two strains of *Neisseria meningitidis*. *Genomics, Proteomics and Bioinformatics*, 8(3):159–169, 2010.

[34] Petr Danecek, James K. Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O. Pollard, Andrew Whitwham, Thomas Keane, Shane A. McCarthy, Robert M. Davies, and Heng Li. Twelve years of SAMtools and BCFtools. *GigaScience*, 10(2):giab008, 2021.

[35] Margaret O. Dayhoff, R. Schwartz, and B. Orcutt. In Margaret O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, chapter A Model of Evolutionary Change in Proteins, pages 345–352. National Biomedical Research Foundation, 1978.

[36] George C. Dicenzo and Turlough M. Finan. The divided bacterial genome: structure, function, and evolution. *Microbiology and Molecular Biology Reviews*, 81(3):e00019–17, 2017.

[37] Nicolas Dierckxsens, Tong Li, Joris R. Vermeesch, and Zhi Xie. A benchmark of structural variation detection by long reads through a realistic simulated model. *Genome Biology*, 22(1):1–16, 2021.

[38] Lianming Du, Qin Liu, Zhenxin Fan, Jie Tang, Xiuyue Zhang, Megan Price, Bisong Yue, and Kelei Zhao. Pyfastx: a robust python package for fast random access to sequences from plain and gzipped FASTA/Q files. *Briefings in Bioinformatics*, 22(4):bbaa368, 2021.

[39] Stefan Elbe and Gemma Buckland-Merrett. Data, disease and diplomacy: GISAID's innovative contribution to global health. *Global Challenges*, 1(1):33–46, 2017.

[40] Joshua E. Elias and Steven P. Gygi. Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. *Nature Methods*, 4(3):207–214, 2007.

[41] Joshua E. Elias, Wilhelm Haas, Brendan K. Faherty, and Steven P. Gygi. Comparative evaluation of mass spectrometry platforms used in large-scale proteomics investigations. *Nature Methods*, 2(9):667–675, 2005.

[42] Kristen Emery, Syamand Hasam, William Stafford Noble, and Uri Keich. Multiple competition-based FDR control and its application to peptide detection. In *International Conference on Research in Computational Molecular Biology*, pages 54–71. Springer, 2020.

[43] Jimmy K. Eng, Ashley L. McCormack, and John R. Yates. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry*, 5(11):976–989, 1994.

[44] A. Murat Eren, Özcan C. Esen, Christopher Quince, Joseph H. Vineis, Hilary G. Morrison, Mitchell L. Sogin, and Tom O. Delmont. Anvi'o: an advanced analysis and visualization platform for 'omics data. *PeerJ*, 3:e1319, 2015.

[45] H. Christina Fan, Yair J. Blumenfeld, Usha Chitkara, Louanne Hudgins, and Stephen R. Quake. Noninvasive diagnosis of fetal aneuploidy by shotgun sequencing DNA from maternal blood. *Proceedings of the National Academy of Sciences*, 105(42):16266–16271, 2008.

[46] Marcus Fedarko, Jay Ghurye, Todd Treangen, and Mihai Pop. MetagenomeScope: web-based hierarchical visualization of metagenome assembly graphs. In *Proceedings of the 25th International Symposium on Graph Drawing and Network Visualization*, volume 10692, pages 630–632. Springer, 2017.

[47] Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Sunderland, MA, 2003.

[48] Xiaowen Feng, Haoyu Cheng, Daniel Portik, and Heng Li. Metagenome assembly of high-fidelity long reads with hifiasm-meta. *Nature Methods*, 19(6):671–674, 2022.

[49] Andrew D. Fernandes, Jennifer N. S. Reid, Jean M. Macklaim, Thomas A. McMurrough, David R. Edgell, and Gregory B. Gloor. Unifying the analysis of high-throughput sequencing datasets: characterizing RNA-seq, 16S rRNA gene sequencing and selective growth experiments by compositional data analysis. *Microbiome*, 2(1):15, 2014.

[50] Johannes Fischer and Florian Kurpicz. Dismantling DivSufSort. *arXiv*, 2017.

[51] Christina Frank, Dirk Werber, Jakob P. Cramer, Mona Askar, Mirko Faber, Matthias an der Heiden, Helen Bernard, Angelika Fruth, Rita Prager, Anke Spode, Maria Wadl, Alexander Zoufaly, Sabine Jordan, Markus J. Kemper, Per Follin, Luise Müller, Lisa A. King, Bettina Rosner, Udo Buchholz, Klaus Stark, and Krause, Gérard for the HUS Investigation Team. Epidemic profile of Shiga-toxin–producing *Escherichia coli* O104:H4 outbreak in Germany. *New England Journal of Medicine*, 365(19):1771–1780, 2011.

[52] Yoshinori Fukasawa, Luca Ermini, Hai Wang, Karen Carty, and Min-Sin Cheung. LongQC: a quality control tool for third generation sequencing long read data. *G3: Genes, Genomes, Genetics*, 10(4):1193–1196, 2020.

[53] Emden R. Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *International Symposium on Graph Drawing*, pages 239–250. Springer, 2004.

[54] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software: Practice and Experience*, 30(11):1203–1233, 2000.

[55] Rodrigo García-López, Jorge Francisco Vázquez-Castellanos, and Andrés Moya. Fragmentation and coverage variation in viral metagenome assemblies, and their effect in diversity calculations. *Frontiers in Bioengineering and Biotechnology*, 3:141, 2015.

[56] Ron Geller, Pilar Domingo-Calap, José M. Cuevas, Paola Rossolillo, Matteo Negroni, and Rafael Sanjuán. The external domains of the HIV-1 envelope are a mutational cold spot. *Nature Communications*, 6(1):1–9, 2015.

[57] Marco Gerlinger, Andrew J. Rowan, Stuart Horswell, James Larkin, David Endesfelder, Eva Gronroos, Pierre Martinez, Nicholas Matthews, Aengus Stewart, Patrick Tarpey, Ignacio Varela, Benjamin Phillimore, Sharmin Begum, Neil Q. McDonald, Adam Butler, David Jones, Keiran Raine, Calli Latimer, Claudio R. Santos, Mahrokh Nohadani, Aron C. Eklund, Bradley Spencer-Dene, Graham Clark, Lisa Pickering, Gordon Stamp, Martin Gore, Zoltan Szallasi, Julian Downward, P. Andrew Futreal,

and Charles Swanton. Intratumor heterogeneity and branched evolution revealed by multiregion sequencing. *New England Journal of Medicine*, 366:883–892, 2012.

[58] Jay Ghurye and Mihai Pop. Better identification of repeats in metagenomic scaffolding. In *Algorithms in Bioinformatics: 16th International Workshop, WABI 2016*, pages 174–184. Springer, 2016.

[59] Adrian J. Gibbs and George A. McIntyre. The diagram, a method for comparing sequences: Its use with amino acid and nucleotide sequences. *European Journal of Biochemistry*, 16(1):1–11, 1970.

[60] Gregory B. Gloor, Jean M. Macklaim, Vera Pawlowsky-Glahn, and Juan J. Egozcue. Microbiome Datasets Are Compositional: And This Is Not Optional. *Frontiers in Microbiology*, 8, 2017.

[61] Antonio Gonzalez, Jose A. Navas-Molina, Tomasz Kosciolek, Daniel McDonald, Yoshiki Vázquez-Baeza, Gail Ackermann, Jeff DeReus, Stefan Janssen, Austin D. Swafford, Stephanie B. Orchanian, Jon G. Sanders, Joshua Shorenstein, Hannes Holste, Semar Petrus, Adam Robbins-Pianka, Colin J. Brislawn, Mingxun Wang, Jai Ram Rideout, Evan Bolyen, Matthew Dillon, J. Gregory Caporaso, Pieter C. Dorrestein, and Rob Knight. Qiita: rapid, web-enabled microbiome meta-analysis. *Nature Methods*, 15(10):796, 2018.

[62] Brian E. Granger and Fernando Pérez. Jupyter: Thinking and storytelling with code and data. *Computing in Science & Engineering*, 23(2):7–14, 2021.

[63] Andrei Grigoriev. Analyzing genomes with cumulative skew diagrams. *Nucleic Acids Research*, 26(10):2286–2290, 1998.

[64] Nitin Gupta, Nuno Bandeira, Uri Keich, and Pavel A. Pevzner. Target-decoy approach and false discovery rate: when things may go wrong. *Journal of the American Society for Mass Spectrometry*, 22(7):1111–1120, 2011.

[65] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA USA, 2008.

[66] Niina Haiminen, Manfred Klaas, Zeyu Zhou, Filippo Utro, Paul Cormican, Thomas Didion, Christian Sig Jensen, Christopher E. Mason, Susanne Barth, and Laxmi Parida. Comparative exomics of *Phalaris* cultivars under salt stress. *BMC Genomics*, 15(6):1–12, 2014.

[67] Niina Haiminen, Filippo Utro, Ed Seabolt, and Laxmi Parida. Functional profiling of COVID-19 respiratory tract microbiomes. *Scientific Reports*, in press.

[68] Richard W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[69] André Hardy. An examination of procedures for determining the number of clusters in a data set. In *New approaches in classification and data analysis*, pages 178–185. Springer, 1994.

[70] Charles R. Harris, K. Jarrod Millman, Stéfan J. Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[71] Tetsuya Hayashi, Kozo Makino, Makoto Ohnishi, Ken Kurokawa, Kazuo Ishii, Katsushi Yokoyama, Chang-Gyun Han, Eiichi Ohtsubo, Keisuke Nakayama, Takahiro Murata, Masashi Tanaka, Toru Tobe, Tetsuya Iida, Hideto Takami, Takeshi Honda, Chihiro Sasakawa, Naotake Ogasawara, Teruo Yasunaga, Satoru Kuhara, Tadayoshi Shiba, Masahira Hattori, and Hideo Shinagawa. Complete genome sequence of enterohemorrhagic *Eschelichia coli* O157:H7 and genomic comparison with a laboratory strain K-12. *DNA Research*, 8(1):11–22, 2001.

[72] Steven Henikoff and Jorja G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[73] Matthew R. Henn, Christian L. Boutwell, Patrick Charlebois, Niall J. Lennon, Karen A. Power, Alexander R. Macalalad, Aaron M. Berlin, Christine M. Malboeuf, Elizabeth M. Ryan, Sante Gnerre, Michael C. Zody, Rachel L. Erlich, Lisa M. Green, Andrew Berical, Yaoyu Wang, Monica Casali, Hendrik Streeck, Allyson K. Bloom, Tim Dudek, Damien Tully, Ruchi Newman, Karen L. Axten, Adrianne D. Gladden, Laura Battis, Michael Kemper, Qiandong Zeng, Terrance P. Shea, Sharvari Gujja, Carmen Zedlack, Olivier Gasser, Christian Brander, Christoph Hess, Huldrych F. Günthard, Zabrina L. Brumme, Chanson J. Brumme, Suzane Bazner, Jenna Rychert, Jake P. Tinsley, Ken H. Mayer, Eric Rosenberg, Florencia Pereyra, Joshua Z. Levin, Sarah K. Young, Heiko Jessen, Marcus Altfeld, Bruce W. Birren, Bruce D. Walker, and Todd M. Allen. Whole genome deep sequencing of HIV-1 reveals the impact of early minor variants upon immune recognition during acute infection. *PLoS Pathogens*, 8(3):e1002529, 2012.

[74] John R. Hollenbeck and Patrick M. Wright. Harking, Sharking, and Tharking: Making the case *for* post hoc analysis of scientific data. *Journal of Management*, 43(1):5–18, 2017.

[75] Ting Hon, Kristin Mars, Greg Young, Yu-Chih Tsai, Joseph W. Karalius, Jane M. Landolin, Nicholas Maurer, David Kudrna, Michael A. Hardigan, Cynthia C. Steiner, Steven J. Knapp, Doreen Ware, Beth Shapiro, Paul Peluso, and David R. Rank. Highly accurate long-read HiFi sequencing data for five complex genomes. *Scientific Data*, 7(1):399, 2020.

[76] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica journal*, 10(1):37–71, 2005.

[77] Jaime Huerta-Cepas, François Serra, and Peer Bork. ETE 3: reconstruction, analysis, and visualization of phylogenomic data. *Molecular Biology and Evolution*, 33(6):1635–1638, 2016.

[78] John D. Hunter. Matplotlib: a 2D graphics environment. *Computing in Science & Engineering*, 9(3):90, 2007.

[79] Laurence D. Hurst. The Ka/Ks ratio: diagnosing the form of sequence evolution. *Trends in Genetics: TIG*, 18(9):486–486, 2002.

[80] Doug Hyatt, Gwo-Liang Chen, Philip F. LoCascio, Miriam L. Land, Frank W. Larimer, and Loren J. Hauser. Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics*, 11(1):1–11, 2010.

[81] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. *De novo* assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2):226–232, 2012.

[82] Anne C. Jäger, Michelle L. Alvarez, Carey P. Davis, Ernesto Guzmán, Yonmee Han, Lisa Way, Paulina Walichiewicz, David Silva, Nguyen Pham, Glorianna Caves, Jocelyne Bruand, Felix Schlesinger, Stephanie J. K. Pond, Joe Varlaro, Kathryn M. Stephens, and Cydne L. Holt. Developmental validation of the MiSeq FGx forensic genomics system for targeted next generation sequencing in forensic DNA casework and database laboratories. *Forensic Science International: Genetics*, 28:52–70, 2017.

[83] Siddhartha Jaiswal, Pierre Fontanillas, Jason Flannick, Alisa Manning, Peter V. Grauman, Brenton G. Mar, R. Coleman Lindsley, Craig H. Mermel, Noel Burtt, Alejandro Chavez, John M. Higgins, Vladislav Moltchanov, Frank C. Kuo, Michael J. Kluk, Brian Henderson, Leena Kinnunen M.Sc., Heikki A. Koistinen, Claes Ladenvall, Gad Getz, Adolfo Correa, Benjamin F. Banahan, Stacey Gabriel, Sekar Kathiresan, Heather M. Stringham, Mark I. McCarthy, Michael Boehnke, Jaakko Tuomilehto, Christopher Haiman, Leif Groop, Gil Atzmon, James G. Wilson, Donna Neuberg, David Altshuler, and Benjamin L. Ebert. Age-related clonal hematopoiesis associated with adverse outcomes. *New England Journal of Medicine*, 371(26):2488–2498, 2014.

[84] Stefan Janssen, Daniel McDonald, Antonio Gonzalez, Jose A. Navas-Molina, Lingjing Jiang, Zhenjiang Zech Xu, Kevin Winker, Deborah M. Kado, Eric Orwoll, Mark Manary, Siavash Mirarab, and Rob Knight. Phylogenetic placement of exact amplicon sequences improves associations with clinical information. *mSystems*, 3(3):10–1128, 2018.

[85] Tyler A. Joseph, Philippe Chlenski, Aviya Litman, Tal Korem, and Itsik Pe'er. Accurate and robust inference of microbial growth dynamics from metagenomic

sequencing reveals personalized growth rates. *Genome Research*, 32(3):558–568, 2022.

[86] Lukas Käll, John D. Storey, Michael J MacCoss, and William Stafford Noble. Assigning significance to peptides identified by tandem mass spectrometry using decoy databases. *Journal of Proteome Research*, 7(01):29–34, 2008.

[87] Minoru Kanehisa. Enzyme annotation and metabolic reconstruction using KEGG. *Protein Function Prediction: Methods and Protocols*, pages 135–145, 2017.

[88] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In Amihood Amir and Gad M. Landau, editors, *Combinatorial Pattern Matching: 12th Annual Symposium*, volume 2089 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2001.

[89] Kazutaka Katoh and Daron M. Standley. MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Molecular biology and evolution*, 30(4):772–780, 2013.

[90] Uri Keich, Attila Kertesz-Farkas, and William Stafford Noble. Improved false discovery rate estimation procedure for shotgun proteomics. *Journal of Proteome Research*, 14(8):3148–3161, 2015.

[91] Uri Keich, Kaipo Tamura, and William Stafford Noble. Averaging strategy to reduce variability in target-decoy estimates of false discovery rate. *Journal of Proteome Research*, 18(2):585–593, 2018.

[92] Bryce Kille, Yunxi Liu, Nicolae Sapoval, Michael Nute, Lawrence Rauchwerger, Nancy Amato, and Todd J. Treangen. Accelerating SARS-CoV-2 low frequency variant calling on ultra deep sequencing datasets. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 204–208. IEEE, 2021.

[93] Chan Yeong Kim, Junyeong Ma, and Insuk Lee. HiFi metagenomic sequencing enables assembly of accurate and complete genomes from human gut microbiota. *bioRxiv*, 2022.

[94] Motoo Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16(2):111–120, 1980.

[95] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B. Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. Jupyter Notebooks—a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[96] Rob Knight, Peter Maxwell, Amanda Birmingham, Jason Carnes, J. Gregory Caporaso, Brett C. Easton, Michael Eaton, Micah Hamady, Helen Lindsay, Zongzhi Liu, Catherine Lozupone, Daniel McDonald, Michael Robeson, Raymond Sammut, Sandra Smit, Matthew J. Wakefield, Jeremy Widmann, Shandy Wikamn, Stephanie Wilson, Hua Ying, and Gavin A. Huttley. PyCogent: a toolkit for making sense from sequence. *Genome Biology*, 8:1–16, 2007.

[97] Mikhail Kolmogorov, Derek M. Bickhart, Bahar Behsaz, Alexey Gurevich, Mikhail Rayko, Sung Bong Shin, Kristen Kuhn, Jeffrey Yuan, Evgeny Polevikov, Timothy P. L. Smith, and Pavel A. Pevzner. metaFlye: scalable long-read metagenome assembly using repeat graphs. *Nature Methods*, 17(11):1103–1110, 2020.

[98] Mikhail Kolmogorov, Jeffrey Yuan, Yu Lin, and Pavel A. Pevzner. Assembly of long, error-prone reads using repeat graphs. *Nature Biotechnology*, 37(5):540–546, 2019.

[99] Tal Korem, David Zeevi, Jotham Suez, Adina Weinberger, Tali Avnit-Sagi, Maya Pompan-Lotan, Elad Matot, Ghil Jona, Alon Harmelin, Nadav Cohen, Alexandra Sirota-Madi, Christoph A. Thaiss, Meirav Pevsner-Fischer, Rotem Sorek, Ramnik J. Xavier, Eran Elinav, and Eran Segal. Growth dynamics of gut microbiota in health and disease inferred from single metagenomic samples. *Science*, 349(6252):1101–1106, 2015.

[100] Sergey Koren, Brian P. Walenz, Konstantin Berlin, Jason R. Miller, Nicholas H. Bergman, and Adam M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research*, 27(5):722–736, 2017.

[101] Jan Krumsiek, Roland Arnold, and Thomas Rattei. Gepard: a rapid and sensitive tool for creating dotplots on genome scale. *Bioinformatics*, 23(8):1026–1028, 2007.

[102] D. J. Lane. 16S/23S rRNA sequencing. In Erko Stackebrandt and Michael Goodfellow, editors, *Nucleic Acid Techniques in Bacterial Systematics*, pages 115–175. John Wiley and Sons, New York, 1991.

[103] Gregory I. Lang and Andrew W. Murray. Mutation rates across budding yeast chromosome VI are correlated with replication timing. *Genome Biology and Evolution*, 3:799–811, 2011.

[104] Ivica Letunic and Peer Bork. Interactive Tree Of Life (iTOL) v4: recent updates and new developments. *Nucleic Acids Research*, 47(W1):W256–W259, 2019.

[105] Semen A. Leyn, Jaime E. Zlamal, Oleg V. Kurnasov, Xiaoqing Li, Marinela Elane, Lourdes Myjak, Mikolaj Godzik, Alban de Crecy, Fernando Garcia-Alcalde, Martin Ebeling, and Andrei L. Osterman. Experimental evolution in morbidostat reveals converging genomic trajectories on the path to triclosan resistance. *Microbial Genomics*, 7(5):000553, 2021.

[106] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[107] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[108] Jean R. Lobry. Asymmetric substitution patterns in the two DNA strands of bacteria. *Molecular biology and evolution*, 13(5):660–665, 1996.

[109] Michael I. Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, 15(12):550, 2014.

[110] Catherine Lozupone and Rob Knight. UniFrac: a new phylogenetic method for comparing microbial communities. *Applied and Environmental Microbiology*, 71(12):8228–8235, 2005.

[111] Chengwei Luo, Rob Knight, Heli Siljander, Mikael Knip, Ramnik J. Xavier, and Dirk Gevers. Constrains identifies microbial strains in metagenomic datasets. *Nature Biotechnology*, 33(10):1045–1052, 2015.

[112] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[113] Siddhartha Mandal, Will Van Treuren, Richard A. White, Merete Eggesbø, Rob Knight, and Shyamal D. Peddada. Analysis of composition of microbiomes: a novel method for studying microbial composition. *Microbial Ecology in Health and Disease*, 26(1):27663, 2015.

[114] Dipankar Manna, Adam M. Breier, and N Patrick Higgins. Microarray analysis of transposition targets in *Escherichia coli*: the impact of transcription. *Proceedings of the National Academy of Sciences*, 101(26):9780–9785, 2004.

[115] Giovanni Manzini. Two space saving tricks for linear time LCP array computation. In *Scandinavian Workshop on Algorithm Theory*, pages 372–383. Springer, 2004.

[116] Jaswinder Singh Maras, Shvetank Sharma, Adil Bhat, Reshu Aggrawal, Ekta Gupta, and Shiv K. Sarin. Multi-omics integration analysis of respiratory specimen characterizes baseline molecular determinants associated with COVID-19 diagnosis. *MedRxiv*, pages 2020–07, 2020.

[117] Clarisse Marotz, Rebecca Molinsky, Cameron Martino, Bruno Bohn, Sumith Roy, Michael Rosenbaum, Moïse Desvarieux, Melana Yuzefpolskaya, Bruce J. Paster, David R. Jacobs, Paolo C. Colombo, Panos N. Papapanou, Rob Knight, and Ryan T. Demmer. Early microbial markers of periodontal and cardiometabolic diseases in ORIGINS. *npj Biofilms and Microbiomes*, 8(1):30, 2022.

[118] Cameron Martino, James T. Morton, Clarisse A. Marotz, Luke R. Thompson, Anupriya Tripathi, Rob Knight, and Karsten Zengler. A Novel Sparse Compositional Technique Reveals Microbial Perturbations. *mSystems*, 4(1):e00016–19, 2019.

[119] Ann M. Mc Cartney, Kishwar Shafin, Michael Alonge, Andrey V. Bzikadze, Giulio Formenti, Arkarachai Fungtammasan, Kerstin Howe, Chirag Jain, Sergey Koren, Glennis A. Logsdon, Karen H. Miga, Alla Mikheenko, Benedict Paten, Alaina Shumate, Daniela C. Soto, Ivan Sović, Jonathan M. D. Wood, Justin M. Zook, Adam M. Phillippy, and Arang Rhie. Chasing perfection: validation and polishing strategies for telomere-to-telomere genome assemblies. *Nature Methods*, pages 1–9, 2022.

[120] Daniel McDonald, Jose C. Clemente, Justin Kuczynski, Jai Ram Rideout, Jesse Stombaugh, Doug Wendel, Andreas Wilke, Susan Huse, John Hufnagle, Folker Meyer, Rob Knight, and J. Gregory Caporaso. The Biological Observation Matrix (BIOM) format or: how I learned to stop worrying and love the ome-ome. *GigaScience*, 1(1):7, 2012.

[121] Daniel McDonald, Morgan N. Price, Julia Goodrich, Eric P. Nawrocki, Todd Z. DeSantis, Alexander Probst, Gary L. Andersen, Rob Knight, and Philip Hugenholtz. An improved greengenes taxonomy with explicit ranks for ecological and evolutionary analyses of bacteria and archaea. *The ISME journal*, 6(3):610–618, 2012.

[122] Daniel McDonald, Yoshiki Vázquez-Baeza, David Koslicki, Jason McClelland, Nicolai Reeve, Zhenjiang Xu, Antonio Gonzalez, and Rob Knight. Striped UniFrac: enabling microbiome analysis at unprecedented scale. *Nature Methods*, 15(11):847–848, 2018.

[123] Michael R. McLaren, Amy D. Willis, and Benjamin J. Callahan. Consistent and correctable bias in metagenomic sequencing experiments. *eLife*, 8:e46923, 2019.

[124] Peter Menzel, Kim Lee Ng, and Anders Krogh. Fast and sensitive taxonomic classification for metagenomics with kaiju. *Nature Communications*, 7(1):1–9, 2016.

[125] Jason R. Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010.

[126] Jeremiah J. Minich, Semar Petrus, Julius D. Michael, Todd P. Michael, Rob Knight, and Eric E. Allen. Temporal, environmental, and biological drivers of the mucosal microbiome in a wild marine fish, *Scomber japonicus*. *bioRxiv*, page 721555, 2019.

[127] Jeremiah J. Minich, Qiyun Zhu, Stefan Janssen, Ryan Hendrickson, Amnon Amir, Russ Vetter, John Hyde, Megan M. Doty, Kristina Stillwell, James Benardini, Jae H. Kim, Eric E. Allen, Kasthuri Venkateswaran, and Rob Knight. KatharoSeq enables high-throughput microbiome analysis from low-biomass samples. *mSystems*, 3(3):e00218–17, 2018.

[128] Siavash Mirarab, Nam Nguyen, and Tandy Warnow. SEPP: SATé-enabled phylogenetic placement. In *Biocomputing 2012*, pages 247–258. World Scientific, 2012.

[129] Roger E. Moore, Mary K. Young, and Terry D. Lee. Qscore: an algorithm for evaluating SEQUEST database search results. *Journal of the American Society for Mass Spectrometry*, 13(4):378–386, 2002.

[130] Aleksandr Morgulis, George Coulouris, Yan Raytselis, Thomas L. Madden, Richa Agarwala, and Alejandro A. Schäffer. Database indexing for production megablast searches. *Bioinformatics*, 24(16):1757–1764, 2008.

[131] Matthew Mort, Dobril Ivanov, David N. Cooper, and Nadia A. Chuzhanova. A meta-analysis of nonsense mutations causing human genetic disease. *Human mutation*, 29(8):1037–1047, 2008.

[132] James T. Morton, Clarisse Marotz, Alex Washburne, Justin Silverman, Livia S. Zaramela, Anna Edlund, Karsten Zengler, and Rob Knight. Establishing microbial composition measurement standards with reference frames. *Nature Communications*, 10(1):2719, June 2019.

[133] James T. Morton, Jon Sanders, Robert A. Quinn, Daniel McDonald, Antonio Gonzalez, Yoshiki Vázquez-Baeza, Jose A. Navas-Molina, Se Jin Song, Jessica L. Metcalf, Embriette R. Hyde, Manuel Lladser, Pieter C. Dorrestein, and Rob Knight. Balance trees reveal microbial niche differentiation. *mSystems*, 2(1):e00162–16, 2017.

[134] Eugene W. Myers. HISim, 2021. https://github.com/thegenemyers/HI.SIM.

[135] Kensuke Nakamura, Taku Oshima, Takuya Morimoto, Shun Ikeda, Hirofumi Yoshikawa, Yuh Shiwa, Shu Ishikawa, Margaret C. Linak, Aki Hirai, Hiroki Takahashi, Md. Altaf-Ul-Amin, Naotake Ogasawara, and Shigehiko Kanaya. Sequence-specific error profile of Illumina sequencers. *Nucleic Acids Research*, 39(13):e90–e90, 2011.

[136] Marta Nascimento, Adriano Sousa, Mário Ramirez, Alexandre P. Francisco, João A. Carriço, and Cátia Vaz. PHYLOViZ 2.0: providing scalable data integration and visualization for multiple phylogenetic inference methods. *Bioinformatics*, 33(1):128–129, 2017.

[137] Joseph I. Naus. Approximations for distributions of scan statistics. *Journal of the American Statistical Association*, 77(377):177–183, 1982.

[138] Samuel M. Nicholls, Wayne Aubrey, Kurt De Grave, Leander Schietgat, Christopher J. Creevey, and Amanda Clare. On the complexity of haplotyping a microbial community. *Bioinformatics*, 37(10):1360–1366, 01 2021.

[139] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V. Bzikadze, Alla Mikheenko, Mitchell R. Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, Sergey Aganezov, Savannah J. Hoyt, Mark Diekhans, Glennis A. Logsdon, Michael

Alonge, Stylianos E. Antonarakis, Matthew Borchers, Gerard G. Bouffard, Shelise Y. Brooks, Gina V. Caldas, Nae-Chyun Chen, Haoyu Cheng, Chen-Shan Chin, William Chow, Leonardo G. de Lima, Philip C. Dishuck, Richard Durbin, Tatiana Dvorkina, Ian T. Fiddes, Giulio Formenti, Robert S. Fulton, Arkarachai Fungtammasan, Erik Garrison, Patrick G. S. Grady, Tina A. Graves-Lindsay, Ira M. Hall, Nancy F. Hansen, Gabrielle A. Hartley, Marina Haukness, Kerstin Howe, Michael W. Hunkapiller, Chirag Jain, Miten Jain, Erich D. Jarvis, Peter Kerpedjiev, Melanie Kirsche, Mikhail Kolmogorov, Jonas Korlach, Milinn Kremitzki, Heng Li, Valerie V. Maduro, Tobias Marschall, Ann M. McCartney, Jennifer McDaniel, Danny E. Miller, James C. Mullikin, Eugene W. Myers, Nathan D. Olson, Benedict Paten, Paul Peluso, Pavel A. Pevzner, David Porubsky, Tamara Potapova, Evgeny I. Rogaev, Jeffrey A. Rosenfeld, Steven L. Salzberg, Valerie A. Schneider, Fritz J. Sedlazeck, Kishwar Shafin, Colin J. Shew, Alaina Shumate, Ying Sims, Arian F. A. Smit, Daniela C. Soto, Ivan Sović, Jessica M. Storer, Aaron Streets, Beth A. Sullivan, Françoise Thibaud-Nissen, James Torrance, Justin Wagner, Brian P. Walenz, Aaron Wenger, Jonathan M. D. Wood, Chunlin Xiao, Stephanie M. Yan, Alice C. Young, Samantha Zarate, Urvashi Surti, Rajiv C. McCoy, Megan Y. Dennis, Ivan A. Alexandrov, Jennifer L. Gerton, Rachel J. O'Neill, Winston Timp, Justin M. Zook, Michael C. Schatz, Evan E. Eichler, Karen H. Miga, and Adam M. Phillippy. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022.

[140] Sergey Nurk, Dmitry Meleshko, Anton Korobeynikov, and Pavel A. Pevzner. metaSPAdes: a new versatile metagenomic assembler. *Genome Research*, 27(5):824–834, 2017.

[141] Sergey Nurk, Brian P. Walenz, Arang Rhie, Mitchell R. Vollger, Glennis A. Logsdon, Robert Grothe, Karen H. Miga, Evan E. Eichler, Adam M. Phillippy, and Sergey Koren. Hicanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. *Genome Research*, 30(9):1291–1305, 2020.

[142] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. PBSIM2: a simulator for long-read sequencers with a novel generative model of quality scores. *Bioinformatics*, 37(5):589–595, 2021.

[143] pandas development team. pandas-dev/pandas: Pandas 1.2.3, March 2021.

[144] Alma E. Parada, David M. Needham, and Jed A. Fuhrman. Every base matters: assessing small subunit rRNA primers for marine microbiomes with mock communities, time series and global field samples. *Environmental Microbiology*, 18(5):1403–1414, 2016.

[145] Donovan H. Parks, Michael Imelfort, Connor T. Skennerton, Philip Hugenholtz, and Gene W. Tyson. CheckM: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes. *Genome Research*, 25(7):1043–1055, 2015.

[146] Fabian Pedregosa and Phillippe Gervais. memory_profiler. https://github.com/pythonprofilers/memory_profiler, 2022.

[147] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[148] Pavel Pevzner and Glenn Tesler. Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. *Genome Research*, 13(1):37–45, 2003.

[149] Pavel A. Pevzner, Haixu Tang, and Glenn Tesler. De novo repeat classification and fragment assembly. *Genome Research*, 14:1786–1796, 2004.

[150] Meg Pirrung, Ryan Kennedy, J. Gregory Caporaso, Jesse Stombaugh, Doug Wendel, and Rob Knight. TopiaryExplorer: visualizing large phylogenetic trees with environmental metadata. *Bioinformatics*, 27(21):3067–3069, 2011.

[151] Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T. Afshar, Sam S. Gross, Lizzie Dorfman, Cory Y. McLean, and Mark A. DePristo. A universal SNP and small-indel variant caller using deep neural networks. *Nature Biotechnology*, 36(10):983–987, 2018.

[152] Zoe A. Pratte, Marc Besson, Rebecca D. Hollman, and Frank J. Stewart. The Gills of Reef Fish Support a Distinct Microbiome Influenced by Host-Specific Factors. *Applied and Environmental Microbiology*, 84(9):e00063–18, February 2018.

[153] Morgan N. Price, Paramvir S. Dehal, and Adam P. Arkin. FastTree 2–approximately maximum-likelihood trees for large alignments. *PLoS One*, 5(3):e9490, 2010.

[154] Christian Quast, Elmar Pruesse, Pelin Yilmaz, Jan Gerken, Timmy Schweer, Pablo Yarza, Jörg Peplies, and Frank Oliver Glöckner. The SILVA ribosomal RNA gene database project: improved data processing and web-based tools. *Nucleic Acids Research*, 41(D1):D590–D596, 2012.

[155] Christopher Quince, Tom O. Delmont, Sébastien Raguideau, Johannes Alneberg, Aaron E Darling, Gavin Collins, and A. Murat Eren. DESMAN: a new tool for de novo extraction of strains from metagenomes. *Genome Biology*, 18(1):1–22, 2017.

[156] Aaron R. Quinlan and Ira M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 2010.

[157] Thomas Quinn and Ionas Erb. Amalgams: data-driven amalgamation for the reference-free dimensionality reduction of zero-laden compositional data. *bioRxiv*, 2020.

[158] Andrew Rambaut. FigTree. http://tree.bio.ed.ac.uk/software/figtree/.

[159] Jeff Reback, Wes McKinney, jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, gfyoung, Sinhrks, Simon Hawkins, Adam Klein, Matthew Roeschke, Jeff Tratner, Chang She, Terji Petersen, William Ayd, MomIsBestFriend, Marc Garcia, Jeremy Schendel, Andy Hayden, Vytautas Jancauskas, Daniel Saxton, Ali McMaster, Pietro Battiston, Skipper Seabold, chris-b1, h-vetinari, Stephan Hoyer, Kaiqi Dong, Wouter Overmeire, and Martin Winkel. pandas-dev/pandas: Pandas 1.1.2, September 2020.

[160] Daniel C. Richter, Felix Ott, Alexander F. Auch, Ramona Schmid, and Daniel H. Huson. MetaSim—a sequencing simulator for genomics and metagenomics. *PLoS One*, 3(10):e3373, 2008.

[161] Monica Riley, Takashi Abe, Martha B. Arnaud, Mary KB. Berlyn, Frederick R. Blattner, Roy R. Chaudhuri, Jeremy D Glasner, Takashi Horiuchi, Ingrid M. Keseler, Takehide Kosuge, Hirotada Mori, Nicole T. Perna, Guy Plunket III, Kenneth E Rudd, Margrethe H. Serres, Gavin H. Thomas, Nicholas R. Thomson, David Wishart, and Barry L. Wanner. *Escherichia coli* K-12: a cooperatively developed annotation snapshot—2005. *Nucleic Acids Research*, 34(1):1–9, 2006.

[162] Javier Rivera-Pinto, Juan Jose Egozcue, Vera Pawlowsky-Glahn, Raul Paredes, Marc Noguera-Julian, and M. Luz Calle. Balances: a new perspective for microbiome analysis. *mSystems*, 3(4):10–1128, 2018.

[163] Peter Rugbjerg and Morten O. A. Sommer. Overcoming genetic heterogeneity in industrial fermentations. *Nature Biotechnology*, 37(8):869–876, 2019.

[164] Jesse J. Salk, Michael W. Schmitt, and Lawrence A. Loeb. Enhancing the accuracy of next-generation sequencing for detecting rare and subclonal mutations. *Nature Reviews Genetics*, 19(5):269, 2018.

[165] Sarah Sandmann, Aniek O. De Graaf, Mohsen Karimi, Bert A. Van Der Reijden, Eva Hellström-Lindberg, Joop H. Jansen, and Martin Dugas. Evaluating variant calling tools for non-matched next-generation sequencing data. *Scientific Reports*, 7(1):1–12, 2017.

[166] Nicolae Sapoval, Amirali Aghazadeh, Michael G. Nute, Dinler A. Antunes, Advait Balaji, Richard Baraniuk, C. J. Barberan, Ruth Dannenfelser, Chen Dun, Mohammadamin Edrisi, R. A. Leo Elworth, Bryce Kille, Anastasios Kyrillidis, Luay Nakhleh, Cameron R. Wolfe, Zhi Yan, Vicky Yao, and Todd J. Treangen. Current progress and open challenges for applying deep learning across the biosciences. *Nature Communications*, 13(1):1–12, 2022.

[167] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-Lite: a grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2016.

[168] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. Reactive Vega: a streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, 2015.

[169] David J. Scanlan, Martin Ostrowski, Sophie Mazard, Alexis Dufresne, Laurence Garczarek, Wolfgang R. Hess, Anton F. Post, Martin Hagemann, I. Paulsen, and Frédéric Partensky. Ecological genomics of marine picocyanobacteria. *Microbiol. Mol. Biol. Rev.*, 73(2):249–299, 2009.

[170] Patrick D. Schloss. Identifying and overcoming threats to reproducibility, replicability, robustness, and generalizability in microbiome research. *mBio*, 9(3):10–1128, 2018.

[171] Michael W. Schmitt, Scott R. Kennedy, Jesse J Salk, Edward J. Fox, Joseph B. Hiatt, and Lawrence A. Loeb. Detection of ultra-rare mutations by next-generation sequencing. *Proceedings of the National Academy of Sciences*, 109(36):14508–14513, 2012.

[172] Edward E. Seabolt, Gowri Nayar, Harsha Krishnareddy, Akshay Agarwal, Kristen L. Beck, Ignacio Terrizzano, Eser Kandogan, Mark Kunitomi, Mary Roth, Vandana Mukherjee, and James H. Kaufman. Functional genomics platform, a cloud-based platform for studying microbial life at scale. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(2):940–952, 2020.

[173] Torsten Seemann. barrnap 0.9: rapid ribosomal RNA prediction, 2018. https://github.com/tseemann/barrnap.

[174] Kathrin M. Seibt, Thomas Schmidt, and Tony Heitkam. FlexiDot: highly customizable, ambiguity-aware dotplots for visual sequence analyses. *Bioinformatics*, 34(20):3575–3577, 2018.

[175] Agnieszka Sekowska, Sofie Wendel, Emil C. Fischer, Morten H. H. Nørholm, and Antoine Danchin. Generation of mutation hotspots in ageing bacterial colonies. *Scientific Reports*, 6(1):1–7, 2016.

[176] Mantas Sereika, Rasmus Hansen Kirkegaard, Søren Michael Karst, Thomas Yssing Michaelsen, Emil Aarre Sørensen, Rasmus Dam Wollenberg, and Mads Albertsen. Oxford nanopore r10.4 long-read sequencing enables the generation of near-finished bacterial genomes from pure cultures and metagenomes without short-read or reference polishing. *Nature Methods*, 19(7):823–826, 2022.

[177] Zijie Shen, Yan Xiao, Lu Kang, Wentai Ma, Leisheng Shi, Li Zhang, Zhuo Zhou, Jing Yang, Jiaxin Zhong, Donghong Yang, Li Guo, Guoliang Zhang, Hongru Li, Yu Xu, Mingwei Chen, Zhancheng Gao, Jianwei Wang, Lili Ren, and Mingkun Li. Genomic diversity of severe acute respiratory syndrome–coronavirus 2 in patients with coronavirus disease 2019. *Clinical Infectious Diseases*, 71(15):713–720, 2020.

[178] Justin D. Silverman, Alex D. Washburne, Sayan Mukherjee, and Lawrence A. David. A phylogenetic transform enhances analysis of compositional microbiota data. *eLife*, 6:e21887, 2017.

[179] Nathaniel Smith and Stéfan van der Walt. A better default colormap for matplotlib. SciPy 2015, 2015. https://www.youtube.com/watch?v=xAoljeRJ3lU.

[180] T. M. Sonneborn. Degeneracy of the genetic code: extent, nature, and genetic implications. In *Evolving Genes and Proteins*, pages 377–397. Elsevier, 1965.

[181] Martin Šošić and Mile Šikić. Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 33(9):1394–1395, 2017.

[182] Sara Steegen, Francis Tuerlinckx, Andrew Gelman, and Wolf Vanpaemel. Increasing transparency through a multiverse analysis. *Perspectives on Psychological Science*, 11(5):702–712, 2016.

[183] John R. Stevens, Todd R. Jones, Michael Lefevre, Balasubramanian Ganesan, and Bart C. Weimer. SigTree: a microbial community analysis tool to identify and visualize significantly responsive branches in a phylogenetic tree. *Computational and structural biotechnology journal*, 15:372–378, 2017.

[184] Antonius Suwanto and Samuel Kaplan. Physical and genetic mapping of the *Rhodobacter sphaeroides* 2.4.1 genome: presence of two unique circular chromosomes. *Journal of Bacteriology*, 171(11):5850–5859, 1989.

[185] Yoshihiko Suzuki and Gene Myers. Accurate k-mer Classification Using Read Profiles. In Christina Boucher and Sven Rahmann, editors, *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*, volume 242 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[186] Alexander P. Sweeten, Michael C. Schatz, and Adam M. Phillippy. ModDotPlot—rapid and interactive visualization of tandem repeats. *Bioinformatics*, 40(8):btae493, 2024.

[187] Ammar Tareen and Justin B. Kinney. Logomaker: beautiful sequence logos in Python. *Bioinformatics*, 36(7):2272–2274, 2020.

[188] The SAM/BAM Format Specification Working Group. Sequence alignment/map format specification, 2022. https://samtools.github.io/hts-specs/SAMv1.pdf.

[189] Luke R. Thompson, Jon G. Sanders, Daniel McDonald, Amnon Amir, Joshua Ladau, Kenneth J. Locey, Robert J. Prill, Anupriya Tripathi, Sean M. Gibbons, Gail Ackermann, Jose A. Navas-Molina, Stefan Janssen, Evguenia Kopylova, Yoshiki Vázquez-Baeza, Antonio González, James T. Morton, Siavash Mirarab, Zhenjiang Zech Xu, Lingjing Jiang, Mohamed F. Haroon, Jad Kanbar, Qiyun Zhu, Se Jin Song,

Tomasz Kosciolek, Nicholas A. Bokulich, Joshua Lefler, Colin J. Brislawn, Gregory Humphrey, Sarah M. Owens, Jarrad Hampton-Marcell, Donna Berg-Lyons, Valerie McKenzie, Noah Fierer, Jed A. Furhman, Aaron Clauset, Rick L. Stevens, Ashley Shade, Katherine S. Pollard, Kelly D. Goodwin, Janet K. Jansson, Jack A. Gilbert, Rob Knight, and The Earth Microbiome Project Consortium. A communal catalogue reveals earth's multiscale microbial diversity. *Nature*, 551(7681):457–463, 2017.

[190] Helga Thorvaldsdóttir, James T. Robinson, and Jill P. Mesirov. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Briefings in Bioinformatics*, 14(2):178–192, 2013.

[191] Armin Töpfer. Juliet - one click minor variant calling, 2017. https://www.pacb.com/wp-content/uploads/4May2017_ArminToepfer_JulietMinorVariantCalling.pdf.

[192] Erdal Toprak, Adrian Veres, Jean-Baptiste Michel, Remy Chait, Daniel L. Hartl, and Roy Kishony. Evolutionary paths to antibiotic resistance under dynamically sustained drug selection. *Nature Genetics*, 44(1):101–105, 2012.

[193] Florian Trigodet, Rohan Sachdeva, Jillian F. Banfield, and A. Murat Eren. Assemblies of long-read metagenomes suffer from diverse errors. *bioRxiv*, 2025.

[194] Anupriya Tripathi, Yoshiki Vázquez-Baeza, Julia M. Gauglitz, Mingxun Wang, Kai Dührkop, Mélissa Nothias-Esposito, Deepa D. Acharya, Madeleine Ernst, Justin J. J. van der Hooft, Qiyun Zhu, Daniel McDonald, Asker D. Brejnrod, Antonio Gonzalez, Jo Handelsman, Markus Fleischauer, Marcus Ludwig, Sebastian Böcker, Louis-Félix Nothias, Rob Knight, and Pieter C. Dorrestein. Chemically informed analyses of metabolomics mass spectrometry data with Qemistree. *Nature Chemical Biology*, 17(2):146–151, 2021.

[195] Filippo Utro, Niina Haiminen, Enrico Siragusa, Laura-Jayne Gardiner, Ed Seabolt, Ritesh Krishna, James H. Kaufman, and Laxmi Parida. Hierarchically labeled database indexing allows scalable characterization of microbiomes. *iScience*, 23(4), 2020.

[196] Stefan Van Der Walt, S. Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[197] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. Altair: Interactive Statistical Visualizations for Python. *The Journal of Open Source Software*, 3:1057, 2018.

[198] Sotirios Vasileiadis, Edoardo Puglisi, Maria Arena, Fabrizio Cappa, Pier S. Cocconcelli, and Marco Trevisan. Soil bacterial diversity screening using single 16S rRNA gene V regions coupled with multi-million read generating sequencing technologies. *PLoS One*, 7(8):e42671, 2012.

[199] Yoshiki Vázquez-Baeza, Antonio Gonzalez, Larry Smarr, Daniel McDonald, James T. Morton, Jose A. Navas-Molina, and Rob Knight. Bringing the dynamic microbiome to life with animations. *Cell Host & Microbe*, 21(1):7–10, 2017.

[200] Yoshiki Vázquez-Baeza, Meg Pirrung, Antonio Gonzalez, and Rob Knight. EMPeror: a tool for visualizing high-throughput microbial community data. *GigaScience*, 2(1):2047–217X, 2013.

[201] Bie M. P. Verbist, Kim Thys, Joke Reumers, Yves Wetzels, Koen Van der Borght, Willem Talloen, Jeroen Aerssens, Lieven Clement, and Olivier Thas. VirVarSeq: a low-frequency virus variant detection pipeline for Illumina sequencing using adaptive base-calling accuracy filtering. *Bioinformatics*, 31(1):94–101, 2015.

[202] Riccardo Vicedomini, Christopher Quince, Aaron E. Darling, and Rayan Chikhi. Strainberry: automated strain separation in low-complexity metagenomes using long reads. *Nature Communications*, 12(1):4485, 2021.

[203] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020.

[204] F. Vogel. Non-randomness of base replacement in point mutation. *Journal of Molecular Evolution*, 1(4):334–367, 1972.

[205] Bo Wang, Lin Wan, Anqi Wang, and Lei M. Li. An adaptive decorrelation method removes Illumina DNA base-calling errors caused by crosstalk between adjacent clusters. *Scientific Reports*, 7(1):1–11, 2017.

[206] Jing Wang, Leon Raskin, David C. Samuels, Yu Shyr, and Yan Guo. Genome measures used for quality control are dependent on gene function and ancestry. *Bioinformatics*, 31(3):318–323, 2015.

[207] Alex D. Washburne, Justin D. Silverman, Jonathan W. Leff, Dominic J. Bennett, John L. Darcy, Sayan Mukherjee, Noah Fierer, and Lawrence A. David. Phylogenetic factorization of compositional data yields lineage-level associations in microbiome datasets. *PeerJ*, 5:e2969, 2017.

[208] Michael Waskom, Olga Botvinnik, Joel Ostblom, Saulius Lukauskas, Paul Hobson, Maoz Gelbart, David C. Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel

Martin, Kyle Meyer, Corban Swain, Alistair Miles, Thomas Brunner, Drew O'Kane, Tal Yarkoni, Mike Lee Williams, and Constantine Evans. mwaskom/seaborn: v0.10.0 (january 2020), January 2020.

[209] Zhi Wei, Wei Wang, Pingzhao Hu, Gholson J. Lyon, and Hakon Hakonarson. SNVer: a statistical tool for variant calling in analysis of pooled or individual next-generation sequencing data. *Nucleic Acids Research*, 39(19):e132–e132, 2011.

[210] Aaron M. Wenger, Paul Peluso, William J. Rowell, Pi-Chuan Chang, Richard J. Hall, Gregory T. Concepcion, Jana Ebler, Arkarachai Fungtammasan, Alexey Kolesnikov, Nathan D. Olson, Armin Töpfer, Michael Alonge, Medhat Mahmoud, Yufeng Qian, Chen-Shan Chin, Adam M. Phillippy, Michael C. Schatz, Gene Myers, Mark A. DePristo, Jue Ruan, Tobias Marschall, Fritz J. Sedlazeck, Justin M. Zook, Heng Li, Sergey Koren, Andrew Carroll, David R. Rank, and Michael W. Hunkapiller. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature Biotechnology*, 37(10):1155–1162, 2019.

[211] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56–61, 2010.

[212] Ryan R. Wick, Mark B. Schultz, Justin Zobel, and Kathryn E. Holt. Bandage: interactive visualization of *de novo* genome assemblies. *Bioinformatics*, 31(20):3350–3352, 2015.

[213] Andreas Wilm, Pauline Poh Kim Aw, Denis Bertrand, Grace Hui Ting Yeo, Swee Hoe Ong, Chang Hua Wong, Chiea Chuen Khor, Rosemary Petric, Martin Lloyd Hibberd, and Niranjan Nagarajan. LoFreq: a sequence-quality aware, ultra-sensitive variant caller for uncovering cell-population heterogeneity from high-throughput sequencing datasets. *Nucleic Acids Research*, 40(22):11189–11201, 2012.

[214] Elizabeth A. Winzeler, Daniel D. Shoemaker, Anna Astromoff, Hong Liang, Keith Anderson, Bruno Andre, Rhonda Bangham, Rocio Benito, Jef D. Boeke, Howard Bussey, Angela M. Chu, Carla Connelly, Karen Davis, Fred Dietrich, Sally Whelen Dow, Mohamed El Bakkoury, Françoise Foury, Stephen H. Friend, Erik Gentalen, Guri Giaever, Johannes H. Hegemann, Ted Jones, Michael Laub, Hong Liao, Nicole Liebundguth, David J. Lockhart, Anca Lucau-Danila, Marc Lussier, Nasiha M'Rabet, Patrice Menard, Michael Mittmann, Chai Pai, Corinne Rebischung, Jose L. Revuelta, Christopher J. Roberts, Petra Ross-MacDonald, Bart Scherens, Michael Snyder, Sharon Sookhai-Mahadeo, Reginald K. Storms, Steeve Véronneau, Marleen Voet, Guido Volckaert, Teresa R. Ward, Robert Wysocki, Grace S. Yen, Kexin Yu, Katja Zimmermann, Peter Philippsen, Mark Johnston, and Ronald W. Davis. Functional characterization of the *S. cerevisiae* genome by gene deletion and parallel analysis. *Science*, 285(5429):901–906, 1999.

[215] Zhiyong Xu, Dimitrios Zikos, Nikolaus Osterrieder, and B. Karsten Tischer. Generation of a complete single-gene knockout bacterial artificial chromosome library of cowpox virus and identification of its essential genes. *Journal of Virology*, 88(1):490, 2014.

[216] Guangchuang Yu. Using ggtree to visualize data on tree-like structures. *Current Protocols in Bioinformatics*, 69(1):e96, 2020.

[217] Yan Zhang, Fan Jiang, Boyuan Yang, Sen Wang, Hengchao Wang, Anqi Wang, Dong Xu, and Wei Fan. Improved microbial genomes and gene catalog of the chicken gut from metagenomic sequencing of high-fidelity long reads. *GigaScience*, 11:giac116, 2022.

[218] Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb Miller. A greedy algorithm for aligning DNA sequences. *Journal of Computational Biology*, 7(1-2):203–214, 2000.

[219] Zhenmiao Zhang, Ishaan Gupta, and Pavel A. Pevzner. GenomeDecoder: Inferring segmental duplications in highly-repetitive genomic regions. *Bioinformatics*, 41(2):btaf058, 2025.

[220] Jaime E. Zlamal, Semen A. Leyn, Mallika Iyer, Marinela L. Elane, Nicholas A. Wong, James W. Wamsley, Maarten Vercruysse, Fernando Garcia-Alcalde, and Andrei L. Osterman. Shared and unique evolutionary trajectories to ciprofloxacin resistance in gram-negative bacterial pathogens. *mBio*, 12(3):e00987–21, 2021.