ABSTRACT

Title of Dissertation: ANALYZING AND INDEXING HUGE REFERENCE SEQUENCE COLLECTIONS

Jason Fan
Doctor of Philosophy, 2023

Dissertation directed by: Rob Patro, Associate Professor, Department of Computer Science

Recent advancements in genome-scale assays and high throughput sequencing have made systematic measurement of model-organisms both accessible and abundant. As a result, novel algorithms that exploit similarities across multiple samples or compare measurements against multiple reference organisms have been designed to improve analyses and gain new insights. However, such models and algorithms can be difficult to apply in practice. Furthermore, analysis of high-throughput sequencing data across multiple samples and multiple reference genomic sequences can be prohibitively costly in terms of space and time. In three parts, this dissertation investigates novel computational techniques that improve analyses at various scales.

In Part I, I present two general matrix-factorization algorithms designed to analyze and compare biological measurements of related species that can be summarized as networks. In Part II, I present methods that improve analyses of high-throughput sequencing data. The first method, ScalpelSig, reduces the computation burden of applying mutational signature analysis in resource limited settings; and the second method, a derivation of *perplexity* for gene and transcript expres-

sion estimation models, enables effective model selection in experimental RNA-seq data where ground-truth is absent.

In Part III, I tackle the difficulties of indexing and analyzing huge collections reference sequences. I introduce the *spectrum preserving tiling* (SPT), a new computational and mathematical abstraction. Mathematically, the SPT explicitly relates past work on compactly representing k-mer sets — namely the compacted de Bruijn graph and recent derivations of spectrum preserving string sets — to the indexing of k-mer positions and metadata in reference sequences. Computationally, the SPT makes possible an entire class of efficient and modular k-mer indexes. I introduce a pair of indexing schemes respectively designed to efficiently support rapid locate and k-mer "color" queries in small space. In the final Chapter of this dissertation, I show how these modular indexes can be effectively and *generically* implemented.

ANALYZING AND INDEXING HUGE REFERENCE SEQUENCE COLLECTIONS

by

Jason Fan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Table of Contents

Chapter 1: Introduction

At the core of computational biology is the *genotype-to-phenotype* problem that poses the following question. Given some biological measurement over genomic sequence — the *genotype* — how do they cause the observable characteristics — the *phenotype* — of a measured biological system, model organism, or individual?

The thirteen-year long effort to systematically sequence the human genome by the Human Genome Project (HGP) to understand the genotype that *is* the human species[1] is one example of the monumental biological and computational effort that has been made to tackle the genotype-to-phenotype problem [1]. Since HGP's first draft of the human genome sequence, the advent of *high-throughput* sequencing technologies for both DNA and transcribed RNA sequences has allowed researchers to interrogate increasingly complex biological systems and phenotypes of interest. These high-throughput technologies have enabled scientists to systematically measure biological systems at genome and transcriptome scale — over *all* annotated genes and isoforms — even at the resolution of *single cells*.

Scientific desire to mechanistically understand increasingly complicated, nuanced and even rare phenotypes has spurred rapid development of high-throughput measurement techniques. An overwhelming amount of raw sequencing data has been generated since. For example, to understand human tissues, the Genotype-Tissue Expression (GTEx) project measured RNA-seq, DNA-seq and

[1]Or rather, a handful of individuals.

molecular assay measurements in more than 50 tissues across nearly 1000 individuals [2]. To better understand, treat and diagnose cancer, The Cancer Genome Atlas program (TCGA) has measured “over 2.5 petabytes of genomic, epigenomic, transcriptomic, and proteomic data … over 20,000 primary cancer and matched normal samples spanning 33 cancer types” [3].

Relating, correlating, and explaining how measured *petabytes* of sequencing data cause observed phenotypes is and will remain challenging. The crux is that an enormous gap between biological abstractions of vastly different scales — from deceivingly simple molecular spellings of nucleotides to complex phenotypes in biological systems like tissues and model organisms — need to be navigated and mechanistically explained. So, to infer and discover new biological insights from a dizzying overabundance of data, computational biologists build fast, space-efficient and (hopefully) accurate algorithms. In three parts, this dissertation develops a variety of computational methods to improve analyses at various scales, eventually culminating in state-of-the-art work in efficient indexing and analysis of huge reference genome collections.

Part I — Networks.

Understanding the phenotype-to-genotype problem from *sequences* alone can be difficult because an overabundance of raw sequencing measurements can be difficult to interpret. As a result, computational biologists often represent and summarize biological measurements using more compact mathematical abstractions that are easier to analyze. One such abstraction is the biological network, or graph, that have biological entities — usually genes — as nodes and functional relationships and correlations as edges. The protein-protein interaction (PPI) network is one illustrative example of interest. A protein-protein interaction network consists of nodes that are genes which are connected if their respective protein products are known to physically interact *in vivo* or *in vitro*. Compared to raw sequencing data these networks are compact. Measurement of biological networks in model organisms form genome-scale "maps" that that reveal how genotypic organization might cause observed phenotypes (e.g. by finding clusters and hubs in a PPI network). Many other compact network representations of biological measurements can also be meaningfully analyzed. For example, to analyze single-cell RNA-seq data, many methods use nearest-neighbor networks or graph based representations to perform dimensionality reduction or infer developmental trajectories [4]. To organize biological knowledge and systematize scientific annotation, the Gene Ontology (GO) hierarchically annotates the relationships between biological processes, cellular components, and molecular function relating to genes on a directed acyclic graph [5].

Biological networks of model biological systems — such as baker's yeast or mouse — not only enable functional understanding of biology within a specific species but also *across* species. For example, if a gene in a well studied *source* species is evolutionarily conserved where there exists an *orthologous* gene in a less well studied *target* species, the biological insight about the gene in the source species can be "transferred" to the target species. Critically, network based comparisons offer insights that may be more functionally complex and insightful than comparing only sequence similarity between genes in different species. Theoretically, entire network (sub)structures that explain complex biological functions can be compared to transfer insights from one model organism to another. Practically, an entire class of *biological network alignment* algorithms have been designed to compare biological networks across species [6, 7, 8, 9, 10, 11, 12].

In the first part of this dissertation, I describe two methods that generalize analyses and move beyond heuristic alignment of nodes across different model species. In Chapter 2, I introduce Munk— a *general purpose* method for embedding networks across multiple species [13]. In Chapter 3, I introduce EMF— a framework of models to better infer missing edges in difficult to measure networks across multiple species [14].

Part II — Sequences.

A caveat to network based representations and analysis is that it presumes, and thus elides, high-quality measurement and analysis of DNA and RNA sequences. For example, constructing reliable and high-quality biological networks require accurate sequencing and genotyping for quality control of *in vitro* samples. For RNA-seq analysis, nearest-neighbor networks rely on accurate alignment of sequenced reads. For methods like HotNet2 that project observed mutation data onto PPI networks to find cancer related genes, accurate sequencing and identification of genetic variation is essential [15]. Designing compact and effective algorithms and tools for analyzing sequencing data is paramount.

To improve sequence based analyses *in practice*, this dissertation addresses two particular issues. First, analyses of RNA-seq measurements of tens-of-millions of reads deriving from over one hundred thousand candidate isoforms (transcripts) result in probabilistic models that can be difficult to work with in practice. In Chapter 5, I address one pain point in the application and gene and transcript abundance estimation tools, also known as *quantifiers*. Much work has been made to make quantifiers fast and theoretically accurate. The validity of popular quantifiers have been supported by comparisons against ground truth in simulated data and quality of analyses that they enable downstream. However, model selection and direct evaluation of quantifiers on experimental data where ground-truth is absent remains difficult. To tackle this issue, I introduce *perplexity* for RNA-seq abundance estimation and address critical considerations unique to RNA-seq [16]. To our knowledge, our derived perplexity metric is the first to enable model selection and evaluation of transcript abundance estimation models in experimental data where ground-truth is entirely absent.

Second, analysis of high-throughput sequencing data at whole-genome-scale requires outsized and often unavailable logistical resources. Though relatively commonplace in academic research, whole-genome sequencing (WGS) analysis remains rare in clinical settings. In the clinic, WGS data require too much space to store and are slow to analyze. In Chapter 4, I introduce ScalpelSig for mutational signature analysis to close the gap between research and clinic [17]. Using existing WGS data, ScalpelSig computationally designs genomic panels that best identify the activity of mutational signatures indicative of diagnostically and scientifically pertinent mutational processes. ScalpelSig panels require the sequencing of far less genetic material and offer one way to close the gap between the outsized requirements of state-of-the-art research and the practical limitations of the clinic.

Part III — Indexing.

In Part I of this dissertation, matrix factorization algorithms demonstrate the power simultaneous analysis of data across multiple biological contexts — namely across multiple species. In Part II of this dissertation, *perplexity* addresses the pain-points of model-selection in RNA-seq and ScalpelSig sidesteps outsized requirements of sequence based analysis. However, no aforementioned method *directly* makes more efficient or reduces the computational requirements of analyzing sequencing data, let alone sequencing data across multiple contexts.

The analysis with respect to large reference collections has been demonstrated to be particularly insightful in many areas. However, the size and set of reference collections must be carefully chosen to maximize scientific discovery while avoiding untenable space requirements. In metagenomic analyses, careful selection and restriction of reference sequences is necessary for practical analysis [18]. In RNA-seq analysis, mapping against "decoy sequences" in addition to known, reference isoforms is an effective heuristic to correct for reads that originate from unknown and novel isoforms [19]. However, only few RNA-seq abundance estimation tools like `salmon` can reasonably do so since the addition of decoy sequences requires the indexing of many more nucleotides [20].

The last part of this dissertation introduces recently published work that improve the indexing and analysis of huge reference sequence collections that require hundreds of gigabytes to represent even when efficiently indexed. I introduce the *spectrum preserving tiling* (SPT), a new computational and mathematical abstraction. Mathematically, the SPT explicitly relates past work on compactly representing k-mer sets — namely the compacted de Bruijn graph and recent derivations of spectrum preserving string sets [21, 22] — to the indexing of k-mer positions and metadata in reference sequences. Computationally, the SPT makes possible an entire class of efficient and modular k-mer indexes. These indexes work modularly by first mapping k-mers-to-tiles, then tiles-to-metadata (e.g., positions on reference sequences).

In Chapter 6, I apply the SPT to the indexing of k-mer positions. I identify that there exist many efficient, compressed, hashing-based schemes that map k-mers-to-tiles but few compression schemes for that map tiles-to-positions. So, by exploiting properties of the SPT, I introduce a novel sampling scheme to trade-off speed for space in a modular index's tile-to-position mapping. Our new sampling scheme allows an index to store a constant number of bits per position rather than a logarithmic number of bits for the positions of unsampled tiles.

In Chapter 7, extending this work, I introduce a new state-of-the-art index to support the k-mer *color* query. A useful approximation to the locate query, the k-mer color query returns the set of reference sequences in which a queried k-mer occurs but not the location where said k-mer occurs. Exploiting the modular nature of the SPT, our new index, `fulgor`, combines the existing state-of-the-art k-mer dictionary `SSHash` [23] to map k-mers-to-tiles (or more specifically, unitigs) with a simple but highly effective hybrid compression scheme to map unitigs-to-colors. I demonstrate in our experiments that `fulgor` is at least 2x smaller than the prior state-of-the-art and is at least twice as fast.

In Chapter 8 I demonstrate how a modular indexing library can be designed and implemented in Rust. Along the way, I introduce a small optimization to Pibiri's `SSHash` with the help of a new definition of minimizers of canonical k-mers. To conclude, I show how our implemented library realizes the modular potential promised by the spectrum preserving tiling, and the associated ideas in Chapters 6 and 7. With a small and succinct example, I show how our theoretical modular indexing ideas can be implemented by a concrete and practical library that make it easy to implement fast and small algorithms and data-structures past, present and future.

Part I

Networks

Chapter 2: MUNK: Joint representations of proteins across multiple species

Disclosure

This Chapter presents an abridged version of first-author work published by [13].

2.1 Background and Motivation

A primary challenge of research with model organisms is to transfer knowledge of genetics — i.e. a mapping of genotype to phenotype — between model organisms and humans. The main promise of researching model organisms stems from researchers' ability to measure the organisms in ways that are infeasible in humans. To realize the promise of this research, it is crucial to transfer knowledge between species — ideally, in two directions. First, discoveries in model organisms can be transferred to improve knowledge of human genetics (e.g. via homology). Second, knowledge of human genetics can be transferred to design better experiments in model organisms (e.g. for disease models).

More specifically, cross-species knowledge transfer can enable a wide variety of applications. First and foremost is the large-scale annotation of protein function by transferring function annotations (e.g. from the Gene Ontology [5]). Addressing this problem remains valuable, even in the era of high-throughput genomics, as fewer than 1% of protein sequences in Uniprot have experimentally-derived functional annotations [24]. Another application of cross-species knowledge transfer is for *pairwise* gene function (genetic interactions). Knowledge of synthetic lethal genetic interactions is crucial for the study of functional genomics and disease [25, 26]. Since genome-wide measurement of synthetic lethal interactions in humans is currently infeasible, computationally transferring knowledge of these interactions from model organisms (such as yeast or mouse) to humans (and human cancers) has become a focus of recent research. A third but less well-explored application is in predicting human disease models through "orthologous phenotypes" or phenologs [27]. McGary et al. [27] reasoned that while conserved genes may retain their *molecular* functions across species, conserved molecular function may manifest as different "species-level" phenotypes. As such, they introduced a statistical test to identify such phenologs.

Cross-species knowledge transfer is quite challenging because many model organisms diverged from humans millions of years ago and have fundamentally different genetic architectures. In many cases, only a relatively small subset of genes between species have sequence homologs. Further, as species diverge, protein functions change and are re-purposed (e.g. [28]) through divergent and convergent evolution, and genetic interactions are often rewired [29, 30].

Existing computational approaches to transfer knowledge across species rely on matching a subset of genes (proteins) in different species by heredity (genetic orthology) or function (functional orthology). One class of computational approach uses sequence data to match genes (proteins) [31, 32]. A second class of methods expands beyond sequence by using proteomics data to match proteins, through protein structure prediction (e.g. [33]) or alignment of protein-protein interaction networks [6, 7, 8, 9, 10, 11], commonly called the network alignment problem.

Many cross-species biological problems cannot be formulated as a matching problem; for example, genetic interactions and phenologs are fundamentally measures of *sets* of genes. This motivates the idea of creating general-purpose multi-species protein representations. These in turn could be used to generate a matching, but could also be interpreted as a vector space or used as input to a learning algorithm. General-purpose representations are fast becoming adopted in different areas of machine learning, from natural language processing (e.g. [34]) to network science (e.g. [35]), and recently have begun to be adopted for biological networks [36, 37].

However, the problem of learning multi-species protein representations from network and sequence data remains largely unexplored. Jacunski et al. [38] showed that protein representations derived from graph theoretic measures of network structure can be used to transfer knowledge of synthetic lethal interactions across species. However, their approach creates the representations in each network independently, does not use sequence data at all, and uses a set of handcrafted features chosen for a particular task. Gligorijević et al. [39] use matrix factorization based on sequence similarity with PPI-based Laplacian smoothing to cluster cross-species protein pairs. However that method does not embed nodes in a common vector space, instead computing scores for a subset of protein pairs that are used as an input to max-weight matching for network alignment. More recently, Khurana et al. [40] developed an embedding for proteins in multiple species for an application concerning neurodegenerative diseases.

2.1.1 Contributions

In this work, we address the limitations of task-specific protein representations, in a way that allows us to move beyond simple matching of proteins across species. We combine protein-protein interaction networks and sequence data from multiple species into unified, biologically meaningful protein representations using network diffusion. Network diffusion is a natural tool for capturing aspects of local and global network structure that correlate with functional similarity of nodes [41].

The key insight of our approach is that homologous proteins can serve as landmarks for relating proteins in different species. We then show the similarity scores derived from these representations as well as the representations themselves are useful for distinct tasks.

Our method makes only two assumptions. First, it assumes that protein function can be captured using a similarity score that is a *kernel*, which encompasses a broad class of useful metrics. Second, it assumes that sequence homology is known for some subset of landmark proteins across the different species.

In this work, we use a *diffusion* kernel to create functional protein representations and call the resulting method Munk (MUlti-Species Network Kernel). Fan et al. [13] evaluate the Munk protein representations on three multi-species tasks. In this proposal, we highlight results in two key areas.

1. **Multi-species functional similarity.** We show that cross-species matchings and similarity scores derived from the Munk representations are significantly correlated with cross-species protein function.

2. **Multi-species synthetic lethality.** We train classifiers on Munk-representations for *pairs* of genes in order to predict synthetic lethal interactions (SLI) in multiple species. We find that classifiers accurately identify SLI in multiple species *simultaneously*, and that they achieve comparable performance to the Sinatra algorithm.

Together, these tasks encompass transferring knowledge both between model organisms, and between model organisms and humans.

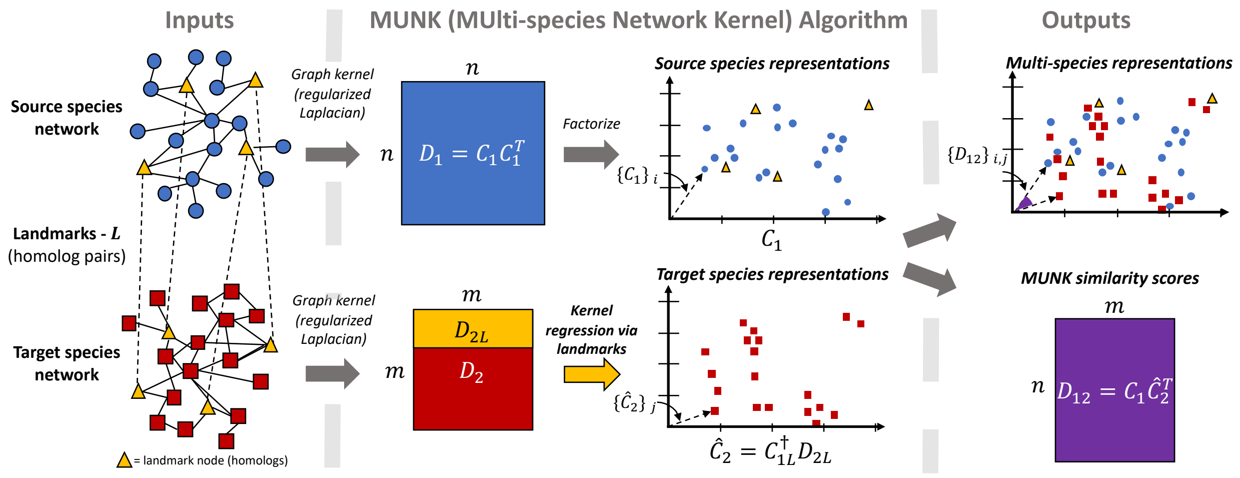

Inputs
MUNK (MUlti-species Network Kernel) Algorithm
Outputs
Source species network
Landmarks - L (homolog pairs)
Target species network
= landmark node (homologs)
Graph kernel (regularized Laplacian)
Graph kernel (regularized Laplacian)
n
n
$D_1 = C_1 C_1^T$
m
m
D_{2L}
D_2
Factorize
Kernel regression via landmarks
Source species representations
$\{C_1\}_i$
C_1
Target species representations
$\{\hat{C}_2\}_j$
$\hat{C}_2 = C_{1L}^{\dagger} D_{2L}$
Multi-species representations
$\{D_{12}\}_{i,j}$
MUNK similarity scores
m
n
$D_{12} = C_1 \hat{C}_2^T$

Figure 2.1: Given a source PPI network, a target PPI network, and a set of landmark (homolog) pairs across species, Munk computes diffusion kernels for each network. Then, Munk factorizes the diffusion kernel for the source species into its reproducing kernel Hilbert space (RKHS). Finally, Munk solves a linear system of the source species' RKHS and the target species' diffusion kernel to create a multi-species vector embedding of source and target proteins. The inner products of these embeddings correlate with functional similarities and the embeddings themselves allow for functional comparisons between proteins across the two networks.

2.2 Methods

In this work, we introduce Munk, a model to that jointly embeds proteins across biological networks from *different* species into the *same* functional space. Munk, illustrated in Fig. 2.1, leverages properties of graph kernels as tools for measuring the similarity of nodes in a network and for creating embeddings. While the use of kernels for the study of individual networks is well known, it remains an open problem to construct network-based kernels that capture the similarity of nodes between *different* networks. This is the challenge that Munk addresses.

2.2.1 Multi-Species Network Kernel (Munk) Embedding

Given a pair of PPI networks from *source* and a *target* species, and a graph kernel (a node-node similarity function) computable for *each* network, Munk jointly embeds target and source species nodes into the same functionally meaningful vector space. In this joint vector space, nodes can be jointly analyzed and compared *across* target and source species for better predictions and new biological insights.

We start by noting that there are a large variety of kernels derived from networks [42; 43, Ch. 2], and that they can model processes such as random walks, heat diffusion, PageRank, electrical resistance, and otherways of capturing node similarity in a network. Many kernels derived from networks have been applied successfully for a wide range of problems associated with biological network analysis. [1]

Though many previous studies have used graph kernels to compare nodes *within* biological networks, to our knowledge few methods have utilized kernels to compare nodes *across* multiple biological networks. To do so, Munk relies on a basic property that any kernel is also an *inner product* in a particular space. That is, for any kernel $\kappa(\cdot, \cdot)$, there is a function $\phi(\cdot)$ that assigns vectors to nodes i and j such that that $\kappa(i, j) = \phi(i)^T \phi(j)$. The corresponding vector space (termed the *reproducing kernel Hilbert space* (RKHS)) introduces a geometric interpretation for the kernel function. In the context of a kernel for network nodes, the RKHS representation can be thought of as an *embedding* of the network into a vector space in a manner that captures node similarity via inner product.

Given a *source* network G_1, a *target* network G_2, and a kernel κ, Munk first embeds the nodes

[1] For example, Cowen et al. [41] review numerous areas where said techniques are applied to discover new biological insights.

of the source network G_1 into the RKHS defined by κ using the associated embedding function, ϕ. Next, Munk makes use of *landmarks* — pairs of nodes in the source and target networks known to have identical function to embed the nodes of the target network G_2 into the same space as the nodes of the source network G_1. Munk maps nodes embeds nodes in the target network into the source network such that landmarks in the target network have the same embeddings as landmarks in the source network. Essentially, we posit that locating *any* node from the target network, G_2, based on its similarity to the set of landmarks in G_2 with a 1-1 mapping to G_1 will also establish its similarity with the *non-landmark* nodes in the source network, G_1. As a result, Munk creates a multi-network kernel — a single kernel function that captures both the similarity of nodes to each other in the source network G_1, and the similarity of nodes between the source and target networks G_1 and G_2.

Let the matrix $D \in \mathbb{R}^{n \times n}$ hold the values of the similarity function $\kappa(i, j)$ for all pairs of n proteins from a particular species. For any such kernel matrix, we can write $D = CC^T$ where C is an $n \times k$ matrix, uniquely defined up to an orthogonal transformation, with $k \leq n$. This follows from the fact that K is positive semidefinite, and means that $\kappa(i, j) = c_i^T c_j$, where c_i is the ith row of C, represented as a column vector. As explained above, the similarity between nodes v_i and v_j is exactly given by the inner product of their corresponding vectors, c_i and c_j.

Now consider a source network $G_1 = (V_1, E_1)$ and a target network $G_2 = (V_2, E_2)$ with $|V_1| = m$ and $|V_2| = n$. Munk assumes and requires the existence of some (small set of) nodes that correspond one-to-one between G_1 and G_2. In the case where G_1 and G_2 are PPI networks, these can be orthologous proteins. For example, for orthologous proteins in different networks, it is well known that evolutionary rates differ over a wide range of magnitudes [44]. Some proteins are highly conserved, and their orthologs will have substantial sequence similarity between G_1 and

G_2. Thus, there is generally a small subset of proteins that can be confidently mapped between G_1 and G_2. We coin these nodes, *landmarks*.

Munk then proceeds as follows. First, Munk constructs kernel (similarity) matrices $D_1 \in \mathbb{R}^{m \times m}$ and $D_2 \in \mathbb{R}^{n \times n}$ corresponding to G_1 and G_2. Next, Munk construct RKHS vector representations C_1 for nodes in the source network G_1 from the factorization $D_1 = C_1 C_1^T$. Let C_{1L} be the subset of the rows of C_1 corresponding to landmarks, and let D_{2L} be the subset of the rows of D_2 corresponding to landmarks (in corresponding order).

The key step then is to construct the vector representations of the nodes in the target network G_2. To do this, Munk treats the similarity scores D_{2L} in the target network as if as if they applied to the embeddings of landmarks in the source network G_1. For a given node in the target network, Munk finds an embedding such that its inner product between the *target* node and a *source* landmark is equal to the inner-product between the *target* node and the corresponding *target* landmark. This implies that the RKHS vectors, $\hat{C}_2$, for nodes in the target network G_2 should satisfy $D_{2L} = C_{1L} \hat{C}_2^T$. This underdetermined linear system has solution set,

$$\hat{C}_2^T = C_{1L}^{\dagger} D_{2L} + (I - C_{1L}^{\dagger} C_{1L}) W, \tag{2.1}$$

where $C_{1L}^{\dagger}$ is the Moore-Penrose pseudoinverse of C_{1L}, and W is an arbitrary matrix. We choose the solution corresponding to $W = 0$, meaning that the vectors $\hat{C}_2^T$ are the solutions having minimum norm.

The resulting solution, $\hat{C}_2$, represents the embedding of the nodes of G_2 (the target) into the same space as the nodes of G_1 (the source). We can then compute similarity scores for all pairs of nodes across the two networks as $D_{12} = C_1 \hat{C}_2^T$. This yields D_{12}, an $m \times n$ matrix of similarity scores between nodes in the source and target networks.

2.2.2 Munk and the Regularized Laplacian

While Munk can be used with any graph or network kernel, Fan et al. [13] perform their study using a kernel that has been shown to capture and encode functional similarities between proteins in a PPI network. Specifically, the regularized Laplacian kernel used. The regularized Laplacian kernel is a natural choice for this task because of its close relationship to the principle of "guilt-by-association" often used by protein function prediction methods [45], and to network diffusion methods [46]. Further theoretical justification for this choice is discussed in the published manuscript [13]. Concretely, the regularized laplacian kernel on a network with adjacency matrix A is $(I + \lambda L)^{-1}$, with $L = D - A$ where D is the degree matrix — a diagonal matrix with node degrees $D_{ii} = \sum_j a_{ij}$, on the diagonal.

The combination of the multi-network kernel embedding described in the previous section with the Regularized Laplacian constitutes Munk, and the resulting cross-species similarity scores are Munk *scores*. We denote the Munk score of two proteins p_i and p_j as d_{ij}, we refer to the RKHS in which G_1 and G_2 are embedded as Munk-space, and the Munk-representations are given by the rows of C_1 and $\hat{C}_2$.

2.2.3 Representations for protein (gene) pairs

We also find it useful to develop representations for pairs of nodes (proteins or genes). These can be used to capture functional similarity between two *pairs* of nodes across species. Further, pair-representations can then be used to predict outcomes for pairs of genes. Given two pairs of nodes

$(v_i, v_j), (v_k, v_\ell)$, we define a pairwise similarity metric such that the score is large only if d_{ik} and $d_{j\ell}$ (or $d_{i\ell}$ and d_{jk}) are both large. This reflects the hypothesis that synthetic lethal interactions occur within pathways, and between pathways that perform the same/similar essential biological function [47, 48].

Hence, to represent a pair, we simply sum the Munk-representations for the nodes in the pair. We then compare pairs by computing Munk scores in the usual way. Given a matrix C of Munk-representations for nodes, we define the Munk-representations for a *pair* of nodes (v_i, v_j) as $P_C(v_i, v_j) = c_i + c_j$. Computing similarity for a two pairs (v_i, v_j) and (v_k, v_ℓ) then yields:

$$(c_i + c_j)^T(c_k + c_\ell) = c_i^T c_k + c_i^T c_\ell + c_j^T c_k + c_j^T c_\ell.$$

In general we expect each of the terms on the right hand side to be close to zero *unless* there is functional similarity between the corresponding nodes, because in high dimension, independent random vectors tend to be nearly orthogonal. We note that the pair-similarity scores and the pair-representations themselves can be used for a variety of tasks.

2.2.4 Data and Experimental protocols

For the experiments below, we study the human (*Homo sapiens*), mouse (*Mus musculus*), baker's yeast (*Saccharomyces cerevisiae*) and fission yeast (*S. pombe*) PPI networks. We downloaded and processed PPI networks for human and mouse from the STRING database [49], PPI networks for *S. cerevisiae* and *S. pombe* from BioGRID databases [50].[2] We downloaded and obtained mappings of homologous genes-pairs across model organisms from NCBI's Homologene database [32]. We

[2]Please refer to Supplemental Information published by Fan et al. [13] for details.

constructed datasets of synthetic lethal interactions (SLI) and non-interactions (non-SLI) from two high-throughput studies of analogous proteins in baker's (*S. cerevisiae*) and fission (*S. pombe*) yeast [29, 51]. For all experiments below 400 random homolog pairs are selected to be landmarks for Munk.

2.3 Results

2.3.1 Munk-representations capture functional similarity across species

Our results show that the similarity scores given by Munk-representations are strongly correlated with functional similarity between human and mouse proteins. Here, we compare pairs of proteins (p_i, p_j), where p_i and p_j are from human (source) and mouse (target), respectively. We only include pairs for which neither p_i or p_j are part of a landmark pair.

We use the Resnik score [52] as a quantitative measure of functional similarity. The Resnik score between two Gene Ontology (GO) [5] terms is the information content of their most informative common ancestor in the GO hierarchy; to compare two proteins we take the maximum Resnik score over all pairs of GO terms that label each protein. The Resnik score has been shown to be one of the best performing metrics for capturing functional similarity within the GO hierarchy [53].

To demonstrate the relation between Munk similarity scores and functional similarity, we order each pair according to their Munk scores, and plot rankings against the Resnik score of the pair. The results (smoothed over non-overlapping windows of 100,000 observations) are shown in Fig. 2.2(a).

Figure 2.2(a) shows that Munk scores are strongly correlated with functional similarity across the entire range of scores. Furthermore, the very largest Munk scores are indicative of protein pairs with particularly high functional similarity.

Next, we show that pairs that are known to be functionally related are distinguishable by their Munk similarity scores. For this purpose, we separate pairs (p_i, p_j) where p_i and p_j are homologous proteins in different organisms from other pairs.

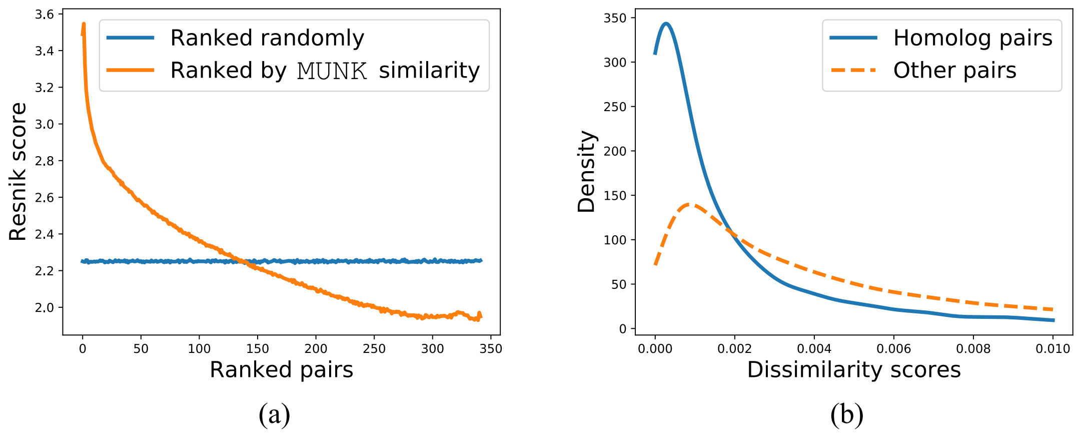


Figure 2.2: The relationship between Munk similarity scores and functional similarity for the human (source) to mouse (target) embeddings. (a) Relationship between functional similarity measured by Resnik score (y-axis) and protein pairs ranked (x-axis) by Munk similarity (shown in orange) and ranked randomly (shown in blue; included as a baseline) — smoothed over non-overlapping windows of size 100,000 on the x-axis. (b) Distribution of Munk dissimilarity scores for homologous protein pairs compared to other (non-homologous) protein pairs.

In Fig. 2.2(b), we show the distribution of Munk dissimilarity among known homolog pairs, as compared to the distribution of scores across other pairs. In this figure, we use reciprocal scores (dissimilarities), meaning that small scores are associated with high functional similarity. Only the left side of the distributions are shown, as the distribution of all pairs extends far to the right and obscures the homolog distribution on the left. The mean Munk dissimilarity scores for human-mouse homologs are 36% lower than the mean across other protein pairs.

In the published study, Fan et al. [13] also show that Munk similarity scores outperform or perform comparably to network alignment methods [7, 11] that aim to find *functionally* similar genes across biological networks that cannot be identified by sequence alone. Critically, Munk goes beyond simply computing similarities and embeds nodes in target and source species networks into a joint vector space that can be analyzed downstream.

2.3.2 Multi-species synthetic lethal interaction prediction with Munk

In this section we demonstrate the advantages of general-purpose cross-species protein representations by using Munk-representations to predict synthetic lethal interactions (SLI) in multiple species simultaneously. Existing matching-based network alignment methods are unable to generalize to this problem, since SLI are a property of *pairs* of genes. Similarly, most existing methods for creating network-based representations are also ill-suited for this problem, since genes in different species are in different vector spaces with different dimensions. The exception to this is the Sinatra algorithm [38], which we benchmark against.

For multi-species synthetic lethal classification, we compare to Sinatra. Sinatra computes "connectivity profiles" from a given network by computing graph theoretic measures of topology for each node, and then trains classifiers to predict synthetic lethal interactions across species from rank-normalized the connectivity profiles. We implemented Sinatra in Python 3, as the authors did not make any software publicly available.

We show that classifiers trained on Munk-representations can accurately predict SLI in two different species of yeast — *S. cerevisiae* and *S. pombe* — simultaneously, providing evidence that gene pairs with SLI in different species are co-located in Munk-space. More specifically, we train a random forest (RF) to classify gene pairs as SLI or non-SLI within both species *simultaneously*, using the source embedding (given by P_{C_1}) and the target embedded into source space (given by $P_{\hat{C}_2}$) (see Section 2.2.3). We perform 4-fold cross-validation, fixing the relative fraction of pairs from each species, and assess the degree of separation between SLI and non-SLI in Munk-space by evaluating the RF classifications with maximum F_1 score (the harmonic mean of precision and recall), the area under the ROC curve (AUROC), and the area under the precision-recall curve

Test	**Features**	**AUROC**	**AUPR**	**Max F_1**
S. cerevisiae	Munk	**0.933**	**0.933**	**0.860**
	Sinatra	0.908	0.907	0.834
S. pombe	Munk	0.876	0.877	**0.814**
	Sinatra	**0.880**	**0.892**	0.808

Table 2.1: Results training classifiers for synthetic lethal interactions on baker's yeast (*S. cerevisiae*) and fission yeast (*S. pombe*) data from BioGRID[**?**] *simultaneously*. We compute performance separately for each species (indicated by "Test species"). For each statistic, we report the average on held-out data from 4-fold cross-validation, and bold the highest (best) score.

(AUPR). We report the average across the four folds, separating the results by species. We use a nested cross-validation strategy to choose the number of trees for the RF that maximizes the held-out AUPR. For simplicity, all of our experiments in this section use *S. cerevisiae* as the source and *S. pombe* as the target.

We train random forest classifiers with matched high-throughput datasets from *S. cerevisiae* and *S. pombe* These datasets consist of SLI and non-SLI pairs among 743 *S. cerevisiae* genes [51] and 550 *S. pombe* genes [29] involved in chromosome biology (see *Section 2.2.4* for additional details of the dataset). The key differences between the chromosome biology SLI datasets and the BioGRID datasets are that the chromosome biology datasets are restricted to functionally similar genes, include 5.5% SLI and 94.5% measured non-SLI in *S. cerevisiae* and 10.6% SLI and 89.4% measured non-SLI in *S. pombe* (unlike the BioGRID data which only measured SLs), and were generated through high-throughput experiments.

Table 2.1 shows that the RFs trained on Munk-representations achieve significant predictive performance on held-out data from the chromosome biology dataset, with an AUROC of 0.864 in *S. cerevisiae* (0.822 in *S. pombe*), AUPR of 0.402 (0.370), and maximum F_1 score of 0.421 (0.423). Notably, Munk outperforms Sinatra by a large margin for predictions in *S. cerevisiae*, while Sinatra outperforms Munk by a smaller margin for predictions in *S. pombe*.

Together, these results show that synthetic lethal interactions are significantly clustered *across species* in Munk-space, and that by using Munk-representations, which leverage knowledge of a subset of homologous genes across species, classifiers can make accurate predictions for other homologous and non-homologous gene pairs.

2.4 Discussion

We introduce a novel, kernel-based algorithm to create general-purpose, multi-species protein representations using biological networks and sequence data. We use a particular diffusion kernel – the regularized Laplacian – to create functional representations, and use the resulting algorithm, Munk, to embed proteins from humans, mice, and yeast into shared spaces. We evaluate the Munk-representations on cross-species functional similarity and multi-species synthetic lethal prediction, showing the Munk-representations lead to comparable performance as specialized methods for these tasks. We also use Munk to expand the notion of orthologous phenotypes beyond evolutionarily conserved sequence and identify known and novel phenologs, providing evidence for non-obvious human disease models. Importantly, in these tasks, we transfer knowledge both from humans to model organisms and from model organisms to humans. Thus, Munk represents a new direction towards realizing the crucial goal of algorithms for transferring knowledge of genetics across species.

Our approach of creating cross-species protein representations can be seen as a component of a *transfer learning* [54] approach for cross-species inference. The promise of transfer learning – using knowledge gained in solving one task to aid in solving a different task – for cross-species inference is to leverage species where data is widely available for predictions in species where data is sparse.

For example, this is the case for genetic interactions, where approximately 90% of pairwise genetic interactions have been measured in baker's yeast [55], while fewer than 1% of pairs have been tested in humans. Transfer learning is often approached by finding appropriate transformations of data features (e.g., "domain adaptation", e.g., see [56]). For Munk, methods for aligning the source and target embeddings may be required to make such a transfer learning approach possible. At the same time, we showed that methods for transferring knowledge across species can be useful even when there is a wealth of data in the target species. Thus, to achieve optimal performance, supervised learners may need to train on multiple species simultaneously.

Beyond kernels derived from protein interaction, there are a wide range of other kernels that can inform biological function assessment, including kernels derived from co-expression, genetic interaction, metabolic pathways, domain structure, and sequence [57, 58, 59]. Because Munk is a method for creating a new kernel encompassing the nodes of multiple networks, it holds potential as a new tool for kernel learning methods such as support vector machines in a wide variety of applications beyond cross-species function prediction.

Chapter 3: Matrix (Factorization) Reloaded: Flexible methods for imputing genetic interactions

Disclosure

This Chapter presents first-author work as published in [14], with minimal changes.

3.1 Background and Motivation

A genetic interaction (GI) is a measure of how a *combination* of gene variants produces a phenotype that is different than expected, given the phenotypes of each *independent* gene variant. Most commonly, a GI is measured for a pair of gene knockouts with a measure of cell viability as the phenotype. Although a single GI provides only limited phenotypic information, mapping a set of GIs in a model organism is thought to be able to resolve fundamental biological questions such as the minimum number of genes required for a viable cell [60, 61]. Furthermore, knowledge of GIs has enabled promising new strategies for cancer treatment [62, 63], and may expand opportunities for treating infectious diseases [64].

Consequently, identifying and characterizing GIs has been a major focus in systems biology for the past two decades, spurring innovations in experimental systems and computational methods. Recently, researchers have sought to go beyond measuring interactions for small sets of specific genes or gene pairs, to develop approaches for generating what are referred to as "unbiased" maps of pairwise *quantitative* GIs between large sets of genes [65]. A quantitative GI for a tested pair of genes is a real-valued score for the direction (positive / alleviating versus negative / aggravating) and strength of the interaction. For example, treated quantitatively, a synthetic lethal interaction is a GI with a score much less than zero. *In vitro* efforts began in baker's yeast with small maps for all pairs of genes involved in key biological functions [29, 51, 66]. These efforts culminated in a landmark study [55] that published a map of GIs for over 90% of all genes in baker's yeast (*S. cer evisiae*).

Despite impressive progress, many challenges remain. These challenges include measuring interactions in species other than baker's yeast, examining higher-order interactions for sets of more than two genes, and measuring GIs for different phenotypes. In fact, in each of these cases, there have been recent experimental studies ([60, 67, 68]). However, the landscape of yet unmeasured GIs remains vast and will not be fully explored through *in vitro* experimentation alone. Thus, there is an enormous need for *in silico* methods to complement the recent and ongoing experimental advances.

To address this need, methods have been developed along a number of dimensions. First, it is important to note the critical difference between the classification problem posed by *binary* classes of extreme GIs, and the regression problem associated with the larger information content contained in *quantitative* GI data. With respect to binary GIs, much work has been focused on the prediction of synthetic lethal interactions [38, 69, 70, 71, 72, 73], sometimes treating the classification problem as standard link prediction [74, 75]. However, genome-scale work in yeast has gone beyond identifying the most extreme interactions to identifying correlations between genes' profiles of quantitative GI scores regardless of magnitude, in order to create genome-wide maps of gene function [76]. Hence, in this paper, we study imputing real valued, quantitative GI scores for all gene-pairs.

Ulitsky et al. [77] were the first to develop methods for the *regression* problem of predicting quantitative GIs using features derived from functional annotations, protein-protein interactions, and the Gene Ontology (GO) [78]. More recently, Ma et al. [79] — building off of the work of Yu et al. [80] — introduced an interpretable deep learning method that uses GO to achieve state-of-the-art performance in predicting GI scores in baker's yeast.

In this work, we present a new computational framework for the quantitative GI regression problem, termed *Extensible Matrix Factorization* (EMF), and show its utility in both baker's yeast *and* fission yeast (*S. pombe*). In developing EMF, we seek to overcome a number of limitations of existing methods.

First, we note that existing state-of-the-art methods are predicated on the availability of *specific* kinds of side information as a necessary input for feature generation. That is, computational methods such as DCell [79] require annotations from GO. However, the availability and quality of GO annotations vary widely across species. For example, baker's yeast has more than double the number of annotations in fission yeast [78]. Thus, the reliance on specific *side information* as input to DCell limits its ability to be used for a wide range of species. Furthermore, no methods have exploited known correlations across GI data [29, 67, 81] in related species (*cross-species information*) to make predictions in species in which data is scarce.

Second, training state-of-the-art methods is computationally intensive. For example, DCell took two to three days to train on data from Costanzo et al. [55]. Methods that require significant time to train can impede efforts to develop and benchmark new models; this bottleneck may grow further as the sizes of GI datasets increase.

To address these limitations, EMF is designed to be a more broadly useful approach that can *flexibly incorporate* various kinds of side information, as available. The EMF framework consists of a collection of composable matrix factorization (MF) models that can optionally exploit known non-uniformities in GIs, within-species side information (via kernelization), and cross-species information (via gene-gene similarities). A core contribution within EMF is *cross-species matrix factorization* (XSMF), a new method for using information from one (*source*) species to improve GI imputation in a second (*target*) species.

We also designed EMF to have low computational cost — EMF models typically takes less than one minute to train on genome-scale data. As evidence of the scalability *and* flexibility of EMF, we use it to impute GIs in baker's *and* fission yeast at genome-scale. To the best of our knowledge, ours is the first study to do so in fission yeast.

Further, we note that recent evidence from data mining literature shows that MF can be competitive with deep learning for some problems when attention is paid to details such as hyperparameter tuning [82]. In light of this, we also present in this study a principled approach to composing EMF models and a rigorous approach to hyperparameter optimization to properly weight model combinations.

In the remainder of this work, we show that (when properly applied) MF compares favorably to previous methods. Our contributions include:

1. **Extensible Matrix Factorization (EMF): a framework of composable matrix factorization models for imputing genetic interactions**. EMF extends and unifies several existing MF methods that have not previously been applied to GIs. EMF consists of: a cross-species model that regularizes learned factors across species based on a gene-gene similarity measure; a kernelized model that regularizes learned factors within species; and, a bias model that learns the mean GI score per gene and (motivated by Koch et al. [81]) regularizes biases

across species.

2. **Rigorous benchmarking of EMF on matched datasets from baker's and fission yeast**. We compare EMF models on matched GI datasets for chromosome biology genes from baker's and fission yeast using automated approaches for hyperparameter selection [83], and show that each component of the EMF framework captures additional and complementary signal in data-scarce settings. We also compare directly to the one earlier MF method for imputing GIs, and find EMF to be superior in performance.

3. **Application of EMF to genome-scale datasets**. We apply the best performing models from our benchmarking experiment on datasets covering 75% and 60% of all non-essential genes in baker's and fission yeast, respectively [67, 76]. Compared to the state-of-the-art as reported in literature, EMF models show superior performance, and train in minutes instead of days.

3.2 Methods

Matrix factorization (MF) [84, 85], also referred to as *matrix completion* [86], is a strategy for imputing missing values in a matrix. The matrix is generally assumed to contain redundancies and potentially other regularities or correlations. In other words, a subset of visible values suffices to approximately infer some or all missing values.

MF has proven to be a broadly effective technique in a wide range of problem areas [85, 87, 88]. Furthermore, it can have a number of advantages over more recently developed methods such as deep learning [82]. A goal of this study is to demonstrate how MF can be an advantageous strategy for the problem of genetic interaction (GI) prediction.

MF takes as input a $n \times m$ matrix X that is *partially observed*. We use Ω to denote the set of indices in X whose values are known. The goal of MF is to impute missing values in X (i.e., $(i, j) \notin \Omega$). The basic MF framework starts from the assumption that X, were it fully-known, would be *effectively low-rank*. That is, X can be well approximated by a matrix R of rank $k \ll \min(m, n)$. MF methods seek to estimate R and use the values of R to estimate the missing values of X.

This suggests the following optimization problem:

$$U, V = \underset{U^*, V^*}{\text{argmin}} \sum_{(i,j)\in\Omega} (x_{ij} - u_i^{T*} v_j^*)^2 \tag{3.1}$$

in which U and V are matrices with k rows, where k is a hyperparameter chosen to model the effective rank of X. This framework allows one to recover $R = U^T V$, and admits an interpretation of corresponding columns u_i and v_j as *latent factors* representing the entities on the i-th row and j-th column of X respectively. Then, each missing value $(i, j) \notin \Omega$ can be imputed by computing the inner product $u_i^T v_j$ of the learned latent factors.

Regularization to reduce overfitting can be achieved by including additional terms, such as the ℓ_2-regularizer from [85] and [84]:

$$U, V = \underset{U^*, V^*}{\text{argmin}} \sum_{(i,j)\in\Omega} (x_{ij} - u_i^{T*} v_j^*)^2 + \lambda \left(\|U^*\|_F^2 + \|V^*\|_F^2\right). \tag{3.2}$$

The basic MF framework succeeds by exploiting the inherent low effective rank of the data. Moreover, an important advantage of MF is the straightforward and principled ways in which it can be adapted to incorporate additional regularities in the data. For example, "side" information (additional data features) may be predictive in a manner that is synergistic with the basic low-rank assumption.

3.2.1 Extensible Matrix Factorization (EMF): a composable *class* of matrix factorization models

We present a set of composable components for MF that exploit *cross-species* and *side* information. We derive these from biological observations and ultimately incorporate these into a unified *Extensible Matrix Factorization* (EMF) framework.

EMF encompasses several existing MF models, including the basic MF model given in (3.2) [84], MF with bias (MF-b) [85], and kernelized probabilistic MF (KPMF) [89]. Our contribution with EMF is in presenting a unified view of these models, and expanding their formulations to cross-species settings.

We describe the framework generally as it applies to matrices of biological data where the rows and columns are indexed by genes. We begin by introducing two novel components for exploiting cross-species information, and then present a component for exploiting side information (i.e. within a species). While we apply the EMF components in both the single-species and cross-species settings, we describe all components as they apply to a cross-species setting. Where emphasis is useful, we describe how the models can be leveraged specifically for imputing genetic interactions (GIs).

3.2.1.1 Cross-species matrix factorization (XSMF)

The first extension to MF that we propose is a *cross-species matrix factorization (XSMF) component*, a novel MF scheme that jointly factorizes matrices in a *target* and a *source* species to better impute missing values in the *target*.

Let $X \in \mathbb{R}^{n \times m}$ be a partially observed matrix for a target species, and $Y \in \mathbb{R}^{n' \times m'}$ be a partially observed matrix for a source species. We use Ω_X and Ω_Y to denote the indices in X and Y whose values are known. We present, piecewise, the optimization objective that defines XSMF.

First, the primary objective of XSMF is to estimate latent factors $U \in \mathbb{R}^{k \times n}$ and $V \in \mathbb{R}^{k \times m}$ that best reconstruct observed values in the target species.

To do so, XSMF minimizes the objective:

$$\mathcal{L}_t = \sum_{(i,j) \in \Omega_X} (x_{ij} - \hat{x}_{ij})^2, \tag{3.3}$$

where $\hat{x}_{ij} = u_i^T v_j$.

Second, XSMF simultaneously estimates latent factors $F \in \mathbb{R}^{k \times n'}$ and $H \in \mathbb{R}^{k \times m'}$ that reconstruct observed values in the source species. To do so, XSMF also minimizes the objective:

$$\mathcal{L}_s = \sum_{(i,j) \in \Omega_Y} (y_{ij} - \hat{y}_{ij})^2, \tag{3.4}$$

where $\hat{y}_{ij} = f_i^T h_j$.

Third, given a *similarity* measure between target species genes and source species genes, $\text{sim}(\cdot, \cdot)$, with corresponding similarity score matrix, S, XSMF *links* the factorizations sought by Eqs. (3.3) and (3.4).

It is important to observe here that matrices belonging to target and source species cannot be naively merged because there is no complete one-to-one correspondence between genes in the rows and columns of the target and source. It is also not useful to naively merge matrices by adding source species values, via new rows and columns for source genes, to the target matrix. In such a merged matrix, the latent factors between source and target genes would be independent and not interact.

Thus, XSMF seeks to maximize *weighted inner products* between the latent factors of genes by minimizing the objective:

$$\mathcal{L}_{\mathbf{x}} = -\sum_i \sum_j \text{sim}(i,j) \cdot u_i^T f_j \tag{3.5}$$

or equivalently,

$$\mathcal{L}_{\mathbf{x}} = -\text{tr}(USF^T). \tag{3.6}$$

Finally, ℓ_2 regularization is also added to reduce overfitting and XSMF also minimizes the regularizer:

$$\mathcal{L}_r = \|U\|_F^2 + \|V\|_F^2 + \|F\|_F^2 + \|H\|_F^2 \tag{3.7}$$

The full objective function that XSMF minimizes, with respect to latent factors, can then be written as:

$$\mathcal{L}_{\text{XSMF}} = \mathcal{L}_t + \lambda_s \mathcal{L}_s + \lambda_{\mathbf{x}} \mathcal{L}_{\mathbf{x}} + \lambda_r \mathcal{L}_r \tag{3.8}$$

with the introduction of user-defined hyperparameters λ_s, $\lambda_{\mathbf{x}}$, and λ_r. In the XSMF model, the parameters λ_s and $\lambda_{\mathbf{x}}$ have useful interpretations. The hyperparameter λ_s controls the tradeoff between reconstructing the target and source species values, and $\lambda_{\mathbf{x}}$ controls the degree to which latent factors of similar genes across species ought to be close in representation.

We highlight that *sim*$(\cdot,\cdot)$ can be *any* reasonable similarity measure of homology. For example, similarity measures like BLAST bitscores [31], string kernels for protein and DNA sequences [90], or similarity scores based on biological networks [13], that have proven to be informative in other contexts can be utilized with little to no modification. Unlike researcher-provided labels in GO, many such similarity measures can be computed with only minimal researcher supervision.

3.2.1.2 Modeling per-gene biases in average values

It has been observed in other settings that MF models that explicitly account for per-column and per-row "biases" have been shown to outperform MF models that do not [85].

In fact, for GI data, the average GI score (i.e., the propensity for a given gene to genetically interact with any other gene) is known to be non-uniform across yeast genomes [55, 67, 76].

Thus, all models in the EMF framework can be extended to account for per-gene biases. For cross-species models, per-gene latent bias terms can be introduced and an imputed value in the target species between genes (i, j) can instead be modified to:

$$\hat{x}_{ij} = u_i^T v_j + b_i. \tag{3.9}$$

In the source species an imputed value can be modified to:

$$\hat{y}_{ij} = f_i^T h_j + b'_i. \tag{3.10}$$

Naturally, ℓ_2 regularization over corresponding vectors of biases, b and b', can be added to the final optimization objective to reduce overfitting.

3.2.1.3 Modeling the conservation of biases across species

EMF can also “link” biases if one expects biases to be correlated across species. In fact, this is the case for GIs. Koch et al. [81] showed that the total number of extreme, synthetic lethal interactions can be correlated between similar genes in baker’s and fission yeast. This observation motivates an additional way to exploit cross-species similarities in the EMF framework. The following regularization term that links biases in the source and target can be added to cross-species models:

$$\mathcal{L}_b = -b^T S b'. \tag{3.11}$$

Adding the regularization term $\mathcal{L}_b$ to the objective of an EMF model encourages biases of similar genes to also be similar.

3.2.1.4 Incorporating arbitrary side information

Recent work in MF has introduced a number of additional ways to incorporate side information — such as networks [91] or kernels [89] — to further improve model performance.

We adapt the kernelized approach taken by Zhou et al. [89] to extend both single-species and cross-species models in EMF. To exploit side information in the target species, kernels that regularize latent factors U and V are introduced. Kernelization enables incorporation of any arbitrary side information about known similarities between (same-species) genes, as long as appropriate kernels K_U and K_V can be computed for genes in the target species. Concretely, the following quadratic terms can either be added in addition to, or replace the usual ℓ_2 regularizers on U and V:

$$\mathcal{L}_{kt} = \text{tr}\,(UK_U^{-1}U^T) + \text{tr}\,(VK_V^{-1}V^T). \tag{3.12}$$

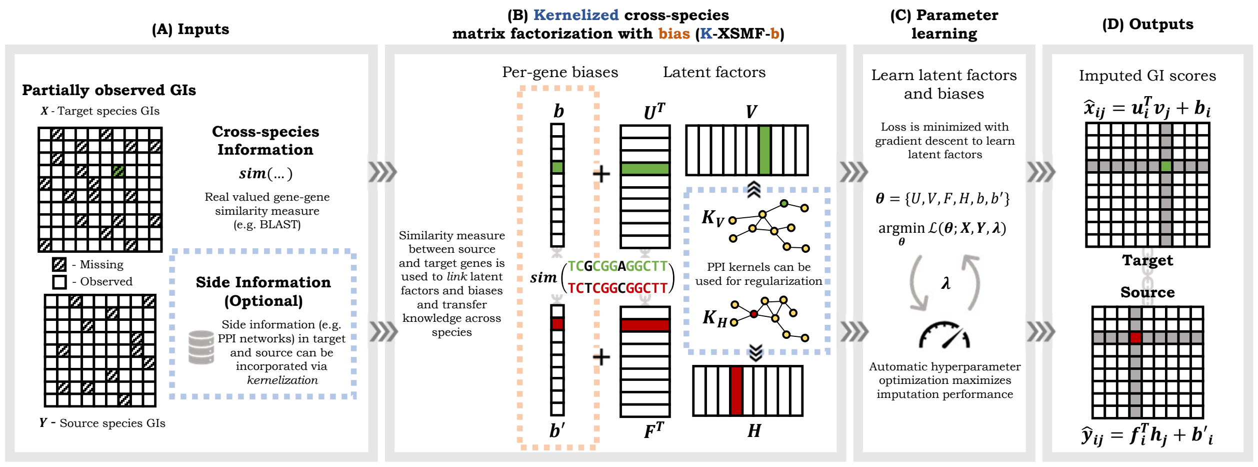
Intuitively, these regularizers encourage corresponding latent factors for two genes to be close if two genes are similar, given a particular kernel.

For cross-species models, assuming the availability of appropriate kernels in the source species, the same technique can be applied to factors F and H, and the following quadratic term can be added for regularization:

$$\mathcal{L}_{ks} = \text{tr}\,(FK_F^{-1}F^T) + \text{tr}\,(HK_H^{-1}H^T). \tag{3.13}$$

3.2.2 A kernelized cross-species model including bias for imputing genetic interactions

In the sections above, we have described, in abstract terms, how loss terms can be composed to form EMF models with varying complexity. As an example, we describe in detail an instantiation of the EMF framework designed specifically to impute GIs. *Kernelized cross-species matrix factorization with bias* (K-XSMF-b) is a cross-species EMF model that imputes missing GIs in a target species. K-XSMF-b takes as input partially observed matrices of GIs of the target and a source species, computed cross-species gene-gene similarities, and side information in both target and source species. We graphically illustrate the components of K-XSMF-b and the greater EMF framework in Fig. 3.1, and describe the optimization objective for K-XSMF-b in parts.



(A) Inputs
Partially observed GIs
X - Target species GIs
- Missing
- Observed
Y - Source species GIs
Cross-species Information
sim(...)
Real valued gene-gene similarity measure (e.g. BLAST)
Side Information (Optional)
Side information (e.g. PPI networks) in target and source can be incorporated via kernelization
(B) Kernelized cross-species matrix factorization with bias (K-XSMF-b)
Per-gene biases
Latent factors
b
U^T
V
b'
F^T
H
Similarity measure between source and target genes is used to link latent factors and biases and transfer knowledge across species
sim(TCGCGGAGGCTT / TCTCGGCGGCTT)
K_V
K_H
PPI kernels can be used for regularization
(C) Parameter learning
Learn latent factors and biases
Loss is minimized with gradient descent to learn latent factors
$\theta = \{U, V, F, H, b, b'\}$
$\mathrm{argmin}_{\theta}\, \mathcal{L}(\theta; X, Y, \lambda)$
λ
Automatic hyperparameter optimization maximizes imputation performance
(D) Outputs
Imputed GI scores
$\hat{x}_{ij} = u_i^T v_j + b_i$
Target
Source
$\hat{y}_{ij} = f_i^T h_j + b'_i$

Figure 3.1: (A) Extensible Matrix Factorization (EMF) is a composable framework of matrix factorization models that takes as input partially observed matrices in a target species and, optionally, a source species. Here we use K-XSMF-b, one realization of EMF, illustrates the range of EMF models as they apply to imputing genetic interactions (GIs). To better impute GIs, K-XSMF-b exploits *cross-species* information from a given gene-gene similarity measure (e.g. BLAST). (B) K-XSMF-b exploits *side information* via regularization from appropriately chosen kernels (e.g. from PPI networks, GO annotations, and other sources; blue box). To model *per-gene biases* in mean GI Scores, K-XSMF-b introduces bias terms for the target and source species (orange box). Similarities from the provided cross-species similarity measure are used to link and regularize both biases and latent factors across species. (C) Latent factors and biases are learned using gradient descent. Importantly, to ensure best possible test-time performance, `hyperopt` is used to automatically select optimal hyperparameters [83]. (D) After hyperparameters are selected and latent factors and biases are learned, missing GIs can be imputed.

First, K-XSMF-b models per-gene biases in target and source GIs, and thus aims to minimize:

$$\mathcal{L}_1 = \sum_{(i,j)\in\Omega_x} (x_{ij} - u_i^T v_j - b_i)^2 + \lambda_s \sum_{(i,j)\in\Omega_y} (y_{ij} - f_i^T h_j - b'_i)^2. \tag{3.14}$$

Then, given kernels K_V and K_H over source and target genes, K-XSMF-b regularizes its factorization with:

$$\mathcal{L}_2 = \mathrm{tr}\left(V K_V^{-1} V^T\right) + \mathrm{tr}\left(H K_H^{-1} H^T\right) + \|U\|_F^2 + \|F\|_F^2. \tag{3.15}$$

Finally, loss terms that link latent factors and biases across species $\mathcal{L}_{\mathbf{x}}$ and $\mathcal{L}_b$, as in Eqs. (3.5) and (3.11), are added. Given hyperparameters λ_s, $\lambda_{\mathbf{x}}$ and λ, the full loss function that of the K-

XMSF-b aims to minimize is:

$$\mathcal{L} = \mathcal{L}_1 + \lambda_{\mathbf{x}}\mathcal{L}_{\mathbf{x}} + \lambda(\mathcal{L}_2 + \mathcal{L}_b). \tag{3.16}$$

Thus, K-XSMF-b is a fully featured EMF model that simultaneously exploits cross-species information, side information in the source and target, and models the effect and conservation of per-gene biases.

3.2.3 Parameter Learning and Hyperparameter Selection

Each loss term in the various EMF models described above is differentiable. Thus all the objective functions we work with are amenable to typical gradient-based optimization algorithms. In this work we use the popular method ADAM to learn our models [92].

For all models, all input GI scores in the target and source species (where applicable) are normalized to zero mean and unit variance prior to training. Accordingly, for imputed GI scores, this normalization operation is inverted prior to evaluation. The input cross-species similarity score matrix is also scaled element-wise to $[0, 1]$ prior to training.

We take care to ensure fair benchmarking of every MF model in our experiments. We use `hyperopt` to automatically tune and optimize model hyperparameters to maximize the performance of each benchmarked model [83].

Furthermore, a consistent early-stopping strategy is adopted for all models for the same purpose. Models are early-stopped when the R^2 score evaluated on the validation set fails to decrease for five consecutive iterations, or when the user-defined maximum number of iterations is reached.

For each combination of model, dataset, and proportion of training examples used, a validation set of 10% of training examples is first held out. Using this validation set, `hyperopt` (50 iterations) is used to determine the best hyperparameters to be used across multiple repeats.

3.2.4 Evaluation

In this work, we primarily evaluate imputation performance of models using the R^2 measure (the coefficient of determination), in contrast to prior studies that have used Pearson's ρ (the correlation coefficient). We report Pearson's ρ where context and comparison to prior work is necessary.

In evaluating EMF models, we rely on R^2 because it is a measure of *goodness-of-fit* while Pearson's ρ is a measure of *correlation* — and the latter does not imply the former. Critically, R^2 correctly rejects a model that systematically mis-estimates the magnitude of predictions while Pearson ρ fails to do so. For example, consider a poor model that systematically predicts values that are exactly half of the ground truth. Despite being very wrong, such a model would output values that have perfect correlation but low (or even negative) R^2 when compared to ground truth.

When imputing GI scores, the difference between goodness-of-fit and correlation is critical because extreme classes of GIs (e.g. synthetic lethal interactions) are binarized on strict numerical thresholds in the literature [76]. Thus, a model that systematically underestimates GI scores will also systematically under-report the number of predicted extreme GIs.

On the data set from Costanzo et al. [76], we also evaluate the ability of models to correctly classify "negative GIs" (analogous to synthetic sick or lethal) as defined by [76]. We follow the protocol taken by Ma et al. [79] and Yu et al. [80] for these evaluations. That is, we impute interaction scores directly and, afterwards, vary the binarization threshold to compute the area under the precision-recall curve (AUPR).

Unless stated otherwise, we use Monte Carlo cross-validation to evaluate all experiments. For training and evaluation, GIs for *unique gene-pairs* are partitioned. Following Zitnik and Zupan [91], if two genes A and B are in both rows and columns of an input matrix and two values are imputed (e.g. across the diagonal of the imputed matrix), the imputed scores are averaged for evaluation. All reported evaluation measures are averaged over 10 random repeats.

3.2.4.1 Comparison against existing factorization based methods

In our experiments, we compare our cross-species models against two existing factorization-based models. We compare our models to KPMF [89], a model originally developed for recommender systems. To the best of our knowledge, we are the first to use KPMF to impute GIs. We also compare our models to Network Guided Matrix Completion, a method that incorporates network information (from PPI networks or the Gene Ontology) to impute GIs [91]. We note that both KPMF and NGMC do not account for per-gene biases and cannot incorporate information across species. Zitnik and Zupan [91] also did not evaluate NGMC on genome-scale datasets available at the time of publication.

Hyperparameter optimization described in Section 3.2.3 is applied to both KPMF and NGMC. The same early stopping criterion described in Section 3.2.3 is applied to KPMF but not NGMC; all NGMC models run for 500 iterations.

3.2.4.2 Comparison against Gene Ontology based methods

We also compare EMF to DCell, the current state-of-the-art neural-network based approach developed by Ma et al. [79], and Ontotype, the best non-deep-learning based method developed by Yu et al. [80]. Both methods featurize labels from GO to predict GIs in baker's yeast at genome-scale.

We downloaded published data and predictions from Yu et al. [80] and Ma et al. [79], and for these comparisons evaluate EMF using the same 4-fold cross-validation procedure carried out by these studies.[1]

3.2.5 Implementation

EMF models are implemented using TensorFlow [93]. For NGMC, we use the implementation released by the authors [91]. `Snakemake` is used extensively to configure and manage experiments [94]. Models and scripts to reproduce experiments are publicly available at: `https://github.com/lrgr/emf`.

3.3 Results

Armed with the EMF framework defined in the previous section, we now evaluate it in three ways. First, we demonstrate its superiority to the state-of-art methods for predicting genetic interactions (GIs). Next, in chromosome biology GI datasets for baker's and fission yeast, we perform a systematic ablation analysis to identify the components of EMF that capture additional signal to better impute GIs. And finally, we apply EMF to impute GIs on genome-scale datasets in both yeast species.

3.3.1 Data

Our experiments were performed on two pairs of GI datasets from baker's and fission yeast. The first pair of GI datasets consists of published epistatic miniarray profiles (E-MAPs) for chromosome

[1]Published predictions from these studies were not stratified by fold. Thus, while we follow the same experimental procedure, we train our models on different folds.

biology genes in baker's and fission yeast [29, 51].

The second pair are genome-scale GI datasets in baker's and fission yeast. Ryan et al. [67] produced an E-MAP covering ~60% of all non-essential genes in fission yeast. Costanzo et al. [76] produced a synthetic genetic array (SGA) covering ~75% of all non-essential genes in baker's yeast. We note that in the SGA dataset for baker's yeast, a confidence measure (P-value) computed from technical replicates is also assigned to each reported GI score [95].

3.3.1.1 Genetic interaction scores

All four datasets measure GI scores with respect to cell growth. Each yields a matrix of real valued GI scores where index (i, j) corresponds to the interaction of column gene i and row gene j. For chromosome biology datasets, matrices of GI scores are symmetric. For genome-scale datasets, the GI scores between a set of array (columns) and query genes (rows) are measured and the set of array and query genes have non-zero intersection. Thus a unique gene-pair can correspond to two measured GI scores. We follow Ma et al. [79] to associate each unique gene pair to a unique GI score. That is, if a gene-pair corresponds to GI scores of opposite signs, the GI scores are discarded. Otherwise, for baker's yeast the GI score with lower P-value is retained, and for fission yeast the average GI score is retained (as significance is not reported for this dataset).

We note that E-MAPs and SGAs both quantify a GI score between a pair of genes using similar principles. Both technologies use imaging to quantify the fitness of the double and corresponding single mutants. The GI score is then defined to be the deviation of the fitness of the double mutant from the multiplicative product of the fitnesses of the single mutants [65]. However, since E-MAPs and SGAs are different technologies, raw GI scores cannot be directly compared. Hence, we applied the normalization strategy described in Section 3.2.3.

All datasets are restricted to GIs between non-essential genes only.

For data from Costanzo et al. [76], we follow Ma et al. [79] and remove GIs involving temperature sensitive alleles. The matrices of GI scores for the datasets described above are all partially observed.

The percentage of missing entries and size of each processed dataset are listed in Table 3.1.

Species	**Reference**	**# Rows**	**# Columns**	**% Missing**
Baker's yeast	[51]	664	664	32%
	[76]	3,885	1,377	19%
Fission yeast	[29]	536	536	21%
	[67]	1955	862	16%

Table 3.1: Summary statistics for genetic interaction datasets.

For experiments with data from Costanzo et al. [76], all GI scores regardless of significance were used for training. We report imputation performance on all scores as well as scores restricted to significant pairs ($P < 0.05$).

3.3.1.2 BLASTp, protein sequences, and PPI networks

We use BLASTp bitscores between proteins sequences across species as the similarity measure for cross-species EMF models. Protein sequences for baker's and fission yeast were downloaded from the Saccharomyces Genome Database and PomBase [96, 97] and used to compute bitscores between genes in the rows of target and source species data. Bitscores between proteins without available sequences were set to zero. For chromosome biology datasets [29, 51], the set of downloaded sequences covered 99.2% and 99.3% of baker's and fission yeast genes. For genome-scale datasets [29, 76], downloaded sequences covered 86.9% and 99.8% of baker's and fission yeast genes.

We downloaded protein-protein interaction (PPI) networks for baker's and fission yeast from

BioGRID database version 3.5.174 [98]. These PPI networks were used for all models that incorporated side information. PPI networks were restricted to genes in the columns of each GI dataset. For chromosome biology GI datasets, the PPI networks covered 99.1% and 73.5% of genes in baker's and fission yeast. For genome-scale GI datasets, the PPI networks covered 99.6% and 78.3% of genes in baker's and fission yeast. Singletons were then added for genes in GI data missing from PPI networks.

3.3.2 Evaluated models

In our experiments, we seek to investigate how composable components of EMF affect, and ultimately improve, GI imputation. We implement seven model instances of the EMF framework by progressively adding components that, model per-gene biases, link factorizations across a target and source species, and regularize with side information within each species.

Of the seven EMF models, four are *single-species* models that factorize GI data in the target species only[2]:

- **Matrix Factorization (MF)** is the simplest matrix factorization model. It uses the optimization objective described in Section 3.2 and (3.2)) [84, 85].

- **MF with bias (MF-b)** is the extension to MF that incorporates a latent bias term, b, as described in Section 3.2.1.2 and (3.9). Also, ℓ_2 regularization over b is also added to prevent overfitting [85].

- **Kernelized Probabilistic Matrix Factorization (KPMF)** is the model developed by Zhou et al. [89] with regularizers described in Section 3.2.1.4 and (3.12). Here, $\mathcal{L}_{kt}$ from (3.12)

[2]We note that MF, MF-b, and KPMF were first introduced by other researchers.

replaces the corresponding ℓ_2 regularizers in MF.

Of these models, only MF has been used to impute missing GIs in prior work [91].

To the best of our knowledge, our work is the first to evaluate MF-b and KPMF for imputing GIs. For context, we also compare EMF models to NGMC, a matrix factorization based model not that is not encompassed by the EMF framework but does utilize PPI networks for GI imputation [91].

Additionally, we implement one other single-species model that is a novel extension to KPMF that has not been explored in prior work:

- **KPMF with bias (KPMF-b)**, is an extension of the KPMF model that incorporates per gene biases. KPMF-b applies the same modification to KPMF that MF-b does to MF.

To determine how EMF components which incorporate cross-species information capture complementary signal to improve performance, we evaluate three *cross-species* models of increasing complexity. These cross-species models use BLASTp bitscores to link the factorizations of GI scores in a target and source species to better impute GIs in the target. One model additionally uses PPI network information in each species to regularize factorizations. Another both models and links per-gene biases across species *and* incorporates PPI network information:

- **Cross-species Matrix Factorization (XSMF)** is the cross-species model described Section 3.2.1.1 with loss function as specified by (3.8).

- **Kernelized XSMF (K-XSMF)** is the cross-species model described Section 3.2.2 with model components that correspond to bias terms removed. Per-gene biases are not fitted and $\mathcal{L}_b$ is removed from the loss function defined by (3.16).

- **K-XSMF with bias (K-XSMF-b)** is the fully featured cross-species model described in Section 3.2.2.

A summary of the data and components used by NGMC and each EMF model is given in Table 3.2.

Algorithm		**Target Species**		**Source Species**		
Name / short description	**abbr.**	**Bias**	**PPI**	**GIs**	**Bias**	**PPI**
Matrix Factorization	MF	-	-	-	-	-
MF with bias	MF-b	Y	-	-	-	-
Kernelized Probabilistic MF	KPMF	-	Y	-	-	-
Network Guided Matrix Completion	NGMC	-	Y	-	-	-
KPMF with bias	KPMF-b	Y	Y	-	-	-
Cross-species MF	XSMF	-	-	Y	-	-
Kernelized XSMF	K-XSMF	-	Y	Y	-	Y
Kernelized XSMF with bias	K-XSMF-b	Y	Y	Y	Y	Y

Table 3.2: Overview of benchmarked MF models. For each model, ‘Y’ indicates the additional MF component and side information used.

Our focus is on imputing GIs in the target species; so for the three cross-species models, all available GIs in the source species are used for training while varying the proportions of the target species’ GIs are held out for evaluation. For example, when baker’s yeast is the target species, we train on the entire fission yeast dataset and part of the baker’s yeast dataset, holding out some of the baker’s yeast dataset for evaluation.

For all kernelized models, PPI network information is incorporated using regularized Laplacian kernels. For KPMF, KPMF-b, K-XSMF, and K-XSMF-b, the kernels used for target species factors are the identity matrix for K_U and the regularized Laplacian for K_V, respectively. Likewise, where applicable, the kernels for source species factors are the identity matrix for K_F and the regularized Laplacian for K_H, respectively. We note that hyperparameters for regularized Laplacian kernels used are also optimized via the same procedure described in Section 3.2.3.

During hyperparameter optimization, the maximum rank searched for in all matrix factorization algorithms is set to $k = 100$ and $k = 200$ for chromosome biology and genome-scale datasets, respectively. The ranges searched over and the selected hyperparameters for all the above-mentioned models are listed in our publicly available implementation.

3.3.3 Matrix factorization outperforms state-of-the-art Gene Ontology based models in baker's yeast

Surprisingly, EMF outperforms the best deep learning-based method for GI prediction even when using strictly less data.

We demonstrate this by establishing a baseline comparison, and contextual correspondence, between a simple EMF model that relies on GI data alone and Gene Ontology (GO) based state-of-the-art models, DCell and Ontotype [79, 80]. Specifically, since we expect mean GI scores across genes in genome-scale datasets to vary greatly, we choose to compare DCell and Ontotype to MF-b, the simplest model within the EMF framework that *only* requires GI data and also models per-gene biases.

We compare MF-b to DCell and Ontotype only in baker's yeast since both have only been applied to genome-scale data in baker's yeast and their predictions are publicly available. We note that DCell and Ontotype cannot predict GIs for *all* ~4.0 million unique gene-pairs with GI scores available from Costanzo et al. [76] because only ~3.3 million gene-pairs can be featurized from GO (as some genes have no annotations). Even though MF-b does not have the same limitation, we nonetheless restrict MF-b to only use the 3.3 million GI scores used by Yu et al. [80] and Ma et al. [79] to perform an apples-to-apples comparison.

Though we argue in Section 3.2.4 that R^2 is a better metric, we report regression performance both in terms of R^2 and Pearson's ρ for context. Ma et al. [79] and Yu et al. [80] only report performance in terms of Pearson's ρ in their work. Following prior studies [79, 80], we also evaluate how well each model predicts extreme GIs and report the AUPR achieved by each model for classifying negative GIs (see Section 3.2.4 for details).

First, following Yu et al. [80] and Ma et al. [79], we evaluate predictions restricted to the subset of GI scores deemed *significant* by Costanzo et al. [76]. When imputing significant GI scores, MF-b outperforms DCell and Ontotype by 17.5% and 75.1% in Pearson's ρ (0.604 versus 0.514 and 0.345), respectively. In terms of R^2, MF-b more than doubles the R^2 score of Ontotype (0.271 versus 0.112) and outperforms DCell (0.265). Furthermore, when classifying negative GIs, MF-b again outperforms DCell and Ontotype, achieving 18.8% and 66.2% improvement in AUPR (0.570 versus 0.480 and 0.343) over DCell and Ontotype.

Second, we hypothesize that methods that perform better at imputing *all* GI scores may be less sensitive to noise or variability in the data; hence we also evaluate models with respect to all imputed scores. When imputing all GI scores, MF-b achieves double the Pearson's ρ of Ontotype and improves over DCell by 19.0% (0.425 versus 0.191 and 0.358, respectively). When classifying negative GIs, MF-b again doubles the AUPR of Ontotype and improves over DCell by 31.5% (0.267 versus 0.104 and 0.203). Surprisingly, Ontotype and DCell both achieve *negative* R^2 scores while MF-b achieves an R^2 score of 0.187. These results indicate that, on all scores, DCell and Ontotype perform worse than a model that predicts the mean. One reason for this, shown by Fig. 3.2, is that when DCell predicts the sign of a GI score incorrectly, it does so more often with greater magnitude than MF-b.

MF-b and other matrix factorization based models presented in this study are also faster to train. On a machine with an NVIDIA GTX 1080Ti GPU, EMF models take less than 1 minute to train on

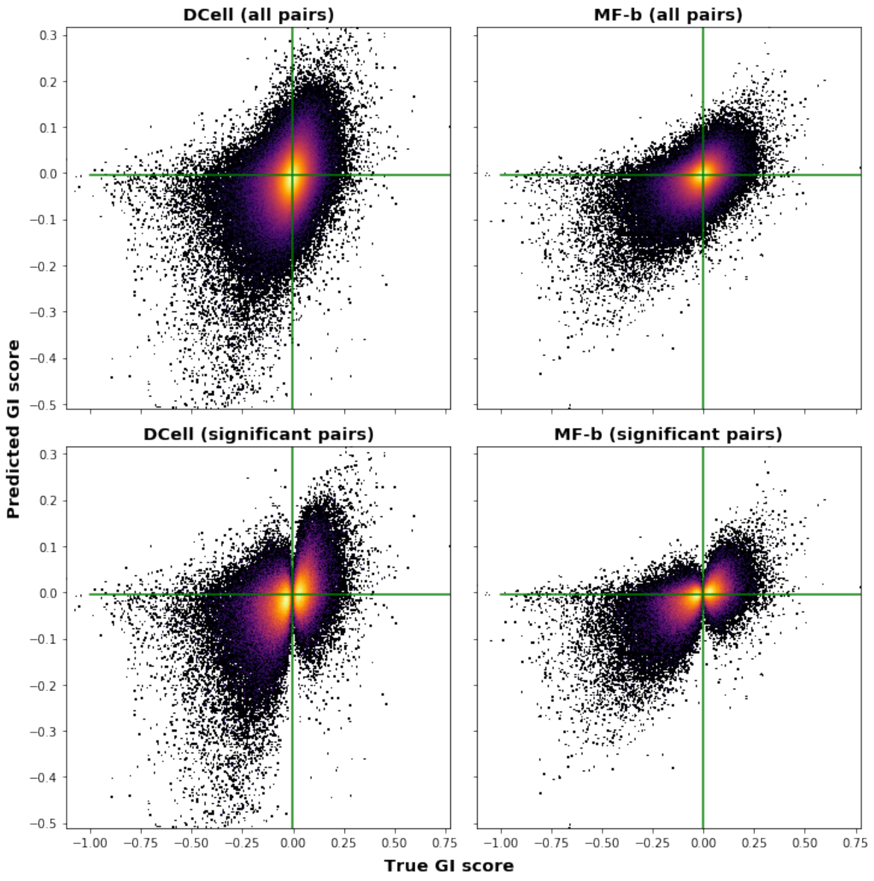


Figure 3.2: True (x-axis) versus predicted (y-axis) GI scores by MF-b and DCell [79]. MF-b is trained with the GI scores for 3.3 million featurizable gene-pairs published by [79] and [80]. In green, $x = 0$ and $y = 0$ are plotted.

the baker's yeast dataset. In fact, the benchmarked MF-b model trains in less than 3 seconds. In comparison, the authors of DCell report in their publicly available software release that the "running time on a standard Tesla K20 GPU takes 2-3 days" [3] on Costanzo et al. [55].

Having established that MF-b, a simple model in the EMF framework, outperforms deep learning and GO-based methods under three different measures, we refrain from comparing other matrix factorization methods to DCell and Ontotype in subsequent sections. Furthermore, since Pearson's ρ fails to detect systematic mis-estimation of GI score magnitudes (see Section 3.2.4), subsequent experiments are evaluated using R^2 only.

[3] `github.com/idekerlab/DCell`

3.3.4 Ablation analysis on matched chromosome biology datasets in baker's and fission yeast

Next, via an ablation analysis, we evaluate how components of the EMF framework affect GI imputation. We perform this analysis on GI datasets for chromosome biology genes in baker's and fission yeast [29, 51]. To compare to prior work, we also compare EMF models to NGMC [91]. To explore the range of settings that may arise in practice, we evaluate models with varying amounts of training data in the target species. These allow us to assess both data-rich and data-scarce settings. Our results demonstrate that EMF models that jointly exploit cross-species and side information consistently impute GIs more accurately. We report these results in Table 3.3.[4]

Algorithm	**% of GIs used in training**							
	Baker's yeast				Fission yeast			
	10%	*25%*	*50%*	*75%*	*10%*	*25%*	*50%*	*75%*
MF	0.054	0.178	0.303	0.380	0.093	0.220	0.370	0.464
MF-b	0.069	0.183	0.308	0.385	0.113	0.234	0.371	0.464
KPMF	0.105	0.215	0.329	0.397	0.119	0.266	0.397	0.472
NGMC	0.050	0.207	0.304	0.329*	0.081	0.256	0.396*	0.479*
KPMF-b	0.102	0.218	0.326	0.393	0.136	0.273	0.391	0.475
XSMF	0.070	0.181	0.304	0.386	0.106	0.232	0.373	0.466
K-XSMF	0.104	0.217	0.327	**0.399**	0.142	**0.278**	**0.405**	**0.480**†
K-XSMF-b	**0.116**	**0.225**	**0.330**	0.397	**0.155**	0.270	0.394	0.476

Table 3.3: R^2 score of imputed versus actual GI scores for chromosome biology datasets in baker's and fission yeast [29, 51]. Models are evaluated with varying proportions of GI scores used during training. The best performing models are indicated in bold. †Standard deviations of best performing model and MF baseline overlap. * Folds that did not converge were excluded from evaluation.

Our results first show that modeling per-gene biases aids GI imputation. The improvement gained by modeling biases is most clear when comparing the performance of MF-b to MF. MF-b

[4]Standard deviations across repeats are published in the supplementary materials that accompany [14].

outperforms MF in all but one experiment. We note that, when cross-species and side information are used, improvement due to modeling biases is less consistent. When 50% and 75% of GIs are observed during training, the difference in performance when adding bias terms to K-XSMF and KPMF is small. However when data is scarce, modeling biases more consistently improves imputation. For example, when 10% of GIs are used during training, K-XSMF-b outperforms K-XSMF by 11.5% and 9.2% in baker's and fission yeast, respectively. In fact, in these scenarios, K-XSMF-b outperforms K-XSMF, and KPMF-b outperforms KPMF, in six out of eight experiments.

Unsurprisingly, all models that exploit side information consistently outperform corresponding models that do not. We highlight some results for the most data-scarce and data-rich scenarios. In baker's yeast, when 10% and 75% of GIs are used, KPMF-b outperforms MF-b by 47.8% and 3.4%, and K-XSMF outperform XSMF by double and by 3.4% percent, respectively. In fission yeast, when 10% and 75% of GIs are used, KPMF-b outperforms MF-b by 20.4% and 2.4%, and K-XSMF outperforms XSMF by 30.2% and 1.1%. We note that while both models use the same side information, KPMF outperforms NGMC in seven of eight experiments across both yeast species.

Moreover, our results show that cross-species models outperform single-species models. Exploiting cross-species information only, XSMF outperforms MF across the board, albeit by a small margin when data is abundant. Again, differences in performances are largest when data is scarce. When 10% of GIs are used, XSMF outperforms MF by 29.6% and 14.0% in baker's and fission yeast, respectively.

Most strikingly, models that exploit cross-species *and* side information (K-XSMF-b and K-XSMF) are the best performing model (bolded in Table 3.3) and outperform the MF baseline without overlapping standard deviations in all but one case. Again, data-scarce scenarios show the largest differences in performance. In baker's yeast, when 10% and 25% of target GIs are used during training, K-XSMF-b outperforms the next best single-species model by 10.5% and 3.2%, respectively. In fission yeast, K-XSMF-b outperforms KPMF-b by 14.0% when 10% of target GIs are used, and K-XSMF outperforms KPMF-b 1.8% when 25% of GIs are used. We highlight that K-XSMF and K-XSMF-b not only exploit cross-species information, but also side information in *both* target *and* source species. These results highlight the utility and versatility of the EMF framework.

It is particularly notable that EMF components (i.e., biases, exploiting side and cross-species information) offer largest improvements in imputation performance when data is scarce. Data-scarce scenarios are most likely to occur when new methods for measuring GIs for new phenotypes or new species are developed.

3.3.5 Results on genome-scale datasets in baker's and fission yeast

Finally, we evaluate a representative set of EMF models on genome-scale GI datasets in baker's and fission yeast [67, 76]. On these datasets, our results show that incorporating cross-species information aids GI imputation when training examples are scarce.

Again, since EMF models do not depend on GO, EMF models are able to impute interactions between *all* 4.0 million unique baker's yeast gene-pairs measured by Costanzo et al. [76] as opposed to the 3.3 million featurizable gene-pairs studied by Yu et al. [80] and Ma et al. [79]. To the best of our knowledge, our study is the first to predict GIs at genome-scale in fission yeast, and GIs for all 4.0 million gene-pairs measured by Costanzo et al. [76] in baker's yeast.

In baker's yeast, as in Section 3.3.3, we evaluate both all imputed scores and the subset of scores deemed to be significant by Costanzo et al. [76]. In fission yeast, we evaluate imputed scores for all held-out gene-pairs since Ryan et al. [67] do not report P-values for measured GI scores. We report these results in Tables 3.4 and 3.5.[5]

Algorithm	**% of GIs used in training**			
	10%	*25%*	*50%*	*75%*
MF	0.049	0.147	0.251	0.316
KPMF-b	0.064	**0.159**	**0.257**	0.317
XSMF	0.062	0.153	0.252	**0.318**†
K-XSMF-b	**0.067**	0.158	0.254	**0.318**†

Table 3.4: R^2 score of imputed versus actual GI scores for EMF models in genome-scale fission yeast dataset [67]. Notation is the same as in Table 3.3.

Algorithm	**% of GIs used in training**			
	10%	*25%*	*50%*	*75%*
MF	0.004 (0.007)	0.088 (0.055)	0.180 (0.133)	**0.267** (**0.189**)
KPMF-b	**0.026** (0.009)	**0.100** (**0.061**)	0.180 (0.126)	0.238 (0.176)
XSMF	0.005 (0.006)	0.084 (0.059)	0.190 (**0.134**†)	0.266 (**0.189**†)
K-XSMF-b	0.019 (**0.011**)	0.085 (**0.061**)	**0.200** (0.130)	0.250 (0.182)

Table 3.5: R^2 score of imputed versus actual GI scores for EMF models in genome-scale baker's yeast dataset [76]. Scores for predictions restricted to significant GI scores as determined by Costanzo et al. [76] appear on the left. Scores for predictions on all pairs to the right in parentheses. Other notation is the same as in Table 3.3.

Perhaps unsurprisingly, when a large amount of data is available, the differences in the best performing models are almost indistinguishable. In fission yeast, XSMF and K-XSMF-b outperforms MF by a small margin when 75% of GIs are used during training. When 50% of GIs are observed during training, KPMF-b outperforms K-XSMF-b by just over 1%. In baker's yeast, when 75% of GIs are observed during training, MF and XSMF are the best performing models and outperform their kernelized counterparts by small margins. Here, one key observation is that in these data rich

[5]Standard deviations across repeats are published in the supplementary materials that accompany [14].

settings, cross-species components of the EMF framework do not impair the single-species matrix factorization models which they extend.

However, when data is scarce, the improved performance of EMF models due to the inclusion of side information and cross-species information is clear. When fewer than 75% of observed GIs are used during training, the best performing EMF models outperform the MF baseline without overlapping standard deviations in all but one case. In fission yeast, when 10% and 25% of observed GIs are used during training, K-XSMF-b and KPMF-b are the best performing models. Further, both cross-species models improve over their single-species counterparts: K-XSMF-b outperforms KPMF-b by 5%, when 10% of observed GIs are used during training, and XSMF outperforms MF by 27% and 4%, when 10% and 25% of observed interactions are used for training.

Likewise, the inclusion of cross-species information and side information aids imputation in baker's yeast when data is scarce. When imputing significant pairs, both KPMF-b and K-XSMF-b roughly quadruple the R^2 score of their non-kernelized counterparts when 10% of GIs are used during training. Here, KPMF-b is clearly the best performing model when 10% and 25% of GIs are used during training. Finally, when imputing all GIs, cross-species models XSMF and K-XSMF-b achieve the best R^2 score, when 10%, 25% and 50% of GIs are used during training.

3.4 Discussion

In this work, we introduce *Extensible Matrix Factorization* EMF, a framework of composable matrix factorization (MF) models for imputing genetic interactions (GIs). The EMF framework unifies several MF strategies for improving imputation. EMF models can explicitly model per-gene biases, and can readily exploit available side information via kernelization. A novel contribution of EMF models is the ability to simultaneously exploit *cross-species* information. Given a cross-species gene-gene similarity measure, EMF models can link factorizations in a *source* and *target* species to better impute missing values in the target.

Surprisingly, even a simple EMF model outperforms the state-of-the-art method for GI prediction. This simple model only requires GIs as input and does not require labels from the Gene Ontology. Via an ablation analysis in chromosome biology GI datasets in baker's and fission yeast, we show how components of the EMF framework improve GI imputation. Furthermore, our results show that EMF models are also effective in genome-scale datasets in both yeast species. To the best of our knowledge, our study is the first to impute GIs in fission yeast at genome-scale.

In sum, the EMF framework highlights the versatility, and surprising utility, of MF based approaches. Our results show that components in the EMF framework that exploit cross-species information are most effective when data is *scarce*. We also emphasize that data scarcity is relative. For example, 10% of available data in baker's yeast equates to approximately 400,000 observations, which is more than have been measured in all but a handful of species. Thus, we expect MF based approaches like EMF to be invaluable for efforts to map GIs in new species. In these scenarios, the incorporation of data across multiple contexts, be it species or phenotypes, may be fruitful if not necessary. Though not the focus of this work, we also anticipate that the performance of cross-species models could be improved via other cross-species similarity measures and other methodological optimizations (e.g., combining kernels via multiple kernel learning [99]).

Part II

Sequences

Chapter 4: ScalpelSig: Designing targeted genomic panels from data

Disclosure

This Chapter presents work published by Franzese, Fan, Sharan, and Leiserson [17] with abridged results. Other sections are presented with minimal changes. Franzese led the study. Fan wrote Section 4.3 and proved Lemma 4.1.

4.1 Background and motivation

Over the past few decades, research has revealed that cancer is a disease characterized by the accumulation of mutations [100, 101, 102, 103]. Over the lifetime of an organism, cells acquire mutations at random positions in the genome. Certain mutations disrupt the function of cellular systems, and if certain cellular systems are disrupted simultaneously, a cell can lose its ability to regulate its rate of reproduction. Dysregulation of the reproductive cycle is one of a handful of "hallmark" qualities [104], which together transform a healthy cell into a cancer cell. These hallmark qualities are shared by all cancers, but they can be caused by random mutations in many respective genes. Further, many such genes are mutated only at low frequency, implying that there are numerous combinations of mutated genes that can lead to cancer (e.g. see the analysis by Lawrence et al. [105]). The randomness involved in determining a cell's path to the disease state is responsible for part of the difficulty of treating cancer [106]. Since cancer genomes vary so widely, the set of therapeutic targets does also, necessitating a diverse set of treatment strategies and engendering difficult decisions about which one to employ for a given patient.

One silver lining to this situation is that the random process by which mutations are acquired in the genome is not uniform — it is patterned [107]. For example, mutations acquired by smoking tend to produce a different pattern than those acquired from UV radiation [108], or those acquired endogenously through errors made during genome replication [109], etc. Recent work demonstrates that identifying which of these mutational processes are active in a tumor genome provides a useful basis with which to categorize the diverse landscape of tumor phenotypes [110]. Indeed, such activity can serve as an effective biomarker in the clinic. Past work has shown that certain mutational processes can indicate a tumor's vulnerability to particular therapies. One prominent example is mismatch repair deficiency, which can indicate the effectiveness of checkpoint inhibitor immunotherapy [111]. A second is homologous recombination repair deficiency, which can indicate effectiveness of PARP inhibitor therapy [112, 113].

Designing and applying methods for the detection of mutational processes is an ongoing effort in the computational cancer biology community. The most well known approach was pioneered in 2012 by Nik-Zainal et al. [114], who utilized machine learning methods to extract the "signature" patterns of several mutational processes from aggregated tumor genome samples. Alexandrov et al. [115] formalize a *mutational signature* as a probability distribution over a set of mutation categories — that is, they suppose that a given mutational process can be identified by the frequency with which it causes each type of mutation. Their landmark paper utilized distributions over 96 mutation categories, defined by all possible single base substitutions with trinucleotide context. To infer the signatures from the data, they formulated a simple linear model of a tumor's mutations. In their model, a relatively small number of mutation signatures are shared across tumors, and the mutations of each tumor are given as a linear combination of these shared signatures plus some noise. To realize this model, they apply non-negative matrix factorization (NMF) to genome-wide mutation counts on a large cohort of tumor samples. When applied to this problem, NMF infers a set of mutational signatures, as well as the activity of each signature on each genome in the cohort (often called the *exposure*). While other learning algorithms and schemes for categorizing mutations have been employed, this approach remains the most common method of signature extraction in mutational signature analysis (MSA) studies [116, 117, 118]. The current state of the art for single base substitution mutational signatures identified 49 signatures, extracted from an analysis of 23,829 tumors. Recent work has highlighted mutational signatures as a powerful diagnostic tool for clinical use, with several signatures implicated as potential therapeutic biomarkers [119, 120].

While these results are indicative of great promise for the future of cancer treatment, current clinical treatments are largely unable to take advantage of these advances. This is primarily due to the outsized sequencing requirements of the computational methods used in MSA studies [115, 118] relative to current clinical sequencing practices [121, 122]. The "gold standard" data source for MSA studies is whole-genome sequencing (WGS). This is because WGS gives a complete and unbiased picture of the mutations present in a tumor genome, which affords increased accuracy during signature extraction. When WGS data is not available or in short supply, MSA studies also commonly use whole-exome sequencing (WES) as a data source. WES takes a subset of the genome that is smaller but still sizable — it is thus cheaper to sequence but still provides utility for signature extraction. Almost all MSA studies use WGS or WES to obtain mutation counts in tumor genomes. By contrast, WGS and WES are *unavailable* to most cancer patients — current clinical sequencing practices are commonly limited to *targeted sequencing*. This term refers to precisely-aimed therapeutic assays that sequence small pieces of the genome with known biological importance. This can provide valuable information about particular genes of interest, but provides a much more limited picture of the distribution of mutations on the genome which MSA seeks to assess. Concretely, targeted sequencing assays identify approximately $1000\times$ fewer mutations compared to WGS [123] and approximately $100\times$ fewer mutations than WES. Further, while there have been calls to make large-scale sequencing assays clinically available, ramping up production of sequencing data in the clinic will take time and resources. Despite the decreasing costs of sequencing itself, providing large-scale genomic sequencing to patients requires infrastructure for analysis, interpretation, and storage of the resulting data. Thus, WGS in particular is unlikely to be offered as a routine clinical diagnostic for many years to come [123]. In the meantime, the therapeutic impact of MSA in the clinic is tied to the important open problem of inferring signature activity from targeted sequencing assays.

A few recent studies have designed methods to detect mutational signature activity from clinically accessible targeted sequencing assays. Campbell et al. [124] found that in patients with hypermutation, panel data could be used to infer exposures using standard methods [125]. Gulhan et al. [126] introduced SigMA to detect signature activity indicative homologous of recombination repair deficiency, with a specific focus and application to gene panel data. Sason et al. [127] introduced Mix to infer mutational signatures and their exposures in clusters of patients for use on datasets with few mutations per patient. In general, these studies make methodological modifications to account for the limited sample of the genome afforded by targeted sequencing assays. Even still, these methods are constrained by *which* regions of the genome are sampled. In particular, these works have sought to detect signature activity from two existing targeted sequencing assays: the Memorial Sloan Kettering Integrated Mutation Profiling of Actionable Cancer Targets (MSK-

IMPACT) [121] and FoundationOne Panel Sequencing [122]. These panels are designed to identify specific *actionable* mutations, which are most commonly found in genes linked with cancer. This goal is fundamentally different from the goal of MSA, which seeks to analyze the *distribution* of mutations over the genome. Accordingly, such panels likely provide a biased view of the true mutation distribution. This is evidenced by a recent prominent study which showed that mutations in cancer genes (which are common locations of actionable mutations) are subject to powerful selective pressure due to their impact on the fitness of a tumor. The authors showed that this can result in distributions of mutations that are not representative of the underlying mutational signatures [128]. Thus it is likely that other regions of the genome are more suitable for interrogating the genome-wide distribution of mutations.s

The problem of designing new targeted sequencing assays tailored to the goals of MSA has been explored in a very limited capacity. One previous study contained a short analysis to identify a panel-sized set of genes which could be used to detect one specific mutational signature [129]. This analysis did not consider non-coding regions as candidates for the panel, and generally was not the main focus of the paper. Perner et al. [130] recently introduced mutREAD, an assay for detecting mutational signature activity, but their approach concerns the method by which the DNA is sequenced rather than panel construction. To our knowledge, no existing study tackles the generalized problem of identifying a panel-sized set of genome regions that are suitable for the detection of signature activity. We take aim at this problem in the present study.

4.1.1 Contributions

In this study, we present ScalpelSig, an algorithm that learns from data to design *genomic panels* optimized for the detection of activity from an arbitrary mutational signature. We use the term genomic panel to refer to a set of genome regions (including both coding and non-coding regions, in contrast to a *gene* panel) whose sequence can be used as a biomarker. We constructed ScalpelSig with the explicit goal of increasing clinical access to MSA, and as such all panels considered in the present study are small enough to be sequenced and analyzed using current clinical infrastructure. To our knowledge, the ScalpelSig method includes two novel contributions to the problem of panel design for mutational signature detection. First, ScalpelSig considers the whole genome (not just coding regions), and second it considers arbitrary mutational signatures rather than just those of homologous recombination repair.

We train ScalpelSig on a large cohort of breast cancer whole genome sequences, and evaluate its performance on held out data. Performance is evaluated by extracting signature exposures from the discovered panel regions with standard methods, and comparing the signature activity within panel regions to the ground truth of genome-wide signature activity. This performance is compared against two benchmarks: the MSK-IMPACT panel and a randomized baseline. Panels designed by ScalpelSig afford superior accuracy for signature detection in five out of six examined signatures. We additionally analyze the generalizability and robustness of our algorithm. We find that ScalpelSig's increased accuracy over baselines is maintained on an independent breast cancer dataset, and for a wide variety of parameterizations.

4.2 Preliminaries

For the purposes of this work, we shall refer to a mutational signature (or *signature* for ease of exposition) as is seminally defined by Nik-Zainal, Alexandrov, and colleagues [114, 115, 116]. A signature is a multinomial distribution over a set of *mutation categories*. The most commonly studied mutation categories are single base substitutions (SBS) and their immediate $5'$ and $3'$ flanking bases [115], leading to 96 categories arising from six canonical SBS categories C>G, C>A, C>T, T>A, T>C and T>G, and four possible $5'$ and $3'$ flanking bases. For example, the A[C>T]G mutational category refers to a C>T SBS immediately flanked by an A ($5'$) and a G ($3'$).

Each tumor or *sample* (belonging to a patient), whose genome is sequenced, can be summarily described by a 96-dimensional count vector of observed single base substitutions belonging to each mutational category. Mutational signature analysis (MSA) assumes that each observed mutation is emitted by a unique mutational signature, and it is generally assumed that only a few signatures are *active* in a tumor and in a cancer type. Briefly, MSA usually involves (1) deriving the distributions of the latent mutational signatures active in a cohort of samples; and/or, (2) determining the *exposure* each sample has to each signature — the proportion of mutations in each sample emitted by each signature.

In this work, we consider problems relating to the latter, more clinically relevant problem where a candidate set of potentially active mutational signatures in each sample is given and known. We perform analysis with respect to a canonical and widely adopted set of signatures determined by the Catalogue of Somatic Mutations in Cancer (COSMIC) ver. 2, and will always refer to (and index) COSMIC signatures by their numerical names (e.g. Signature 1) [131].

4.3 Methods

A panel optimized for detecting signature activity should consist of a small set of genomic regions which, when sequenced, allow the accurate identification of *genome-wide* patterns of mutations. Initially, one may ask whether this is a feasible goal. For intuition, we refer to previous work which shows a strong relationship between cancer type and regional mutation density [132], as well as other regionally varying chromatin features [133]. Further, many mutational signatures are tied to molecular mechanisms which vary in their activity over the genome [134, 135]. These findings suggest that some mutational processes may prefer to cause mutations in *specific* regions of the genome. Therefore, some regions of the genome may be far more informative than others for assaying the activity of these processes and their associated signatures.

Finding these informative regions poses a significant challenge, in part due to the technical attributes of NMF, the most widely-used algorithm for signature extraction [116, 117]. Notably, NMF depends only on genome-wide mutation counts as input. Thus, NMF determines only the *counts* of mutations attributed to each signature and is agnostic to the *location* of mutations in the genome. This presents a barrier for understanding the regional distribution of signature activity.

Further, NMF is computationally intensive, and its resource requirements are amplified within the present use-case. Broadly, the outputs of NMF lack robust structure: solutions are non-unique, the loss surface is non-convex, and several random initializations of the algorithm or specialized initialization approaches [136] are required to obtain a reliable result. Thus, it is computationally intractable to utilize NMF to compute signature activity and explore the combinatorial space of possible panels.

One way to respond to these challenges is by simplifying the problem. Instead of finding regions that assay the activity of all signatures at once, we break the problem down into distinct binary classification tasks. We reason that in clinical applications, signature activity serves primarily as a biomarker to decide whether a patient should recieve a particular tailored treatment. In this use case, it is more important for a panel to detect if one specific signature has *substantial* activity on a given genome, rather than an estimate of the absolute number of mutations (or exposure) attributed to a collection of signatures. This idea guides the construction of our algorithm.

We introduce ScalpelSig (Scalar projection Panels for mutational Signatures), a method for discovering genomic panels to detect mutational signature activity. Given a mutational signature, ScalpelSig designs a genomic panel optimized to distinguish samples where said signature is (substantially) *active* from those where it is *inactive*. ScalpelSig designs such a panel by selecting a set of discerning *windows* over the genome. The ScalpelSig algorithm can be briefly described in three steps.

1. ScalpelSig divides the genome into non-overlapping windows of a fixed size.
2. ScalpelSig computes, with a *window scoring function*, a heuristic measure of mutational signature activity in each window across a given cohort of samples.
3. ScalpelSig combines the highest scoring windows to generate and output a genomic panel of a given fixed size.

4.3.1 ScalpelSig: Scalar projection Panels for mutational Signatures

In order to efficiently optimize over the combinatorial space of possible panels, ScalpelSig uses *scalar projection* as a heuristic measure of mutational signature activity. Unlike typical methods for detecting signature activity, scalar projection has a closed form algebraic expression, and can be computed deterministically in constant time. It also has robust structural guarantees — in particular it is a linear transformation. These properties make optimization tractable, and we show that scalar projection is indeed a reasonable measure of signature activity. For vectors u and v, we write the scalar projection of u onto v as: $\text{proj}_v(u) = \frac{1}{||v||}\langle v, u\rangle$.

Notably, the scalar projection of mutation category counts onto a mutational signature gives the magnitude of the least-loss signature activity vector (i.e. the vector that results from scaling a mutation signature by its exposure), if said signature were the only active signature in the sample. We formalize this result in Lemma 4.1.

Lemma 4.1. *Given any mutation count vector c and a signature vector q, the magnitude of the least-loss exposure vector in the direction of q is given by the scalar projection of c onto q, written* $\operatorname{proj}_q(c) = \frac{1}{||q||}\langle q, c\rangle$.

Proof. We seek the exposure a such that the residual between c and aq (the exposure *vector*) is minimized. Writing $||\cdot||$ to mean the 2-norm, we wish to solve $\min_a ||c - aq||$, or equivalently:

$$\min_a ||c - aq||^2. \tag{4.1}$$

Writing (4.1) as an inner product, yields:

$$||c - aq||^2 = \langle c - aq, c - aq\rangle = a^2||q||^2 - 2a\langle q, c\rangle + ||c||^2. \tag{4.2}$$

Equation (4.2) is quadratic in a. Thus, we solve for a by taking the derivative with respect to a and setting it to be 0, which gives $0 = 2a||q||^2 - 2\langle q, c\rangle$. Rearranging, we find the solution, $a = \frac{\langle q,c\rangle}{||q||^2}$. Notice that a is indeed the least-loss exposure and is non-negative since q and c are non-negative.

The exposure *vector* aq is given precisely by the projection of c onto q. Writing the unit vector in the direction of q as $\hat{q} = \frac{q}{||q||}$, we get,

$$aq = \frac{\langle q, c\rangle}{||q||^2} q = \frac{\langle q, c\rangle}{||q||} \frac{q}{||q||} = \operatorname{proj}_q(c)\, \hat{q}. \tag{4.3}$$

Thus, the *magnitude*, $\text{proj}_q(c)$, of the least-loss exposure *vector*, aq, relates to the *value* of the least-loss exposure, a, by a constant factor, $\frac{1}{||q||}$, with $a = \frac{1}{||q||}\text{proj}_q(c)$. $\square$

Given a mutational signature q, a training set S of samples with subset A that has signature q active, and a maximum panel size N, ScalpelSig designs and outputs a panel, $\mathcal{P}$, that distinguishes samples with substantial signature activity from those without. ScalpelSig designs such a panel by selecting an optimally-scoring subset from the set of all non-overlapping genome windows of a given fixed width.

Let a mutational signature q be a vector representing a multinomial distribution over the standard 96 mutation categories described in Alexandrov et al. [115], and let c_w^i represent the 96-dimensional vector of mutation category counts that fall within genome window w for patient i. We define a panel $\mathcal{P}$ as a set of genome windows. Then, we denote the mutation category counts for patient i over panel $\mathcal{P}$, to be $c_{\mathcal{P}}^i = \sum_{w\in\mathcal{P}} c_w^i$.

Using scalar projection as a heuristic for signature activity, ScalpelSig seeks to find a panel where the estimated activity of q is high only in samples where q is known to have substantial activity on the whole genome (i.e. the samples in A). Formally, ScalpelSig solves the following optimization problem to find a genomic panel $\mathcal{P}$ that best detects the activity of signature q:

$$\begin{aligned} \underset{\mathcal{P}}{\text{maximize}} \quad & \sum_{i\in A}\text{proj}_q\left(c_{\mathcal{P}}^i\right) - \sum_{j\in S-A}\text{proj}_q\left(c_{\mathcal{P}}^j\right), \\ \text{subject to} \quad & |\mathcal{P}| = N. \end{aligned} \tag{4.4}$$

By the linearity of scalar projection, we have, for each sample i,

$$\text{proj}_q\left(c_{\mathcal{P}}^i\right) = \text{proj}_q\left(\sum_{w\in\mathcal{P}} c_w^i\right) = \sum_{w\in\mathcal{P}}\text{proj}_q\left(c_w^i\right). \tag{4.5}$$

Thus, the objective (4.4) can be rewritten,

$$\begin{aligned} \underset{\mathcal{P}}{\text{maximize}} \quad & \sum_{w \in \mathcal{P}} \left\{ \sum_{i \in A} \text{proj}_q \left(c_w^i \right) - \sum_{j \in SA} \text{proj}_q \left(c_w^j \right) \right\}, \\ \text{subject to} \quad & |\mathcal{P}| = N. \end{aligned} \tag{4.6}$$

This formulation of the problem allows us to optimize without exploring the large combinatorial space of possible panels. Instead, ScalpelSig determines an optimal panel by simply selecting the *top scoring* N windows for the window scoring function,

$$h(w) = \sum_{i \in A} \text{proj}_q \left(c_w^i \right) - \sum_{j \in SA} \text{proj}_q \left(c_w^j \right). \tag{4.7}$$

Notably, the contrastive definition of this window scoring function naturally mitigates the impact of background noise, i.e. enrichment of signature-associated mutation types for spurious reasons such as underlying nucleotide composition. To see this, observe that if a window contains spurious enrichment of a set of mutation types, we would expect inactive samples to have about as many mutations of these types as active samples. As a result, we would expect the second term in the equation (the sum of scores from inactive samples) to be high, thus discouraging the selection of such a window.

Given that tumors are known to have widely varying mutation rates, it is important to ensure that the panel is not simply tailored to patients and windows with the highest numbers of total mutations. To this end, we introduce a parameter $\alpha \in (0, 1]$ and reparameterize the window scoring function to be:

$$h_\alpha(w) = \sum_{i \in A} \text{proj}_q \left(c_w^i\right)^\alpha - \sum_{j \in SA} \text{proj}_q \left(c_w^j\right)^\alpha . \tag{4.8}$$

Setting $\alpha = 1$ is equivalent to (4.7), whereas setting $\alpha = 0.5$ applies square roots to each of the summed terms. With $\alpha = 0.5$, the scoring function downweights the contribution from samples that have anomolously high projections. This yields a window scoring function that favors windows with high activity in *most* active samples, and not windows with high activity in *only a few* samples in the training set. While arbitrary values of α could be used, in this paper we only consider $\alpha = 1$ and $\alpha = 0.5$.

We graphically illustrate the ScalpelSig algorithm in Fig. 4.1.

4.3.2 Mutational signatures and breast cancer cohorts

In this work we primarily analyze a publicly available cohort of 560 breast cancer genomes [137]. We chose this dataset because of its large number of samples, and because many different mutational signatures are typically active in breast cancer [131], allowing for a varying set of test cases for our framework. Further, the motivation of using mutational signatures as clinical biomarkers is particularly relevant to breast cancer, wherein HR deficiency (and its associated mutational signatures) is a promising biomarker for PARP inhibitor therapy [120]. whole-genome sequencing (WGS) was performed on each sample. The dataset contains approximately 3.5 million total mutations categorized into the standard 96 categories used in Alexandrov et al. [115].

To further assess the generalizability of ScalpelSig, we also evaluate genomic panels learned from the above cohort [137] on a completely held-out cohort of 237 WGS samples from Staaf et al. [138].

4.3.2.1 Computing mutational signature exposures

For the estimation of mutational signature exposures on whole genomes, as well as all ScalpelSig panels and other baselines, we use an in-house, open-source Python implementation of the SignatureEstimation framework [139].[1]

For each sample in each cohort, we compute exposures to mutational signatures taken from COSMIC (ver. 2), a curated mutational signature repository [131]. In this paper, we will always refer to (and index) COSMIC signatures by their numerical names (e.g. Signature 1). Only thirteen of the thirty COSMIC signatures are known to be active in breast cancer: Signatures 1, 2, 3, 5, 6, 8, 10, 13, 17, 18, 20, 26, 30 [131]. Accordingly, from each sample's genome-wide mutation counts, we compute exposures solely to these signatures. From these exposures, we define a sample as *active* for a signature if the exposure of that signature is responsible for 5% or more of the total mutations (compared to all other signatures), and *inactive* otherwise.

Classification accuracy for signature activity detection for ScalpelSig and other baselines are computed from exposures estimated via this framework (see the following section for more information).

4.3.3 Evaluation of panels

For each signature of interest, we evaluate a ScalpelSig panel by how well it detects mutation signature activity in held-out samples. As our primary measure of performance, we ask the following question: given a sample, can we determine whether a signature is active on the *whole genome* by looking solely at panel regions obtained from ScalpelSig?

[1] `https://github.com/lrgr/signature-estimation-py`

We formalize this question with the following classification task: for each sample in the test set predict whether a signature of interest is *active* on the whole genome, given the signature exposure estimated from mutations that fall within a ScalpelSig panel.

4.3.3.1 Comparison of ScalpelSig against MSK-IMPACT and other baselines

We primarily analyze the performance of ScalpelSig on the cohort of 560 samples from Nik-Zainal et al. [137]. Our experiments use stratified sampling to split the data into training and testing sets. In each experiment we report the mean performance across 15 random test/train splits. For each signature and unless otherwise noted, ScalpelSig uses 90% of samples as a training set to design a panel. Afterwards, signature exposures are extracted from panel regions on the remaining samples. We measure how well these panel exposures distinguish active from inactive samples by computing area under the precision-recall curve (AUPR).

To further demonstrate the effectiveness of ScalpelSig panels, we also use Spearman's rank correlation to measure the strength of the relationship between exposures computed solely from mutations that fall within panel regions, and exposures computed from whole genome mutation counts. Our primary set of experiments are performed on 2.5 Mb ScalpelSig panels. This size was chosen to match the size of the MSK-IMPACT panel [121]. We evaluate ScalpelSig with $\alpha \in \{0.5, 1.0\}$ and set the window size to 10 Kb for all experiments.

We perform experiments only for mutational signatures that are active in more than 5% of samples in the breast cancer cohort [116]. We also omit Signatures 1 and 5, since these signatures are known to be endogenous and "clock-like" and are expected to be present in *all* samples [140]. In sum, we perform experiments to evaluate ScalpelSig panels optimized for the detection of each of the six remaining signatures (2, 3, 8, 13, 18, and 30). We report the number of active samples for each signature in Table 4.1.

We compared ScalpelSig panels against three benchmarks, each an alternative approach that sequences fewer bases than WGS: the MSK-IMPACT panel [121], whole exome sequencing (WES), and a randomized baseline. To compare against the MSK-IMPACT panel [121], we identified the subset of mutations which fell within panel regions. We obtained genomic coordinates, including "off-target" positions in non-coding regions, of the MSK-IMPACT panel from Gulhan et al. [126]. To compare against the baseline of whole-exome sequencing, we took the subset of mutations from the dataset which fell within exonic regions as identified by the GENCODE project [141].

We also compare ScalpelSig panels to a randomized baseline — the mean performance of 1000 random panels each 2.5Mb in size. Each random panel is generated by sampling 250 unique windows from 10 Kb non-overlapping windows across the genome. Windows with no mutations in the cohort were removed prior to sampling. The performance of each baseline is evaluated on the same test set as the ScalpelSig panels in each experiment.

4.3.3.2 Evaluation on held-out breast cancer cohort

To establish that the panels discovered by ScalpelSig are potentially applicable in the clinic, we further assess the generalizability of ScalpelSig panels learned from one study to new samples in another. That is, we also evaluate ScalpelSig on a completely held-out cohort of samples from Staaf et al. [138]. Here, we use the entire cohort of 560 samples from Nik-Zainal et al. [137] to train ScalpelSig panels, and evaluate these panels in an unseen cohort of 273 samples from Staaf et al. [138]. For this experiment, we only perform analysis with respect to signatures that resulted in panels that outperform baselines in Nik-Zainal et al. [123] (Signatures 2, 3, 8, 13, and 18).

Signature	Active Samples	Active Samples
2	232	41.4%
3	278	49.6%
8	494	88.2%
13	262	46.8%
18	64	11.4%
30	135	24.1%

Table 4.1: The number of *active* samples in the 560 breast cancers samples for the mutational signatures known to be present in breast cancer. A signature is active if that signature is responsible for 5% or more of the total mutations in the sample. Signatures 6, 10, 17, 20, and 26 are active in fewer than 5% of samples and thus are not considered in this study. Signatures 1 and 5 are also not considered because they are expected to be active in *every* sample [140].

4.4 Results

In the desired use case for a mutational signature panel, a clinician would sequence panel regions and use them to make a judgement about the mutational signatures that have acted on the patient's genome. Accordingly, a good panel is one where signature activity within panel regions is predictive of signature activity genome-wide. Our experiments measure how well a panel makes this prediction with two complementary metrics: (1) we compute area under the precision-recall curve (AUPR) for a binary classification task which asks whether a signature is substantially active in a sample given the exposure of that signature in panel regions; (2) we compute the Spearman correlation between signature exposure in panel regions and signature exposure genome-wide. In the interest of clinical accessibility, we focus on ScalpelSig panels that are roughly equivalent in size to the MSK-IMPACT panel, since this size is evidently practical for clinical use. However, to further characterize ScalpelSig's performance we analyze panels at various smaller sizes. We contextualize our results by comparing ScalpelSig's performance with that of the MSK-IMPACT panel, a random panel, and whole exome sequencing. We show that ScalpelSig improves panel accuracy substantially. In multiple cases, ScalpelSig even outperformed baselines using panels that were less than two-thirds the size of said baselines.

4.4.1 ScalpelSig outperforms baselines

Panels discovered by ScalpelSig outperform the MSK-IMPACT panel and randomized baseline on five out of the six signatures examined (Signatures 2, 3, 8, 13, and 18) in terms of both AUPR of activity/inactivity classification (Fig. 4.2) and Spearman correlation between panel exposures and genome-wide exposures (Table 4.2). The 2.5 Mb panels constructed using the $\alpha = 1$ parameterization of the window scoring function outperformed the MSK-IMPACT panel on four out of six signatures (Signatures 2, 3, 13, and 18). 2.5 Mb panels constructed with $\alpha = 0.5$ outperform the $\alpha = 1$ setting in almost all cases, obtaining better AUPR than both the MSK-IMPACT panel and randomized baseline — with the exception of Signature 30.

Classifications based on signatures extracted from WES are consistently more accurate than the other measurements, but this is to be expected since the exome ($\sim$30 Mb) covers over ten times as much genetic material as the panels ($\leq$ 2.5 Mb). As such, we posit that the performance gap between the random baseline and WES gives a reasonable notion of how much the classification performance can be improved by simply observing more genomic material in a naive fashion (the performance gap between MSK-IMPACT and WES could serve a similar function, but MSK-IMPACT performs worse than the random baseline in most cases). Thus the efficacy of ScalpelSig panels is demonstrated by their ability to partially bridge this performance gap (see Table 4.3).

By this metric, ScalpelSig's increase in performance over the baseline panels ranged from moderate to sizable, bridging at least 7.9% and at most 38.3% of the performance gap between the randomized panel and whole exome sequencing (Table 4.3). If we were to assume, solely for the purpose of ballpark estimation, that performance scales linearly with addition of exomic regions

Signature	ScalpelSig (2.5Mb)	MSK-IMPACT ($\sim$2.5Mb)	Random Panel (2.5Mb)
2	***0.3819***	0.2695	0.2068
3	***0.3883***	0.2123*	0.3138
8	***0.3895***	0.0275	0.1276
13	***0.5749***	0.4517	0.4580
18	0.1482*	0.0309*	0.0215
30	0.0569*	0.0528*	0.0091

Table 4.2: Spearman correlation between exposures computed only from panel regions and exposures computed from whole genome mutation counts. Values shown are mean Spearman correlation coefficients across 15 randomized test and train sets. ScalpelSig is run with $\alpha = 0.5$ for all signatures. In ScalpelSig and MSK-IMPACT columns, values where fewer than half of the trials yielded P-value < 0.05 are marked with an asterisk. The highest value in each row is bolded and italicized — except when most of the trials for that value are not significant.

	AUPR			
Signature	**Whole Exome**	**Random Panel**	**ScalpelSig**	**Gap Bridged**
2	0.7959	0.5785	0.6330	25.1%
3	0.8727	0.6826	0.7191	19.1%
8	0.9338	0.8860	0.8929	14.5%
13	0.9370	0.7408	0.7562	7.9%
18	0.3027	0.1542	0.2111	38.3%

Table 4.3: Comparison of ScalpelSig ($\alpha = 0.5$; 2.5Mb) panels to Whole Exome (30 Mb) and Random Panel (2.5Mb) benchmarks for five out of six examined signatures. The right-most column gives the percentage of the performance gap between WES and the random panel that is bridged by our method.

to the panel, these improvements would represent an increase in performance proportional to the addition of between 2.1 Mb and 10.5 Mb of exomic material to the 2.5 Mb baseline. As noted in the Methods section, the $\alpha = 0.5$ parametrization is designed to obtain results that better generalize outside of the training set by lowering the impact of windows where just a few active samples have very high signature activity, and favoring windows where a large number of active samples have moderately high activity. The improved performance of $\alpha = 0.5$ over $\alpha = 1$ suggests the necessity and effectiveness of this parameterization.

As an aside, it is worth noting that the MSK-IMPACT panel performs worse than the random baseline on all examined signatures except Signatures 2 and 30. The MSK-IMPACT panel was designed for the detection of common driver mutations, so it follows reasonably that the mutations it captures have a different distribution than the distribution of mutations over the whole genome (and consequently, the signatures obtained from its captured mutations may be inconsistent with genome-wide signatures). We additionally note that the random panel benchmark is in actuality given by the mean performance of 1000 panels with randomly selected windows (see Methods) — thus while it is a useful point of comparison, it does not represent a clinically actionable assay, as the performance of any *individual* random panel is highly variable.

Finally, to assess the generalizability of the panels designed by ScalpelSig, we evaluated its performance on a completely held-out dataset. To identify a single panel per signature, we trained ScalpelSig on the whole cohort of 560 breast cancer genomes described above [137], and evaluated the resulting panels using a new cohort of 237 breast cancer genomes as a test set from Staaf et al. [138]. We found that ScalpelSig continued to outperform the MSK-IMPACT panel in this setting, both in terms of Spearman correlation and AUPR (Table 4.4). This provides preliminary evidence that the improved performance of ScalpelSig panels generalizes beyond our initial dataset.

4.4.1.1 Performance at smaller panel sizes

We investigated how performance of designed panels changed at different panel sizes, moving beyond our initial focus on the 2.5 Mb panel size used by MSK-IMPACT. One would naturally expect that the more genomic material is sampled, the better performance should become, since the number of observed mutations approaches the whole genome mutation count as the panel gets larger. This sensible intuition holds in most cases, but there are a few exceptions worth discussing. First, we note that panel performance is *not* a monotonically increasing function of panel size (Fig. 4.2). This

	Spearman correlation		AUPR	
Sig.	ScalpelSig	MSK-IMPACT	ScalpelSig	MSK-IMPACT
2	***0.3437***	0.1804	***0.3844***	0.2900
3	***0.5048***	0.4749	***0.9321***	0.9137
8	***0.4275***	0.2004	***0.9406***	0.9195
13	***0.5365***	0.3276	***0.7318***	0.6837
18	—	—	***0.1458***	0.0298

Table 4.4: Evaluation of ScalpelSig ($\alpha = 0.5$) and MSK-IMPACT on a completely held-out dataset of 237 breast cancer genomes from Staaf et al. [138]. Each row reports the results of a single ScalpelSig panel, trained on all 560 samples from the previous dataset. Spearman correlations with P-value ≥ 0.05 are not shown. The highest values in each row for each of the two evaluation metrics (Spearman and AUPR) are bolded and italicized.

phenomenon is unintuitive, but in fact observing additional genome regions is not guaranteed to increase accuracy, and may even decrease it. To see this, it is important to understand that individual genome windows may have mutation distributions which differ starkly from the genome-wide distribution, due to variation in nucleotide composition and other factors. Adding such windows to a panel could produce a mutation distribution that misrepresents the genome-wide distribution, despite containing a greater number of mutations. This misleading distribution would result in a false impression of signature activity. Thus it is reasonably possible for performance to decline with the addition of certain windows.

We further observe that panels constructed with $\alpha = 0.5$ for Signatures 2, 13, and 18 all appear to plateau in their performance before the 2.5 Mb panel size is reached — this plateau seems to occur at 1.5 Mb for Signatures 2 and 13, and at 2.0 Mb for Signature 18. This indicates that in some cases, panels significantly smaller than those presently in clincial use may be sufficient to achieve the full performance boost provided by our framework. If one recalls that the panel is formed from the highest-scoring windows, this behavior makes sense: as the panel gets bigger, progressively lower-scoring windows are added, so it follows that the performance increase might plateau.

4.5 Discussion

While a growing body of literature attests to the efficacy of mutational signature activity as a predictive biomarker for targeted cancer therapies, the sequencing demands of almost all methods for mutational signature analysis are not met by current clinical infrastructure. In this study we seek to address this problem with ScalpelSig, an algorithm that designs genomic panels optimized for the detection of mutational signature activity. ScalpelSig takes as input a mutational signature and a set of training tumor genomes, and learns genome regions wherein mutations are highly indicative of signature activity.

In the present study, we train ScalpelSig on breast cancer data to obtain panels optimized for the detection of six respective mutational signatures. We find that in five out of six examined signatures, ScalpelSig panels outperform the commonly studied MSK-IMPACT panel and a random baseline. The increased accuracy of our panels is substantial even when compared to whole exome sequencing (see Table 4.3), which uses over $10\times$ as much genomic material as targeted panel sequencing. We show that ScalpelSig panels bridge significant portions of the performance disparity between the baseline panels and the whole exome benchmark. In the most favorable case, this performance increase is roughly proportional to the addition of 10.5 Mb of exonic material to the baseline panels. We find that the performance of ScalpelSig is robust under a variety of parameterizations. For four of six examined signatures, panels smaller than 2.5 Mb are sufficient to obtain all or most of the demonstrated performance increase. Further, our panels maintain strong performance even when the amount of training data is significantly reduced. These results suggest that the performance increase afforded by our method may generalize beyond the conditions of our study – e.g., to other cancer types that have less training data available.

Of the signatures investigated in this paper, Signature 3 is arguably the most exciting as a clinically actionable biomarker. Signature 3 has been found to be correlated with biomarkers of homologous recombination repair deficiency [120, 126, 129, 137], which is a biomarker for PARP inhibitor therapy [112]. Recently, Gulhan et al. [126] developed an algorithm for improved detection of Signature 3 from MSK-IMPACT panel data. Our results demonstrate that ScalpelSig panels confer a substantial increase in accuracy for detection of Signature 3 compared to MSK-IMPACT when using standard methods for signature detection. Concretely, relative to MSK-IMPACT, ScalpelSig panels give an 83% increase in Spearman correlation coefficient between Signature 3 activity in panel regions and genome-wide Signature 3 activity. We note however that when the panels were evaluated on a held-out dataset, the improvement for Signature 3 was much more modest, indicating that the impact of distributional changes across data sets merits further investigation. Even still, this begs the question of whether the algorithm used in Gulhan et al. [126] could be trained using ScalpelSig panel data to achieve a synergistic boost in accuracy for detection of Signature 3. If so, combining the two approaches could be a boon for detecting HR deficiency in the clinic. We point to this idea as an important direction for future work.

For other future work, we plan to investigate other methods for inferring signature exposures and active signatures, including Huang et al.'s method for automatically detecting which signatures are active in each sample with confidence [139]. While we show that ScalpelSig's good performance is stable on a variety of parameterizations, additional work towards finding optimal parameterizations (e.g., hyperparameter tuning via cross-validation) may be beneficial. We leave this fine-grained optimization for future work. In terms of exploring clinical applications of ScalpelSig, one obvious avenue would be to train the algorithm for other cancer types and signatures, and validate it with completely held out datasets. Ultimately, we anticipate that such a validated genomic panel for detecting multiple signatures across multiple cancer types will be the most straightforward path to clinical impact. To that point, we performed an exploratory analysis that points to the challenge of creating a multiple signature panel. Further improvement of the accuracy of such a panel will likely require a more sophisticated algorithm – one that can score genome regions for how well they assay the activity of multiple signatures simultaneously

Chapter 5: Perplexity: evaluating RNA-seq expression estimation in the absence of ground-truth

Disclosure

This Chapter presents first-author work as published in [16], with minimal changes.

5.1 Background and motivation

Due to its accuracy, reproducibility, simplicity and low cost, RNA-seq has become one of the most popular high-throughput sequencing assays in contemporary use, and it has become the *de facto* method for the profiling of gene and transcript expression in many different biological systems. While there are many uses for RNA-seq that span the gamut from *de novo* transcriptome assembly [142, 143] through meta-transcriptome profiling [144], one of the most common uses is to interrogate the gene or isoform-level expression of known (or newly-assembled) transcripts, often with the subsequent goal of performing a differential analysis between conditions of interest.

Because of the popularity of gene and transcript expression profiling using RNA-seq, considerable effort has been expended in developing accurate, robust and efficient computational methods for inferring transcript abundance estimates from RNA-seq data. Some popular approaches focus on counting the aligned RNA-seq reads that overlap genes in different ways [145, 146]. However, these approaches have no principled way to deal with reads that align well to multiple loci (e.g., to different isoforms of a gene, or between sequence-similar regions of related genes), and this restricts their use primarily to gene-level analysis, where they may still under-perform more sophisticated approaches that attempt to resolve fragments of ambiguous origin [147].

Alternatively, many approaches offer the ability to estimate transcript-level expression using RNA-seq data (which can, if later desired by a user, be aggregated to the gene-level). The majority of these approaches perform statistical inference over a probabilistic generative model of the experiment based either on sufficient statistics of counts [148, 149] or the set of fragment alignments themselves [150]. Moreover, in addition to methods focused on deriving point estimates for transcript abundances, there has been considerable development of probabilistic Bayesian approaches for this inference problem [20, 151, 152, 153, 154, 155], as well as recent attempts at multi-sample probabilistic models for simultaneous experiment-wide transcript abundance estimation [156, 157]. Bayesian approaches can sometimes offer more accurate or robust inference than methods based strictly on maximum likelihood estimation, but these Bayesian models invariably expose prior distributions, with associated hyperparameters, upon which the resulting inferences depend.

Interestingly, the recommended best practices suggested by the different Bayesian (or variational Bayesian) approaches for selecting hyperparameters differ. Specifically, Nariai et al. [153] evaluate performance varying the prior used in their variational Bayesian expectation maximization (VBEM)-based method, and they conclude that a small prior (i.e., $\alpha < 1$) leads to a sparse solution, which, in turn, results in improved accuracy. On the other hand, Hensman et al. [152] perform inference using a prior of $\alpha = 1$ read per transcript. They find that, doing so, their method produces the most *robust* estimates (i.e., with the highest concordance between related replicates) that are also more accurate under different metrics that they measure. Their conclusion is that methods adopting a maximum likelihood model inferred using an expectation maximization procedure tend to produce sparse estimates close to the boundary of the parameter space which leads to less robust estimation among related samples. Unfortunately, regardless of how prior studies have argued for a "better" prior, none provide an empirical or practical procedure for model selection. Rather, they show that a value works well across a range of data under some evaluation metric, and set this as the default value for all inference tasks. Given the number of existing methods that can make use of prior information (including methods like those by Srivastava et al. [158] for single-cell data, or those by Liu et al. [159] that use orthogonal modalities of data to set priors), it becomes increasingly important to develop methods that lets one robustly and automatically select an appropriate prior (hyperparameter) for these algorithms.

To perform model (or hyperparameter) selection for transcript abundance estimators, one must be able to evaluate estimated abundances. However, evaluation of abundance estimates remains a challenge for current methods on experimental data where ground truth is completely absent. Notably, evaluation of transcript abundance estimators on experimental data have relied on careful experiment design that enables comparisons to complementary assays (e.g., correlation with qPCR) or measurements (e.g., concordance with known mixing proportions or spike-ins) [160]. Such evaluation procedures vary from study-to-study, and are simply not possible when complementary experiments are not designed or available. Thus, the natural question is then: *can the quality of transcript abundance estimates be meaningfully evaluated on the set of given fragments directly?*

It may initially be unintuitive to think that the "goodness" of a transcript abundance estimate can be evaluated in the absence of ground truth. However, in a related line of research, likelihood-based metrics for assessing the quality of *de novo* assemblies, where ground truth is unavailable, have been explored. For example, Rahman and Pachter [161] developed a method to compute the likelihoods of assembled genomes; Li et al. [162] developed a likelihood-based score to evaluate transcriptome assemblies; Smith-Unna et al. [163] developed a method to assess the quality of assembled contigs in transcriptomes; and Clark et al. [164] developed a method that is applicable to both genome and metagenomic assemblies. Furthermore, if we look to other unsupervised problem settings where ground truth annotations are absent, metrics for measuring the "goodness" of estimated models with latent parameters not only exist, but are regularly used. For example, metrics such as the silhouette score used to evaluate clustering algorithms come to mind [165]. In fact, evaluation of unsupervised probabilistic models, especially language and topic models in natural language processing, is commonplace [166, 167]. Specifically, *perplexity*, the inverse geometric mean per-*word* likelihood of a held-out test set, has been ubiquitously used to compare models [166].

In this work, we derive perplexity for transcript abundance estimation with respect to held-out per-*read* likelihoods. As we shall see, the perplexity of a held-out fragment set given an abundance estimate, computed via a quantify-then-validate approach, is a theoretically and experimentally motivated measure of the quality of the given estimate. Notably, perplexity quantifies an important biologically motivated intuition — that a good abundance estimate ought to generalize and generate the validation set, which is, in a sense, a form of a technical replicate, with high probability.

Perplexity can be used wherever the assessment of the quality of abundance estimates is desired. For example, perplexity can be used to compare different transcript abundance estimation algorithms or, as suggested above, to perform model selection to obtain the most accurate estimates from a given algorithm. In this work, we focus on experimentally assessing perplexity with respect to the latter, model selection for the prior used to estimate abundances with `salmon` [20]. In `salmon`, the reads-per-transcript prior size is a hyperparameter that controls its preference for inferring sparse or smooth abundance estimates. Notably, the problem of model selection offers a succinct assessment and immediately useful application of how perplexity can be computed to evaluate and compare the quality of candidate transcript abundance estimates.

5.1.1 Contributions

Theoretically, we derive and motivate a notion of *perplexity* for transcript abundance estimation – a metric for evaluating inferred estimates in the absence of ground truth. Experimentally, we demonstrate that perplexity for transcript abundance estimates is well behaved, and establish empirical correspondence between perplexity and other metrics that are more commonly used to demonstrate the “goodness” of transcript abundance estimates.

We summarize our experimental contributions below:

1. In experimental data from the Sequencing Quality Control (SEQC) consortium [160], we show that transcript abundance estimates with the lowest perplexity (lower is better) achieve the highest correlation with complementary qPCR measurements of biological replicates.

2. In simulated data, perplexity is concordant with respect to three measurements against ground truth: Spearman correlation with respect to expressed transcripts, AUROC with respect to unexpressed transcripts, and downstream differential transcript expression analysis.

3. In a proof-of-concept style experiment, we demonstrate that perplexity can be computed for *almost any* transcript abundance estimation model.

Evidenced by these results, we propose perplexity as the first and, to our knowledge, only theoretically and experimentally justified metric for model selection for transcript abundance estimation in *experimental* data where ground truth is entirely absent.

5.2 Preliminaries: (approximate) likelihood for transcript abundance estimation

Before deriving *perplexity* for transcript abundance estimation, we shall briefly recall and define the necessary objects that pertain to the *likelihood* of the probabilistic model that underpins transcript abundance estimation (as in [20, 150]).

The transcript abundance estimation problem, or quantification, from short RNA-seq *fragments* (a term used to refer, generically, to either single reads or read pairs), is the problem of assigning each fragment f_j of an input fragment-set $\mathcal{F} = \{f_1, \ldots f_N\}$ to its transcript of origin. For this work, we shall only consider quantification with respect to a given reference transcriptome whereby a quantifier maps each input fragment f_j to a transcript in an input set of reference transcripts $\mathcal{T} = \{t_1, .., t_M\}$.

Given the sequence of an input fragment, said fragment may align to more than one transcript, t_i, in the reference transcriptome $\mathcal{T}$. Here, the *de facto* method for determining transcript of origin for fragments that multi-map to more than one transcript is to view the true fragment to transcript assignment as a latent variable, and to infer the latent variable's expected value by performing inference in the underlying probabilistic model.

Assuming an appropriate normalization of alignment scores, we write the probability of observing a fragment, f_j, given that it originates from (or aligns to) transcript t_i to be $\Pr(f_j \mid t_i)$. The probability that a molecule in a sample that is selected for sequencing is the transcript t_i is then $\Pr(t_i \mid \boldsymbol{\theta})$, a multinomial over $\mathcal{T}$. Marginalizing over all possible alignments, the *likelihood* of observing the fragment set $\boldsymbol{F}$ given model parameters $\boldsymbol{\theta}$ is,

$$\Pr(\boldsymbol{F} \mid \boldsymbol{\theta}) = \prod_{j}^{N} \sum_{i}^{M} \Pr(t_i \mid \boldsymbol{\theta}) \cdot \Pr(f_j \mid t_i). \tag{5.1}$$

In this work, we shall work with the *range-factorized* equivalence class approximation of the likelihood that has proven to be effective and is efficient to compute [168]. Here, sets of fragments in $\boldsymbol{F}$ that map to the same set of transcripts, and have similar conditional probabilities of arising from these transcripts, are said to belong to the equivalence class $\boldsymbol{F}^q$ (indexed by q). Instead of working with alignment probabilities $\Pr(f_j \mid t_i)$ of each fragment, fragments in an equivalence class $\boldsymbol{F}^q$ are approximated to have the same conditional probability $\Pr(f_j \mid \boldsymbol{F}^q, t_i)$ for mapping to each transcript t_i. Let $\mathcal{C}$ be the set of equivalence classes induced by $\boldsymbol{F}$ and $\Omega(\boldsymbol{F}^q)$ be the set of transcripts to which $f \in \boldsymbol{F}_q$ map. The range-factorized equivalence class approximation of the likelihood $\Pr(\boldsymbol{F} \mid \boldsymbol{\theta})$ is,

$$\Pr(\boldsymbol{F} \mid \boldsymbol{\theta}) \approx \prod_{\boldsymbol{F}^q \in \boldsymbol{C}} \left(\sum_{t_i \in \Omega(\boldsymbol{F}^q)} \Pr(t_i \mid \boldsymbol{\theta}) \cdot \Pr(f_j \mid \boldsymbol{F}^q, t_i) \right)^{N^q}. \tag{5.2}$$

Here, the approximate likelihood can be computed over the number of unique equivalence classes, which is considerably smaller than the number of all possible alignments for all fragments.

5.3 Methods: deriving perplexity for RNA-seq

We propose a subtle but instructive change in the usual computational protocol for evaluating transcript abundance estimates. We propose a *quantify-then-validate* approach which evaluates the quality of transcript abundance estimates directly on read-sets, analogous to *train-then-test* approaches for evaluating probabilistic predictors common in natural language processing (NLP) and other fields [169, Ch. 1.3]. Instead of quantifying all available fragments and then performing evaluation with respect to complementary measurements downstream, the quantify-then-validate approach validates and evaluates the quality of a given abundance estimate directly on a set of held-out *validation* fragments withheld from inference.

We derive and adapt from NLP, the notion of *perplexity* for transcript abundance estimation for this quantify-then-validate approach [166, 167]. Perplexity is computed given only an abundance estimate, and a held-out validation set of fragments as input. Thus, perplexity evaluates the quality of abundance estimates on fragments directly and can evaluate estimates from experimental data in the absence of ground truth. Most importantly, evaluating perplexity with the quantify-then-validate approach enables quantitative, evidence-based, cross-validated selection of hyperparameters for transcript abundance estimation methods that use them.

Perplexity for transcript abundance estimation quantifies the intuition that an abundance estimate for a given sample ought, with high probability, explain and generate the set of fragments of a technical replicate. The key observation is that the likelihood $\Pr(\boldsymbol{F} \mid \boldsymbol{\theta})$ is simply a value that can be computed for any fragment set $\boldsymbol{F}$ and any abundance estimate $\boldsymbol{\theta}$ (model parameters), irrespective of whether $\boldsymbol{\theta}$ is inferred from $\boldsymbol{F}$. It is the context and application of the likelihood, $\Pr(\boldsymbol{F} \mid \boldsymbol{\theta})$, that yield semantic meaning.

Given a fragment set, $\mathcal{F}$, over which one seeks to infer and evaluate abundance estimates, the quantify-then-validate procedure is as follows. First, partition the input set into a *quantified* set, $\boldsymbol{F}$, and a *validation* set, $\widehat{\boldsymbol{F}}$. Second, *quantify* and infer abundance estimates (model parameters) $\boldsymbol{\theta}$ given the quantified set $\boldsymbol{F}$. Third, *validate* and compute the perplexity, $PP(\widehat{\boldsymbol{F}}, \boldsymbol{\theta})$ — the inverse geometric mean held-out per-read likelihood of observing the validation set, $\widehat{\boldsymbol{F}}$ — given model parameters $\boldsymbol{\theta}$ and the validation set $\widehat{\boldsymbol{F}}$. The lower the perplexity, the better the parameters $\boldsymbol{\theta}$ describe the held-out fragments $\widehat{\boldsymbol{F}}$, and the better the abundance estimate parameterized by $\boldsymbol{\theta}$ ought to be. In fact, if we believe that the generative model is truly descriptive of the distributions that arise from the underlying biological and technical phenomena, perplexity is, in expectation, minimized when the "true" latent parameters are inferred.

Formally, given an abundance estimate $\boldsymbol{\theta}$, and a validation fragment-set $\widehat{\boldsymbol{F}} = \{\hat{f}_1, \ldots, \hat{f}_{\widehat{N}}\}$, the perplexity for transcript abundance estimation is:

$$\begin{aligned} PP(\widehat{\boldsymbol{F}}, \boldsymbol{\theta}) &= \exp\left\{-\frac{1}{\widehat{N}} \log \Pr(\widehat{\boldsymbol{F}} \mid \boldsymbol{\theta})\right\} \\ &= \exp\left\{-\frac{1}{\widehat{N}} \sum_{j=1}^{\widehat{N}} \log \Pr(\hat{f}_j \mid \boldsymbol{\theta})\right\}, \end{aligned} \tag{5.3}$$

with per-fragment likelihood,

$$\Pr(\hat{f}_i \mid \boldsymbol{\theta}) = \sum_{i=1}^{M} \Pr(t_i \mid \boldsymbol{\theta}) \cdot \Pr(\hat{f}_j \mid t_i). \tag{5.4}$$

Crucially, the probability $\Pr(\hat{f}_j \mid \boldsymbol{\theta})$ of observing each held out fragment given $\boldsymbol{\theta}$ is computed and marginalized over the product of two terms, $\Pr(\hat{f}_j \mid t_i)$ that depends only on the validation set of held-out fragments, and $\Pr(t_i \mid \boldsymbol{\theta})$ that depends only on the given abundance estimate.

One particular application of the perplexity metric, which we explore here, is to select the best abundance estimate out of many candidate estimates arising from different hyperparameter settings for quantifiers. Thus, in this work, we use the range-factorized equivalence class approximation for perplexity (as in (5.2)) throughout [168]. Given the range-factorized equivalence classes, $\widehat{\boldsymbol{C}}$, induced by the *validation* set, $\widehat{\boldsymbol{F}}$, (where $\widehat{N^q}$ is the number of fragments in an equivalence class $\widehat{\boldsymbol{F}^q} \in \widehat{\boldsymbol{C}}$) the approximation is:

$$PP(\widehat{\boldsymbol{F}}, \boldsymbol{\theta}) \approx \exp\left\{-\frac{1}{\widehat{N}} \sum_{\widehat{\boldsymbol{F}}^q \in \widehat{\boldsymbol{C}}} \widehat{N}^q \cdot \log \Pr(\hat{f}_i \mid \widehat{\boldsymbol{F}}^q, \boldsymbol{\theta})\right\}, \tag{5.5}$$

with approximate per-fragment likelihood,

$$\Pr(\hat{f}_i \mid \widehat{\boldsymbol{F}}^q, \boldsymbol{\theta}) = \sum_{t_i \in \Omega(\widehat{\boldsymbol{F}}^q)} \Pr(t_i \mid \boldsymbol{\theta}) \cdot \Pr(\hat{f}_j \mid \widehat{\boldsymbol{F}}^q, t_i). \tag{5.6}$$

We use `salmon`'s selective-alignment based probabilistic model for conditional probabilities $\Pr(\hat{f}_j \mid \widehat{\boldsymbol{F}}^q, t_i)$ and effective lengths of transcripts, since the model and equivalence class approximation `salmon` uses has proven to be a fast and effective way to approximate the full likelihood [157, 168]. For the scope of this work, `salmon`'s format for storing range-factorized equivalence classes conveniently contains all relevant information and values to compute perplexity with vastly smaller space requirements than would be required to store per-fragment alignment probabilities $\Pr(\hat{f}_j \mid t_i)$.

5.3.1 "Impossible" fragments given parameter estimates.

We now address a perplexity-related issue that is unique to evaluating transcript abundance estimates — that an observed event in the validation set may be deemed "impossible" given model parameters $\boldsymbol{\theta}$. The marginal probability, $\Pr(\hat{f}_j \mid \boldsymbol{\theta})$, for observing a fragment $\hat{f}_j$ in the validation set given some abundance estimate, $\boldsymbol{\theta}$, may actually be zero, even if said validation fragment aligns to the reference transcriptome. This occurs exactly when all transcripts, t_i, to which the validation fragment $\hat{f}_j$ map are deemed unexpressed by $\boldsymbol{\theta}$ (i.e. $\Pr(t_i \mid \boldsymbol{\theta}) = 0$ for all such transcripts). Here, we say that $\hat{f}_j$ is an *impossible fragment* given $\boldsymbol{\theta}$, and that $\boldsymbol{\theta}$ *calls* $\hat{f}_j$ impossible. When impossible fragments are observed in the validation set, perplexity is not a meaningful measurement.

To illustrate how impossible fragments come to be, consider the toy example in which all fragments in a quantified set that align to transcripts A, B, or C only ambiguously map to $\{A, B\}$, or to $\{A, C\}$. That is, no such fragments uniquely map — a phenomenon observed rather frequently for groups of similar isoforms expressed at low to moderate levels. Now, suppose that an abundance estimation model assigns all such fragments to transcript A and produces an estimate $\boldsymbol{\theta}$. The quantifier may be satisfying a prior that prefers sparsity; or prefers to do so because transcript A is considerably shorter than transcripts B and C, which gives it a higher conditional probability under a length normalized model. In this case, the marginal probability, $\Pr(\hat{f}_j \mid \boldsymbol{\theta})$, of observing a validation fragment $\hat{f}_j$ that maps to $\{B, C\}$ is exactly zero given the parameters $\boldsymbol{\theta}$.

As an example, we randomly withhold varying percentages of fragments from one sample (`SRR1265495`) as validation sets and use all remaining fragments to estimate transcript abundances with `salmon`'s default model (i.e. the VBEM model using prior size of 0.01 reads-per-transcript). Fig. 5.1 shows that at all partitioned percentages, impossible fragments in the validation set are prevalent with respect to estimated abundances. In fact, due to the prevalence of impossible reads, perplexity as written in (5.5) is undefined (or infinite) for all estimates and all validation sets in the experiments below. An important observation in both the toy and experimental examples is that there likely exist better abundance estimates that would call fewer fragments impossible, while still assigning high likelihood to the rest of the (possible) fragments. For example, an abundance estimate that reserves even some small probability mass to transcript B in the toy example would not call the validation fragments in question impossible.

5.3.2 Why perplexities need to be smoothed

The problem with impossible fragments is not only that they exist. The problem is that, for a fixed validation fragment set, perplexity deems an abundance estimate that calls even one fragment impossible equally as bad as an abundance estimate that calls all fragments impossible. Here, both estimates would have unbounded perplexity since the validation set has zero likelihood given each estimate. However, the former ought be preferred over the latter.

Other fields that have adopted and used perplexity (e.g. natural language processing) usually sidestep the issue of impossible events entirely both by construction and pre-processing, working only with smoothed probabilistic models in which no event has probability zero, or removing rare words from input language corpora. However, neither strategy is available nor appropriate for evaluating transcript abundance estimates. It is neither reasonable nor useful to amend and modify each of the many modern quantifiers to produce smooth outputs (outputs in which no transcript has truly zero abundance), and fragments and transcripts cannot be pre-processed away since the set of expressed transcripts cannot be identified *a priori*. One may also be tempted to simply remove impossible fragments from a validation set, $\widehat{\boldsymbol{F}}$, before computing a perplexity or hold out fragments — but this also is not a valid strategy. This is because two different abundance estimates $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$ may call different validation fragments in $\widehat{\boldsymbol{F}}$ impossible, and comparisons of likelihoods $\Pr(\widehat{\boldsymbol{F}}' \mid \boldsymbol{\theta}')$ and $\Pr(\widehat{\boldsymbol{F}} \mid \boldsymbol{\theta})$ are only meaningful if the validation sets are the same (i.e. $\widehat{\boldsymbol{F}} = \widehat{\boldsymbol{F}}'$). Furthermore, there is no straightforward strategy to sample and hold-out validation fragments so that no fragments are impossible. This is because most validation fragments cannot be determined to be impossible prior to abundance estimation, and any non-uniform sampling strategy would alter the underlying distributions that estimators aim to infer.

To compare estimates that may call different validation fragments impossible, the proposed perplexity metric (as in (5.5)) must be *smoothed*. Strategies that smooth perplexities ought *penalize* estimates that call fragments impossible. That is, impossible fragments under such smoothing strategies ought result in a penalty and overcome the shrinkage of $\Pr(\widehat{\boldsymbol{F}} \mid \boldsymbol{\theta})$ to zero. Below, we detail two such smoothing strategies for computing perplexities: (a) *Laplacian smoothed* perplexity and (b) *Good-Turing smoothed* perplexity.

We schematically illustrate how a smoothed perplexity measure, using the proposed quantify-then-validate protocol, can be computed to evaluate the quality of transcript abundance estimates in Fig. 5.2.

5.3.3 Laplacian smoothed perplexity

We define *Laplacian smoothed* perplexity given abundance estimate $\boldsymbol{\theta}$ to be the perplexity evaluated with the smoothed distribution $\widetilde{\Pr}_\beta(t_i \mid \boldsymbol{\theta})$ in place of $\Pr(t_i \mid \boldsymbol{\theta})$. The Laplacian smoothing scheme smooths input abundance estimates by redistributing a small constant probability mass across the reference transcriptome. Let $\Pr(t_i \mid \boldsymbol{\theta}) = \eta_i$ and M be the number of transcripts in the reference. The smoothed distribution parameterized by β is defined to be:

$$\widetilde{\Pr}_\beta(t_i \mid \boldsymbol{\theta}) = \frac{\eta_i + \beta}{1 + M\beta}. \tag{5.7}$$

Laplacian smoothed perplexity is flexible and easy to implement but requires the user to *set* a value (preferably small e.g. 1×10^{-8}) for the smoothing parameter β.[1] At the cost of not being

[1] This is equivalent to adding, for each transcript t_i in the reference, $\beta \cdot \sum_j^M c_j/\tilde{\ell}_j$ reads-per-nucleotide to the expected fragments per-transcript counts c_i then re-normalizing to obtain TPMs, given effective transcript lengths $\tilde{\ell}_i$ (as defined in `salmon` [20]).

parameter-free, Laplacian smoothed perplexity allows the user to tune the degree to which impossible reads are penalized. The smaller the value of β, the smaller larger the penalty an estimate incurs for each validation fragment it calls impossible

5.3.4 Good-Turing smoothed perplexity — adaptive and parameter-free

The major drawback of Laplacian smoothed perplexity is that it depends on a reasonable *a priori* selection of a value for the smoothing parameter β. One further concern is that Laplacian smoothed perplexity is not adaptive and does not account for the amount of evidence from which an input estimate is derived — i.e. the read-depth or the number of quantified reads in a sample. For a fixed value of β, the Laplacian smoothed perplexity smooths probabilities inferred from a million fragments equally as much as probabilities inferred from a trillion fragments. However, for the latter estimate that is inferred from much more data, it is more sensible to smooth and redistribute less probability mass.

For example while varying one of `salmon`'s hyperparameters, Laplacian smoothed perplexities suggest the existence of a locally optimal behavior when computed with a wide range of values for β (see Fig. 5.16). However, the locally optimal behavior can no longer be observed if Laplacian smoothed perplexities are computed with $\beta = 1 \times 10^{-6}$.

A better, adaptive, smoothing strategy would directly estimate the probability of observing fragments from transcripts that are not expressed. Abstractly, the problem to be solved is to estimate the probabilities of observing unobserved events. Here, we turn to the Simple Good-Turing (SGT) method [170] that has been applied in a wide range of areas, including estimating the probabilities of unseen sequences in computational linguistics [170], as well as for the detection of empty droplets in droplet-based single-cell RNA sequencing protocols [171].

Below, we define the *Good-Turing smoothed perplexity* measure, where smoothed probabilities are derived from SGT smoothed fragment per-transcript counts.

Given frequencies over a population — i.e. the number of reads originating from each trancsript — the SGT method estimates:

1. the *total* probability mass that ought be assigned to unseen events — the "expression" of unexpressed transcripts, and
2. the appropriate adjustments for probabilities of observed events — the adjusted probabilities for expressed transcripts.

It is not immediately obvious how to implement SGT smoothing for the purpose of smoothing transcript abundance estimates. One issue is that the SGT estimator expects as input, integer valued frequencies of observed events, while input abundance estimates for computing perplexity are real-valued estimated frequencies of per-transcript counts. For the purposes of smoothing and computing perplexity, we round the estimated number of fragments per-transcript, c_i, to the nearest integer and treat these as raw frequencies of events for SGT smoothing.

The SGT method also requires that input frequencies-of-frequencies (i.e. the number of transcripts that have the same fragments per-transcript) to be log-linear. Empirically, we show in Fig. 5.3 that rounded input abundance estimates do, indeed, follow a log-linear distribution. The confirmed log-linear relationship demonstrates the rounding step to be a reasonable approximation.

The SGT method estimates the adjusted frequencies $r^\star$ for each event observed r times. These adjusted frequencies are then used to compute per-event (or per-transcript) probabilities. Let c_i be the rounded number of fragments per-transcript t_i. Let the frequency of frequencies $n_r = |\{t_i \mid c_i = r\}|$. And let there be **n** total reads. The SGT method computes and outputs,

1. the adjusted frequencies, $r^\star = (r+1)\frac{S(n_{(r+1)})}{S(n_r)}$;

2. and the total probability, $P_0 = \frac{n_1}{\mathbf{n}}$, for observing any transcript with $c_i = 0$.

Here, $S(n_r)$ computes a smoothed frequency of frequencies. Frequencies of frequencies n_r have to be smoothed because n_r for many large r are zero in observed data. The precise details for computing the smoothed $S(n_r)$ are described in [170]. In brief, SGT smooths n_r by fitting a fitted log-linear function on r against n_r and reading off values of n_r for "large" r.

Good-Turing smoothed perplexity is perplexity computed with the smoothed per-transcript distribution $\widetilde{\Pr}(t_i \mid \boldsymbol{\theta})$ in place of $\Pr(t_i \mid \boldsymbol{\theta})$. Here, the smoothed per-transcript distribution is derived from adjusted frequencies $r^\star$ and P_0.

For each "expressed" transcript t_i with count, $c_i = r$, greater than zero, the SGT smoothed probability is proportional to the transcript's adjusted frequency normalized by its effective length, with $\widetilde{\Pr}(t_i \mid \boldsymbol{\theta}) \propto r^\star/\tilde{\ell}_i$. The smoothed probabilities for expressed transcripts are normalized so that they sum to $(1 - P_0)$.

For each "unexpressed" transcripts with $c_i = 0$, the SGT smoothed probability is proportional to the transcript's effective length, and is derived from distributing the probability mass P_0 uniformly over the effective lengths of all unexpressed transcripts in the reference. Here, the smoothed per-transcript distribution is defined $\widetilde{\Pr}(t_i \mid \boldsymbol{\theta}) \propto P_0 \ell_i$. The smoothed probabilities for unexpressed transcripts are normalized so that they sum to P_0.

For all following sections we shall use perplexity to mean Good-Turing smoothed perplexity unless stated otherwise.

5.3.5 Model selection using perplexity in practice

Arguably, one of the most useful outcomes of being able to evaluate the quality of abundance estimates in the absence of ground truth is the ability to perform model selection for transcript abundance estimation in experimental data. For those familiar with train-then-test experimental protocols for model selection in machine learning or NLP, model selection for transcript abundance estimation *vis-a-vis* our proposed quantify-then-validate approach is analogous and identical in abstraction. However, since, to our knowledge, this work is the first to propose a quantify-then-validate approach for transcript abundance estimation, we shall briefly detail how perplexity ought to be used in practice.

Let us consider model selection via 5-fold cross-validation using perplexity given some fragment set $\mathcal{F}$. First, $\mathcal{F}$ is randomly partitioned into five equal sized, mutually exclusive validation sets, $\{\widehat{\boldsymbol{F}}_1, \dots, \widehat{\boldsymbol{F}}_5\}$ — and quantified sets are subsequently defined, $\boldsymbol{F}_i = \mathcal{F} - \widehat{\boldsymbol{F}}_i$. Now, suppose we desire to choose between L model configurations (e.g. from L hyperparameter settings). Then for each ℓ-th candidate model, we produce a transcript abundance estimate from each i-th quantified set, $\boldsymbol{\theta}_i^{(\ell)}$. To select the best out of the L candidate models, one simply selects the model that minimizes the average perplexity over the five folds, $\frac{1}{5}\sum_i \overline{PP}(\widehat{\boldsymbol{F}}_i, \boldsymbol{\theta}_i^{(\ell)})$.

One additional practical consideration should also be noted. Given *any* pair of quantification and validation sets $\boldsymbol{F}$ and $\widehat{\boldsymbol{F}}$, a validation fragment, $\hat{f}_j \in \widehat{\boldsymbol{F}}$, can be *necessarily impossible*. A necessarily impossible validation fragment is one that maps to a set of transcripts to which no fragments in the quantified set $\boldsymbol{F}$ also map. Such a fragment will always be called impossible given any abundance estimate deriving from the quantified set $\boldsymbol{F}$, since no fragments in $\boldsymbol{F}$ provide any evidence that transcripts to which $\hat{f}_j$ map are expressed.

It is of limited meaning to evaluate estimates with respect to necessarily impossible fragments. For the purposes of this work, we shall consider the penalization of an abundance estimate only with respect to impossible fragments that are recoverable — in other words, fragments that could be assigned non-zero probability given a better abundance estimate inferable from $\boldsymbol{F}$. As such, we remove necessarily impossible validation fragments from $\widehat{\boldsymbol{F}}$, given $\boldsymbol{F}$, prior to computing perplexity whenever fragment sets are partitioned into validation and quantified fragment sets.

5.3.6 Data

5.3.6.1 Sequencing Quality Control (SEQC) project data

We downloaded Illumina HiSeq 2000 sequenced data consisting of 100+100 nucleotide paired-end reads from the Sequencing Quality Control (SEQC) project [160]. SEQC samples are labeled by four different conditions $\{A, B, C, D\}$, with condition A being Universal Human Reference RNA and B being Human Brain Reference RNA from the MAQC consortium [172], with additional spike-ins of synthetic RNA from the External RNA Control Consortium (ERCC) [173]. Conditions C and D are generated by mixing A and B in 3:1 and 1:3 ratios, respectively.

In this work, we analyze the first four replicates from each condition sequenced at the Beijing Genomics Institute (BGI) – one of three official SEQC sequencing centers. For each sample, we aggregate fragments sequenced by all lanes from the flowcell with the lexicographically smallest identifier.[2] Quantitative PCR (qPCR) data of technical replicates for each sample in each condition are downloaded via the `seqc` BioConductor package.

[2]Scripts to download and aggregate SEQC data are available at `github.com/thejasonfan/SEQC-data`

5.3.6.2 Simulated lung transcript expression data

We simulated read-sets based on 10 sequenced healthy lung samples, with Sequence Read Archive accession number `SRR1265{495-504}` [174]. Transcript abundance estimates inferred by Salmon using the `--useEM` flag for each sample are used as ground truth abundances for read simulation (expressed in transcripts per million (TPM) and expected read-per-transcript counts). Then, transcript abundances in samples `SRR1265{495-499}`, for 10% of transcripts expressed in at least one of the five samples, are artificially up or down regulated by a constant factor ($2.0\times$) to simulate differential transcript expression. We treat the resulting read-per-transcript counts as ground truth, and generate for each sample a fragments set of 100+100 nucleotide paired-end reads using Polyester at a uniform error rate of 0.001 with no sequence specific bias [175].

5.3.7 Evaluation and experiments

The purpose of the experiments in this work are twofold. First, to establish the relationship and correspondence between perplexity and commonly used measures of goodness or accuracy in transcript abundance estimation. And second, to demonstrate how model and hyperparameter selection can be performed using perplexity. In particular, we perform and evaluate hyperparameter selection for `salmon` with respect to the prior size in the variational Bayesian expectation maximization (VBEM) model used for inference [20]. The user-selected prior size for the VBEM model in `salmon` encodes the prior belief in the number of reads-per-transcript expected for any inferred abundance estimate. This hyperparameter controls `salmon`'s preference for inferring sparse or smooth estimates – the smaller the prior size, the sparser an estimate `salmon` will prefer. As discussed above, prior studies on Bayesian models have not necessarily agreed on how sparse or smooth a good estimate ought to be [152, 153] – the experiments in this work aim to provide a quantitative framework to settle this disagreement.

We perform all experiments according to the proposed quantify-then-validate procedure and report results with respect to various metrics over a 5-fold cross-validation protocol. We use the Ensembl human reference transcriptome GRCh37 (release 100) for all abundance estimation and analysis [176].

5.3.7.1 Evaluation versus parallel SEQC qPCR measurements

We analyze the relationship between perplexity and accurate abundance estimation in experimental data from the SEQC consortium. In SEQC data, we evaluate accuracy of abundances estimated by `salmon` by comparing estimates to qPCR gene expression data on biological replicates, a coarse proxy to ground truth. We evaluate the Spearman correlation between gene expressions of qPCR probed genes in SEQC replicates versus the corresponding abundance estimates. Gene expression from estimated transcript expression is aggregated transcript-to-gene annotations from `EnsDb.Hsapiens.v86` [177] using `txImport` [147]. From gene expression data, Ensembl genes are mapped to corresponding Entrez IDs via `biomaRt` [178], and 897 genes are found to have a corresponding qPCR measurement in downloaded SEQC data. Expressions for genes with repeated entries in SEQC qPCR data are averaged.

5.3.7.2 Evaluation versus ground truth on simulated data

In simulated data, since ground truth abundances are available, we compare estimated TPMs (computed by `salmon`) against ground truth TPMs under two metrics.

First, we consider the Spearman correlation with respect to known expressed transcripts (i.e., transcripts with non-zero expression in ground truth abundances). We choose to evaluate Spearman correlation with respect to ground truth non-zero TPMs because of the presence of many unexpressed transcripts in the ground truth, meaning a high number of values tied at rank zero. Here, small deviations from zeros can lead to large changes in rank, leading to non-trivial differences in the resulting Spearman correlation metric. We demonstrate this phenomenon with respect to the ground truth abundance of a simulated sample (`SRR1265495`) with a mean TPM of 5.98, in which 49% of transcripts are unexpressed (82,358 / 167,268). We report the change in Pearson correlation, R^2 score, and Spearman correlation of ground truth TPMs versus ground truth TPMs perturbed with normally distributed noise at varying standard deviations. As we can see from Fig. 5.4, even small perturbations cause non-trivial changes in Spearman rank correlation, while changes in Pearson correlation are entirely imperceptible. The Pearson correlation, however, suffers from the well known problem that, in long-tailed distributions spanning a large dynamic range, like those commonly observed for transcript abundances, the Pearson correlation is largely dominated by the most abundant transcripts.

Second, we complement measuring Spearman correlation of non-zero ground truth TPMs with reporting the area under receiver operating characteristic (AUROC) for recalling ground truth zeros based on estimated abundances. While the measurement of Spearman correlation on the truly expressed transcripts is robust to small changes in predicted abundance near zero, it fails to account for false positive predictions even if they are of non-trivial abundance. The complementary metric of the AUROC for recalling ground truth zeros complements that metric, since it is affected by false positive predictions.

5.3.7.3 Differential expression analysis on simulated data

We perform transcript level differential expression analysis and analyze the recall of known differentially expressed transcripts in simulated lung tissue data (see Section 5.3.6.2). We perform differential expression analysis at the trancript level using `swish` [179] using 20 inferential replicates from `salmon`. We modified `salmon` to ensure that prior sizes supplied via the `--vbPrior` flag are propagated to the Gibbs sampling algorithm. We plot receiver operating characteristic (ROC) curves and report the mean AUROC for predicting differentially expressed transcripts over multiple folds. We assign $P = 1$ to transcripts for which `swish` does not assign adjusted P-values.

5.3.7.4 Evaluation of `eXpress` abundance estimates

We measure the change in perplexities of abundance estimates inferred by `eXpress` (version 1.5.1) when running 0, 1, and 2 additional rounds of the online expectation maximization (EM) optimization step. We use the `--additional-online` parameter to specify the number of additional online EM steps. We provide to `eXpress` alignments to the human transcriptome computed by `bowtie2` [180] using the parameters recommended by `eXpress` with: `-a -X 600 --rdg 6,5 --rfg 6,5 --score-min L,-.6,-.4 --no-discordant --no-mixed`.

To compute perplexity, we use the transcript effective lengths computed by `eXpress` for each transcript inferred to be expressed. For each transcript inferred to be unexpressed, we use transcript lengths in place of effective lengths, since `eXpress` sets the effective length for these transcripts to zero. We take effective counts computed by `eXpress` to be expected fragment per-transcript counts.

5.4 Results

5.4.1 Low perplexity implies accurate abundance estimates in experimental SEQC data

In experimental data from the Sequencing Quality Control (SEQC) project [160], we demonstrate that perplexity can be used to perform parameter selection and select the `salmon` VBEM prior size that leads to the most accurate transcript abundance estimates. We note that perplexity plots for replicates are similar within conditions A-D, and thus include only plots for the first replicate in each condition in the main text. For completeness, plots for all samples are presented in Figs. 5.10 to 5.13.

Empirically, perplexity is well-behaved over all samples in the experimental data. As shown in Figs. 5.6 and 5.7, plots of perplexity against VBEM prior size and Spearman correlation against VBEM prior size both display an empirically convex shape minimized at the same VBEM prior size. This suggests that minimizing perplexity is, at least, locally optimal with respect to the set of explored hyperparameters.

Furthermore, for almost all samples, perplexity is minimized where correlation with qPCR measurements is maximized. For all replicates in conditions $\{B, C, D\}$, estimates that minimize perplexity with respect to held-out validation fragments achieve the best correlation with qPCR measured gene expression. For replicates in these conditions, abundances inferred using a prior size of 1 read-per-transcript resulted in estimates with the lowest perplexity. In replicates from condition A, estimates with lowest perplexity are significantly better than estimates at default hyperparameter settings (0.01 reads-per-transcript).

Perhaps surprisingly, both perplexity and correlation against qPCR measurements prefer a reads-per-transcript prior size that is larger than the 0.01 reads-per-transcript that is the current default for the `salmon` VBEM model. Selecting a larger per-transcript prior for transcript abundance estimation with `salmon` results in estimates that are more smooth. Compared to a sparser estimate, a smoother abundance estimate likely calls fewer validation time fragments impossible. Here, the number reads an estimate calls of impossible is symptomatic of two kinds of inferential errors — that some transcripts are incorrectly inferred to be unexpressed, and that other transcripts are assigned inaccurate inferred expression.

Without perplexity, it would be difficult to determine empirically, or *a-priori*, that a VBEM prior size of 1 is an optimal parameter setting since no comparison to ground-truth is possible. To the best of our knowledge, this experiment is the first to carry out both an effective and ubiquitously applicable quantitative strategy to perform model selection in the context of transcript abundance estimation on experimental data in the absence of ground truth.

5.4.2 Perplexity versus ground truth and differential expression analysis in simulated data

In simulated data, the relationship between perplexity and measurements against ground truth, though well-behaved, is admittedly less direct. In short, under the implemented experimental framework, minimizing perplexity does not always find the best performing estimates. Across all 10 samples, perplexity prefers abundance estimates that are smoother than estimates that are most accurate when compared to ground truth. For brevity, we include in the main text perplexity plots of three samples (`SRR1265{496,503,504}`) that are representative of three main modalities of perplexity plot behaviors (Fig. 5.8). For completeness, and plots for all samples are presented in the Figs. 5.14 and 5.15).

In all but two samples (`SRR1265{497,504}`), perplexity plots display a empirically convex shape with a local minima close to the optimal VBEM prior size (1 read-per-transcript). For example, for sample `SRR1265503`, perplexity is minimized at a VBEM prior setting of 1 reads-per-transcript, *exactly* the best performing hyperparameter setting with respect to Spearman correlation (Fig. 5.8; middle). And for sample `SRR1265496`, we can see that perplexity prefers VBEM prior setting in a wide local minima ranging from 1 to 3 reads-per-transcript (Fig. 5.8; top). Sample `SRR1265504` is one sample for which a local minimal perplexity cannot be identified with respect to the range of hyperparameters scanned (Fig. 5.8; bottom). However, the perplexity plot for `SRR1265504` displays a knee-like behavior which suggests that after a certain VBEM prior size, larger VBEM prior sizes are no longer preferred — which is consistent across all perplexity plots and comparisons to ground truth.

These experiments in simulated data suggest that, perhaps, perplexity remains an imperfect tool. Nonetheless, these observations do offer insights about how perplexity ought to be used in practice. First, perplexities may prefer abundance estimations smoother than ideal. In particular, when perplexities for two VBEM prior settings are close, or when perplexities are roughly minimized for a range of values, one ought to select the model that outputs the sparsest estimates. Second, careful (albeit qualitative) inspection of perplexity plots can be used to select an optimal hyperparameter setting experiment-wide. For example, inspection of perplexity plots (Figs. 5.14 and 5.15) over all samples show either knee-like behaviors beginning at, or local minimas centered close to a VBEM prior size of 1 reads-per-transcript — the best hyperparameter setting.

Notably, the results also show that perplexity can simply be used to quantitatively reject poor abundance estimates (or the hyperparameters that generate them). Although the significance of this property may be overlooked at first, perplexity is to our knowledge the only metric that can do so when ground truth is not available.

We also analyze the accuracy of differential transcript expression (DTE) analysis of estimates with the same VBEM prior size experiment-wide. We report AUROC of DTE calls up to a nominally useful maximum false discovery rate (FDR) of 0.05 (Fig. 5.9). Not surprisingly, AUROC of DTE calls mirror the shape of Spearman correlations of estimates inferred from different VBEM prior sizes. Again, for each individual sample, minimizing perplexities may not always select the best hyperparameter setting. But, experiment-wide, perplexity plots do begin to exhibit minima or knee-like behaviors at VBEM prior size of 1 reads-per-transcript — the best performing hyperparameter setting with regard to DTE (Fig. 5.9).

5.4.3 Perplexity measures improved accuracy due to additional `eXpress` online optimization rounds

Crucially, perplexity can be used to evaluate the performance of arbitrary abundance estimators that output per-transcript probabilities $\Pr(t_i \mid \boldsymbol{\theta})$. This is because, perplexity is computed from decoupled per-transcript terms $\Pr(t_i \mid \boldsymbol{\theta})$ from abundance estimates inferred only from the quantification fragment set, and per-fragment terms $\Pr(f_j \mid t_i)$ from mapping probabilities calculated only from the the validation fragment set. The comparison of different models simply requires agreement on per-fragment probabilities $\Pr(\hat{f}_j \mid t_i)$ for all fragments in the validation set. Per-fragment probabilities $\Pr(\hat{f}_j \mid t_i)$ can simply be computed from any tool that makes these available (e.g., `salmon`).

Thus, perplexity can be especially useful for investigating and verifying specific behaviors of different abundance estimation algorithms. To demonstrate this, we explore how perplexities can be calculated to investigate the improvement due to additional online optimization rounds when running `eXpress` [181]. `eXpress` uses a streaming optimization algorithm — online expectation-maximization (EM) — to quantify transcript abundance from the alignments of RNA-seq reads. Theoretically and empirically, additional rounds of the online-EM step is known to improve accuracy. Without perplexity, this behavior can only be verified when parallel measurements in experimental data are available(e.g., qPCR on biological replicates). With perplexity, this behavior can be verified from a sample's fragment-set directly. According to perplexities shown in Fig. 5.5, running `eXpress` using one or two additional online EM rounds results in improved abundance estimates in four out of five folds. In this case, the perplexity results concord with the expectation that additional rounds of the online-EM step improves convergence and lead to improved estimates of transcript abundance.

When running an inference algorithm, a user can go beyond simply verifying that an abundance estimation model converges on input, quantified fragments. With perplexity, a user can now verify that said model generalizes, and is accurate with respect to held-out, validation fragments drawn *exactly* from the "true" latent distribution.

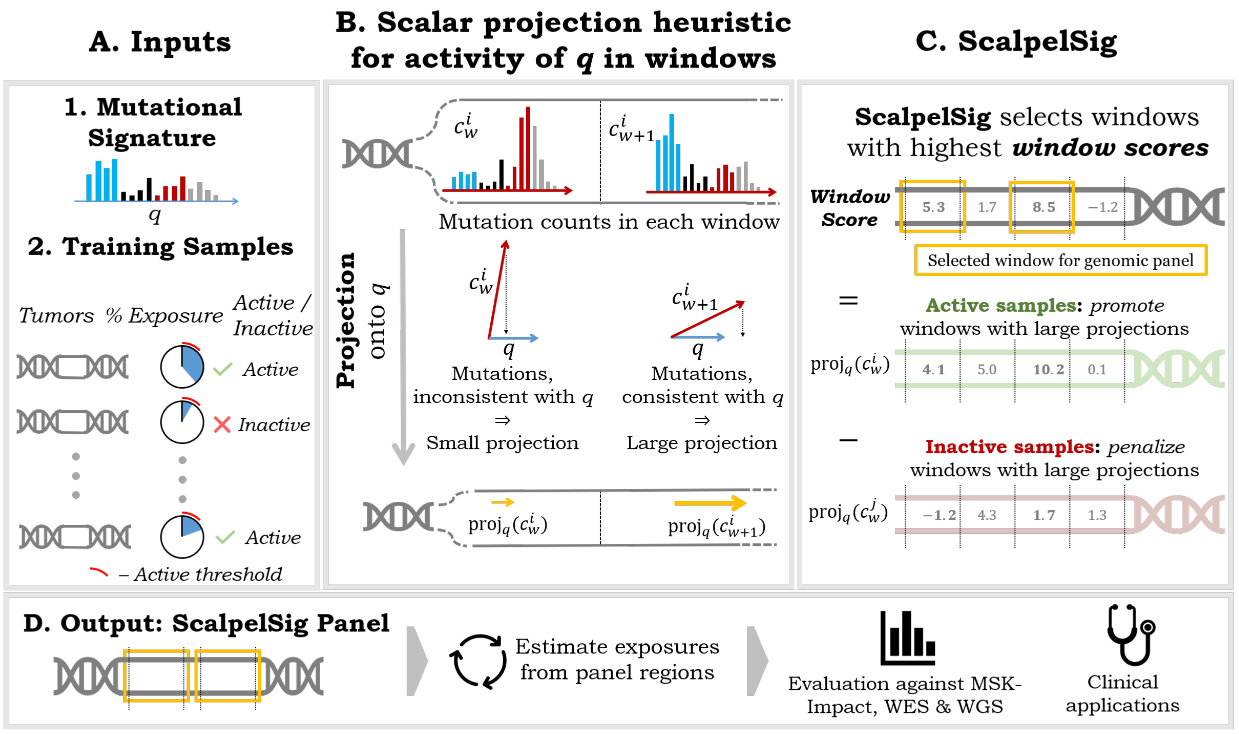

A. Inputs
B. Scalar projection heuristic for activity of q in windows
C. ScalpelSig
1. Mutational Signature
q
2. Training Samples
Tumors
% Exposure
Active / Inactive
Active
Inactive
Active
– Active threshold
c_w^i
c_{w+1}^i
Mutation counts in each window
Projection onto q
Mutations, inconsistent with q ⇒ Small projection
Mutations, consistent with q ⇒ Large projection
$\mathrm{proj}_q(c_w^i)$
$\mathrm{proj}_q(c_{w+1}^i)$
ScalpelSig selects windows with highest window scores
Window Score
5.3
1.7
8.5
−1.2
Selected window for genomic panel
=
Active samples: promote windows with large projections
$\mathrm{proj}_q(c_w^i)$
4.1
5.0
10.2
0.1
−
Inactive samples: penalize windows with large projections
$\mathrm{proj}_q(c_w^j)$
−1.2
4.3
1.7
1.3
D. Output: ScalpelSig Panel
Estimate exposures from panel regions
Evaluation against MSK-Impact, WES & WGS
Clinical applications

Figure 4.1: ScalpelSig designs a genomic panel to detect the activity of a given mutational signature. (A) ScalpelSig takes as input a mutational signature q and a set S of training samples with whole-genome sequencing data. Signature exposures are estimated for training samples. Samples which have more than 5% of mutations attributed to signature q are labeled active, and inactive otherwise. (B) Projection of each genome window's mutation count vector onto q yields a heuristic measure of signature exposure. The value of the scalar projection is highest when the given window has a high number of mutations with a distribution of mutation categories similar to q. (C) ScalpelSig evaluates windows with a contrastive window scoring function (see (4.8)). that encourages the selection of windows with large projections in active samples, and small projections in inactive samples. Given a panel size parameter N, ScalpelSig combines the N top scoring windows into a panel. (D) ScalpelSig panels detect signature activity with improved accuracy, and require considerably less genomic material than WES or WGS. Thus ScalpelSig panels offer an improved, clinically accessible assay of signature activity.

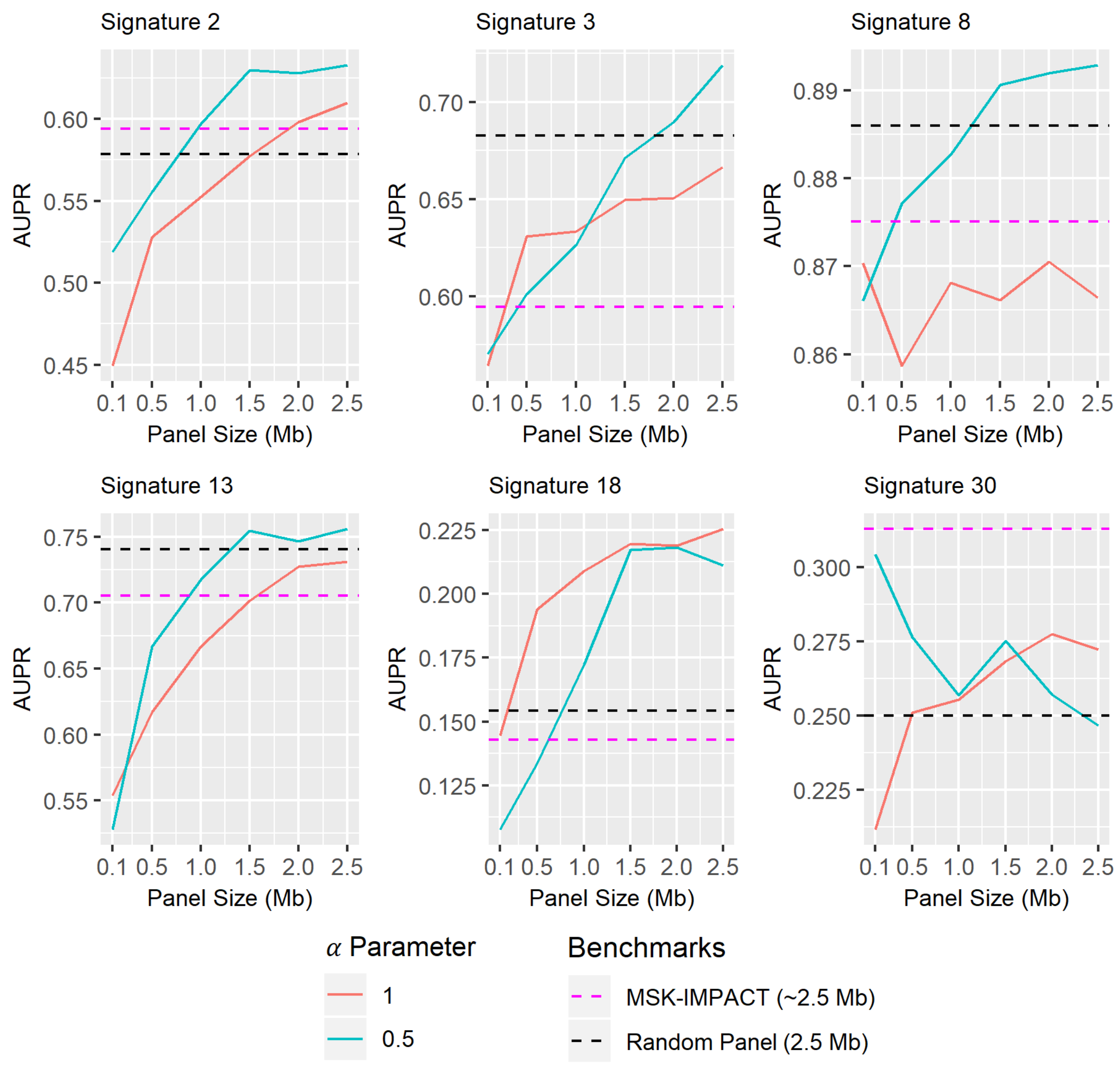


Figure 4.2: Assessment of genomic panels constructed with our framework for their ability to predict mutational signature activity at various panel sizes. Values shown are mean AUPR across 15 randomized test and train sets. Each plot tests distinct panels optimized for the detection of a particular mutational signature using two different settings of the α parameter: $\alpha = 1$ (orange) and $\alpha = 0.5$ (blue). Values are compared against a randomized baseline panel (dotted black line; see Methods for details), and mean AUPR of the MSK-IMPACT panel (dotted magenta line), for the same test sets.

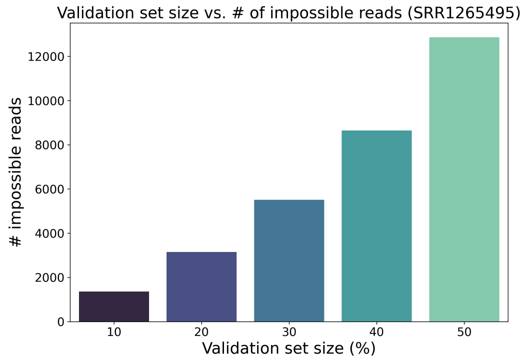


Figure 5.1: Number of fragments called impossible versus withheld validation fragment set size for sample `SRR1265495`. All remaining fragments are used to estimate abundances using `salmon`'s VBEM model using default parameters (i.e. using a prior size of 0.01 reads-per-transcript).

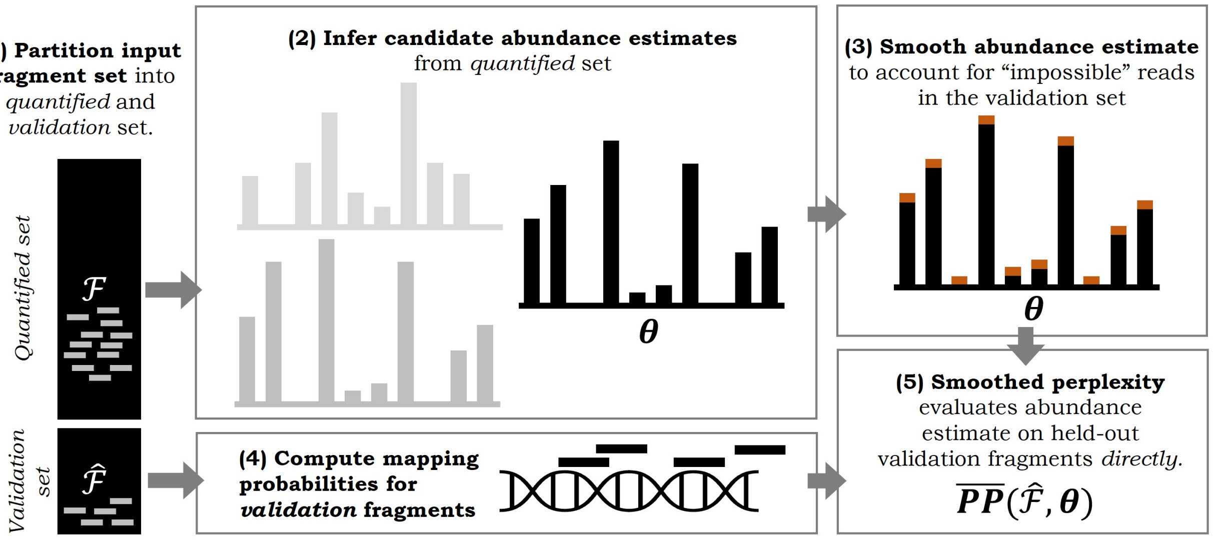

(1) Partition input fragment set into quantified and validation set.
Quantified set
$\mathcal{F}$
Validation set
$\hat{\mathcal{F}}$
(2) Infer candidate abundance estimates from quantified set
$\boldsymbol{\theta}$
(3) Smooth abundance estimate to account for "impossible" reads in the validation set
$\boldsymbol{\theta}$
(4) Compute mapping probabilities for validation fragments
(5) Smoothed perplexity evaluates abundance estimate on held-out validation fragments directly.
$\overline{PP}(\hat{\mathcal{F}}, \boldsymbol{\theta})$

Figure 5.2: Overview of the *quantify-then-validate* approach using *smoothed perplexity* to evaluate the quality of abundance estimates directly on fragment sets in the absence of ground truth. (1) An input fragment set is first partitioned into a *quantified* and a *validation* set. (2) Abundance estimates for different candidate models (e.g. for explored hyperparameters as part of model selection) are inferred from the *quantified* fragment set only. (3) To account for "*impossible*" fragments and avoid shrinkage to unbounded perplexities, given abundance estimates are smoothed (see Section 5.3.2). (4) Mapping probabilities to the reference transcriptome are computed for fragments in the validation set. (5) *Smoothed perplexity* is computed given each input abundance estimate and the held-out validation fragment set to evaluate and perform model selection — the lower the perplexity, the better an abundance estimate describes the held-out set of validation fragments.

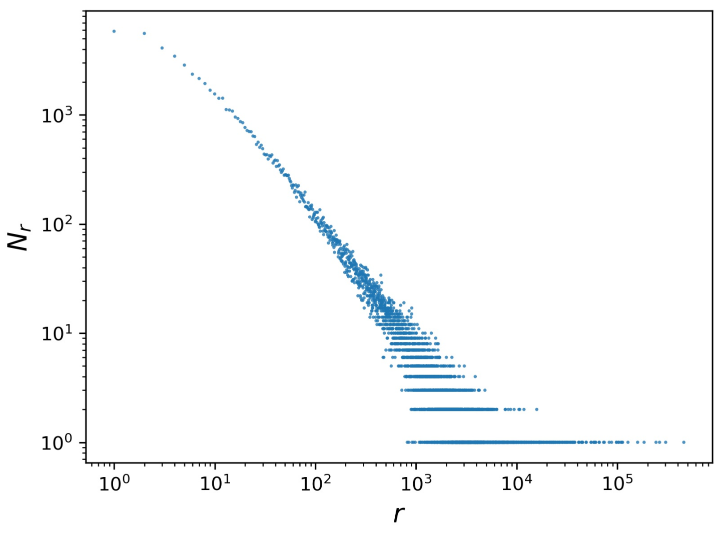


Figure 5.3: Frequencies-of-frequencies follow log-linear distribution for SEQC sample A1.

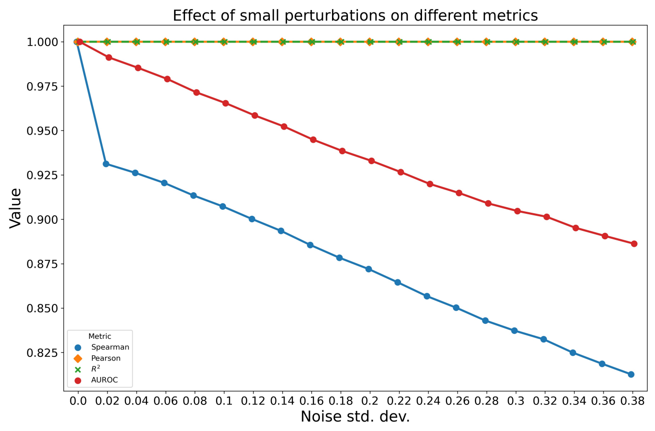


Figure 5.4: Spearman correlation, Pearson correlation and R^2 with respect to all transcripts in the reference, and AUROC for recalling ground truth unexpressed transcripts, with respect to added normally distributed noise with varying standard deviations. Plotted lines for Pearson correlation and R^2 overlap.

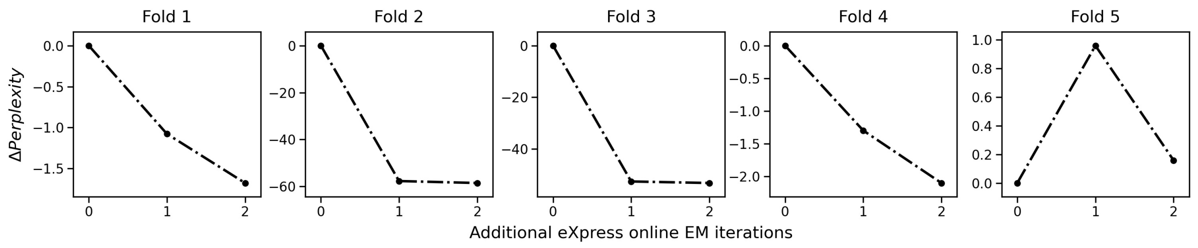


Figure 5.5: Change in perplexity from additional eXpress online expectation-maximization (EM) rounds. Reduction in perplexity indicates improved quality of estimated abundances after each online EM round.

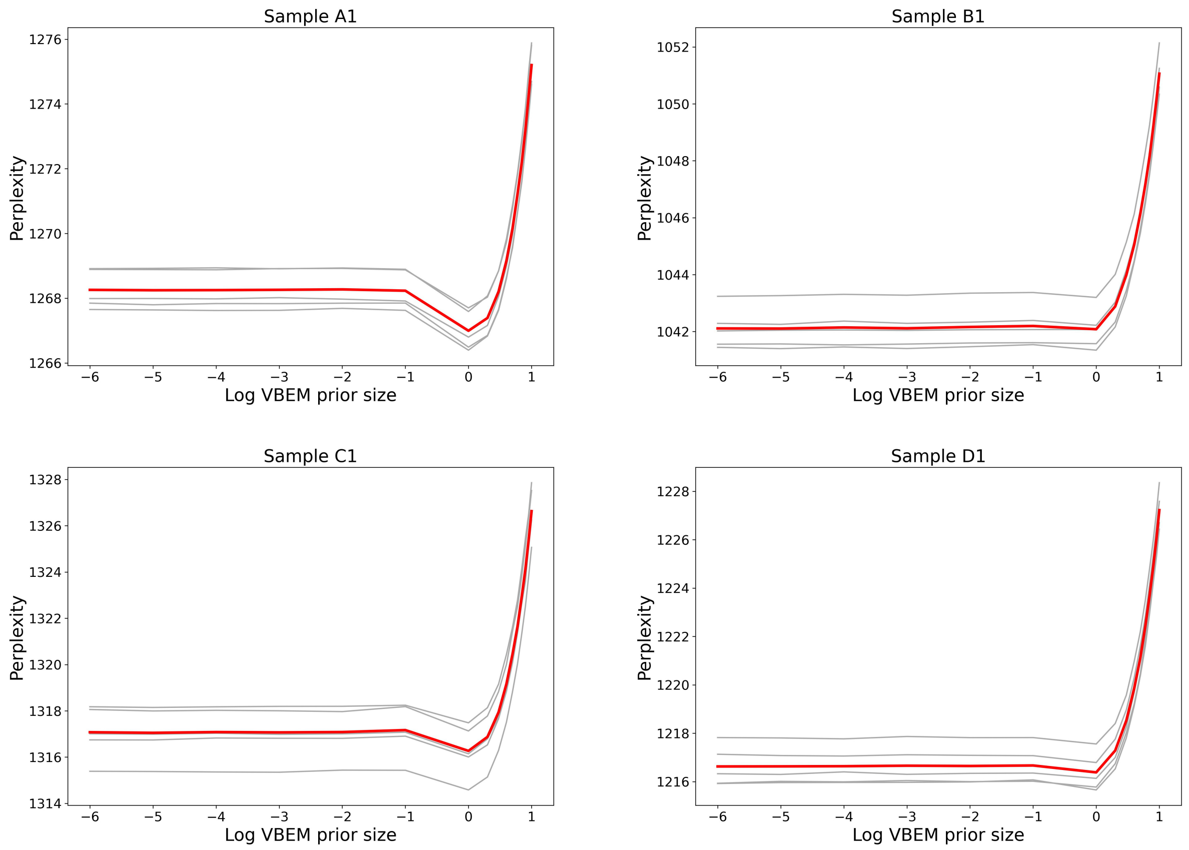


Figure 5.6: Perplexity plots for SEQC samples. Plots show perplexity versus VBEM reads-per-transcript prior size for SEQC samples — plots only for the first replicate of samples from conditions A-D are shown. Perplexity plots for other replicates are consistent within condition and are included in the Appendix. Mean perplexities across five folds are plotted in red, and perplexities for each fold are plotted in gray.

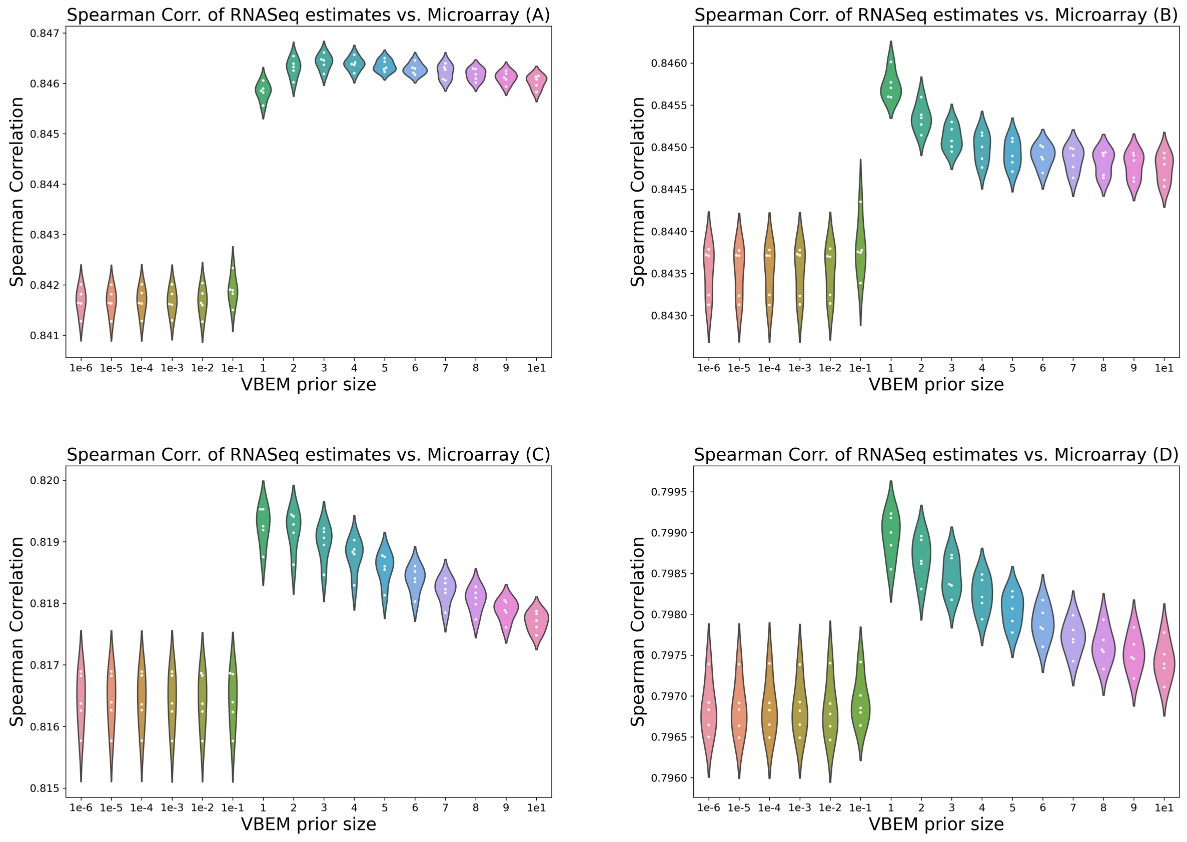


Figure 5.7: Spearman correlation of abundance estimates at various VBEM reads-per-transcript prior sizes, versus parallel qPCR microarray gene-expression measurements conditions *A*-*D*. Each point in above plots indicate the mean correlation across replicates for a given fold.

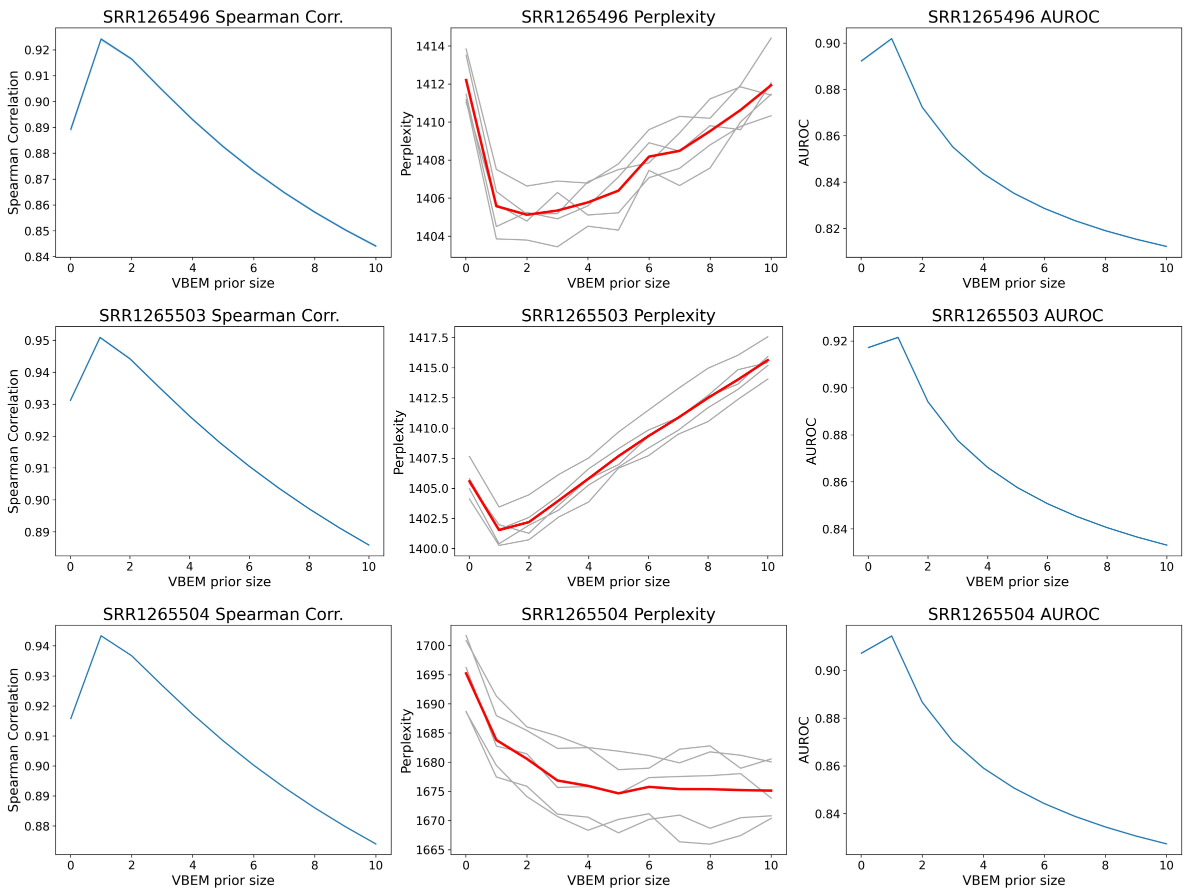


Figure 5.8: Quality of transcript abundance estimates as a function of VBEM per-nucleotide prior size for samples SRR1265{496,503,504}. (Left column) Spearman Correlation with respect ground truth expressed transcripts. (Middle column) Perplexity of abundance estimates (perplexities per-fold indicated in gray and mean perplexities in red). (Right column) AUROC for retrieving ground truth unexpressed transcripts. Leftmost plotted points for all plots use default salmon VBEM prior size of 0.01 reads-per-transcript.

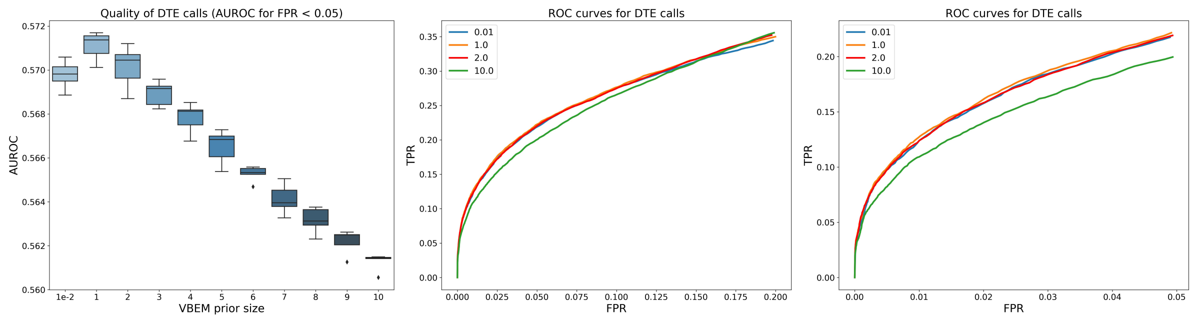


Figure 5.9: Accuracy of differential expression analysis with respect to experiment-wide selection of VBEM per-nucleotide prior size. (Left) AUROC with respect to DTE calls at real FPRs up to 0.05. (Middle) ROC curve up to FPR = 0.20. (Right) ROC curve up to FPR = 0.05. To reduce visual clutter, only the ROC curves some representative VBEM prior size settings are plotted.

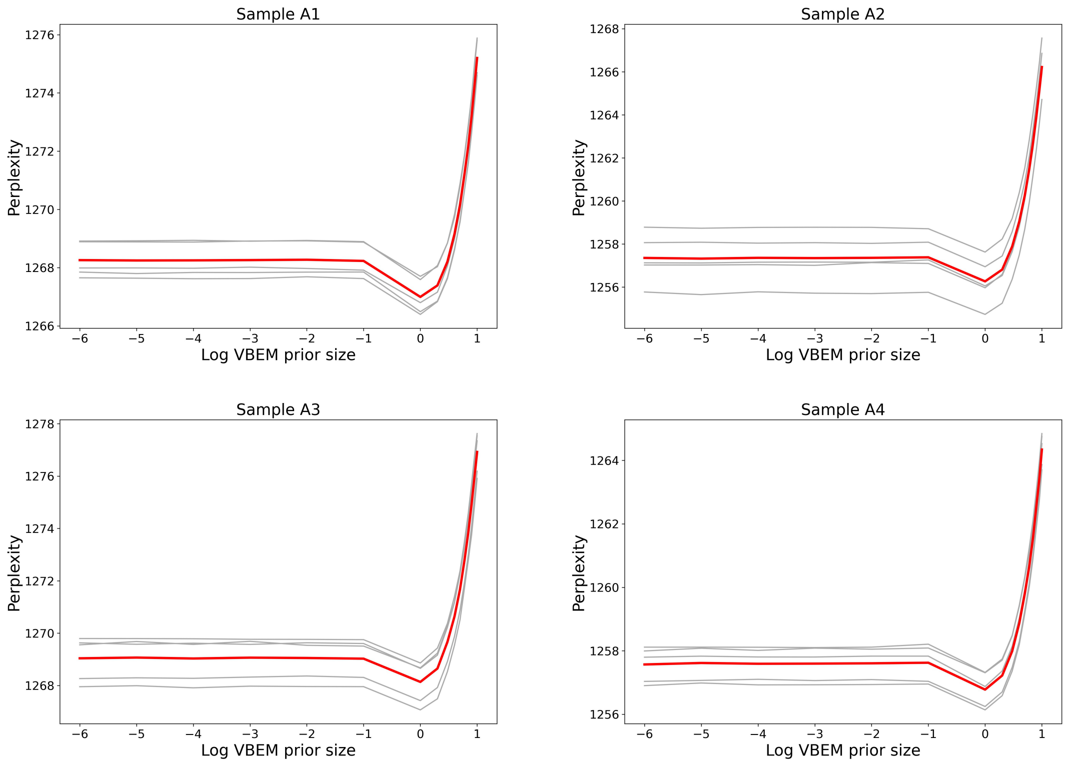


Figure 5.10: Perplexity plots for SEQC A samples. Plots show perplexity versus VBEM reads-per-transcript prior size for SEQC samples. Mean perplexities across five folds are plotted in red, and gray perplexities for each fold are plotted are plotted in gray.

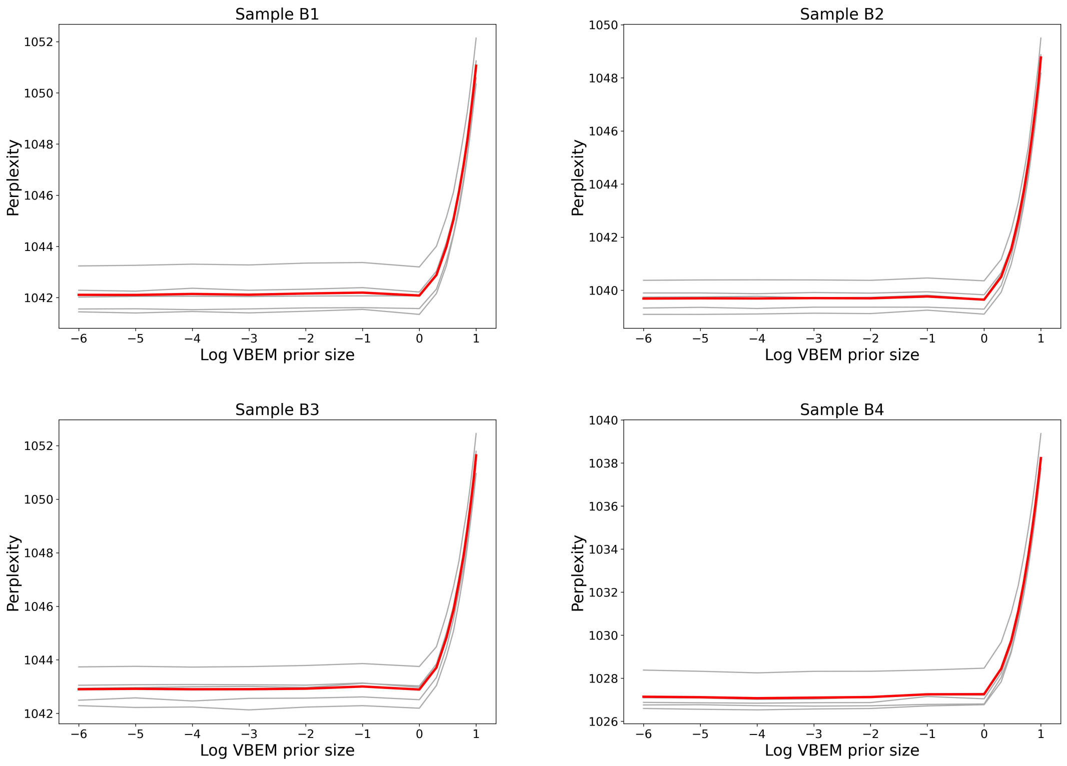


Figure 5.11: Perplexity plots for SEQC B samples. Plots show perplexity versus VBEM reads-per-transcript prior size for SEQC samples. Mean perplexities across five folds are plotted in red, and gray perplexities for each fold are plotted are plotted in gray.

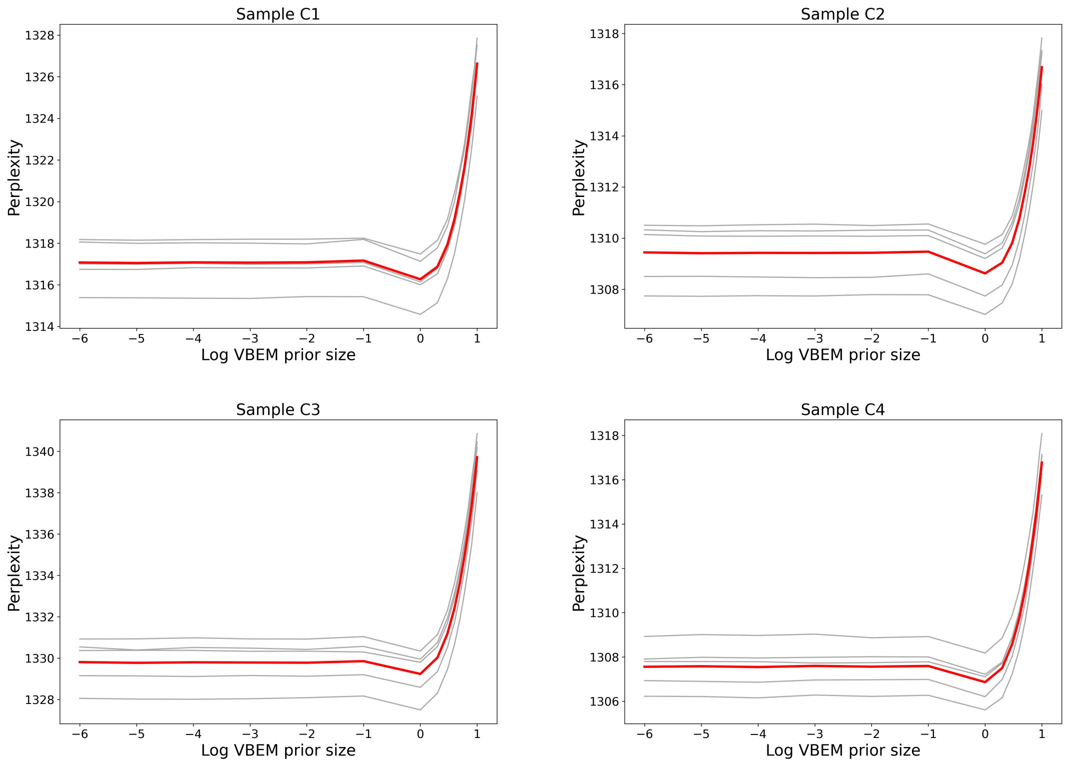


Figure 5.12: Perplexity plots for SEQC C samples. Plots show perplexity versus VBEM reads-per-transcript prior size for SEQC samples. Mean perplexities across five folds are plotted in red, and gray perplexities for each fold are plotted are plotted in gray.

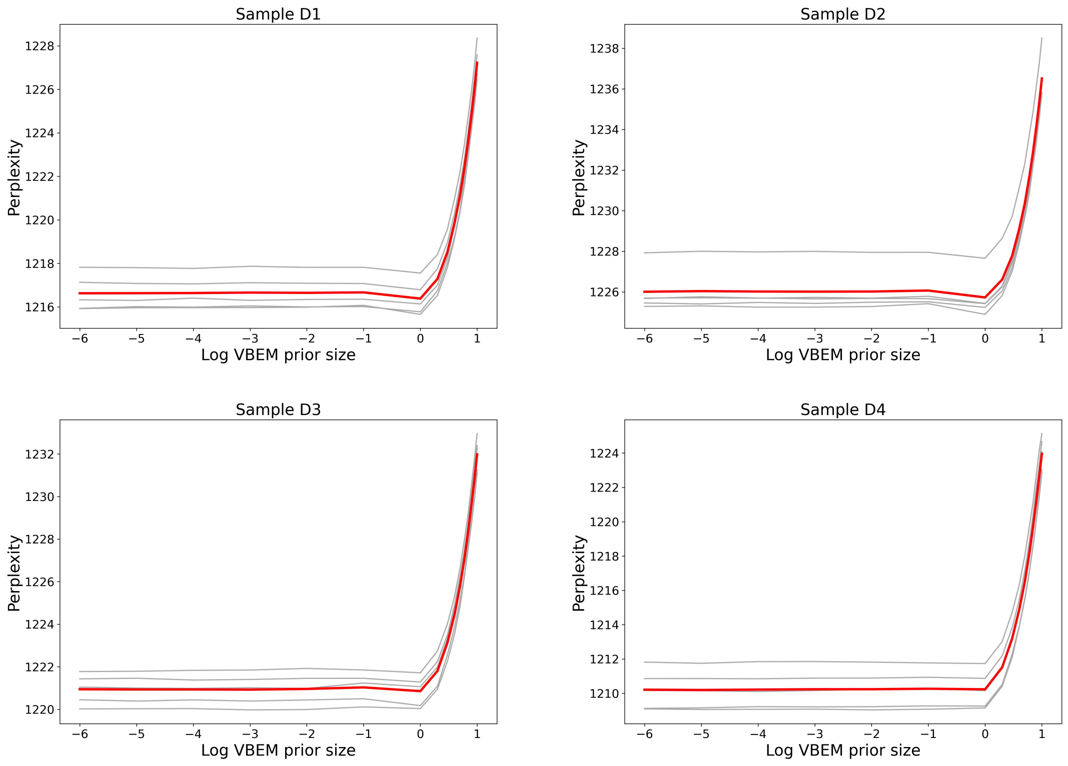


Figure 5.13: Perplexity plots for SEQC D samples. Plots show perplexity versus VBEM reads-per-transcript prior size for SEQC samples. Mean perplexities across five folds are plotted in red, and gray perplexities for each fold are plotted are plotted in gray.

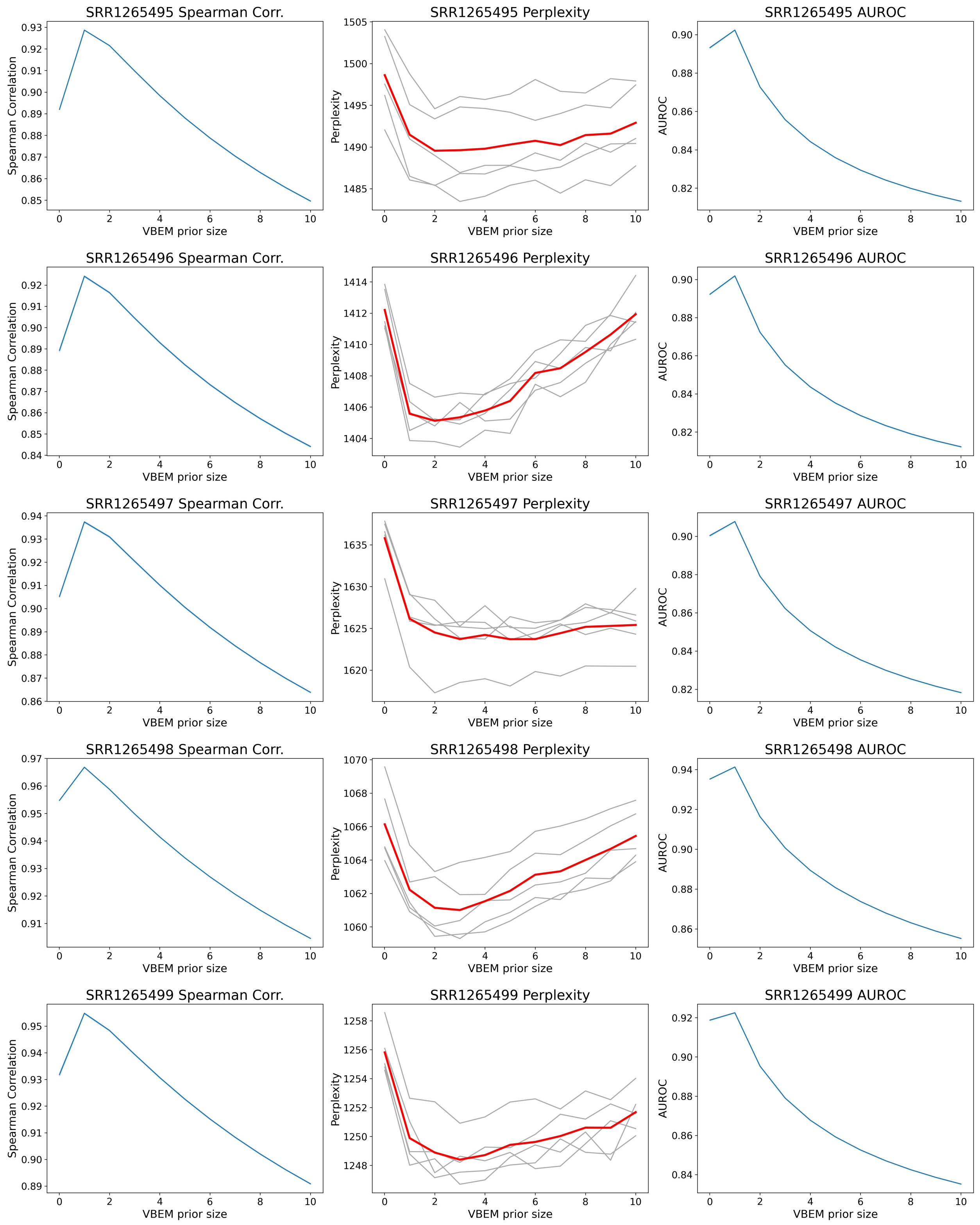


Figure 5.14: Quality of transcript abundance estimates as a function of VBEM per-nucleotide prior size for samples `SRR1265{495-499}`. (Left column) Spearman Correlation with respect ground truth expressed transcripts. (Middle column) Perplexity of abundance estimates; perplexities per-fold indicated in gray and mean perplexities in red. (Right column) AUROC for retrieving ground truth unexpressed transcripts. Leftmost plotted points for all plots use default `salmon` VBEM prior size of 0.01 reads-per-transcript.

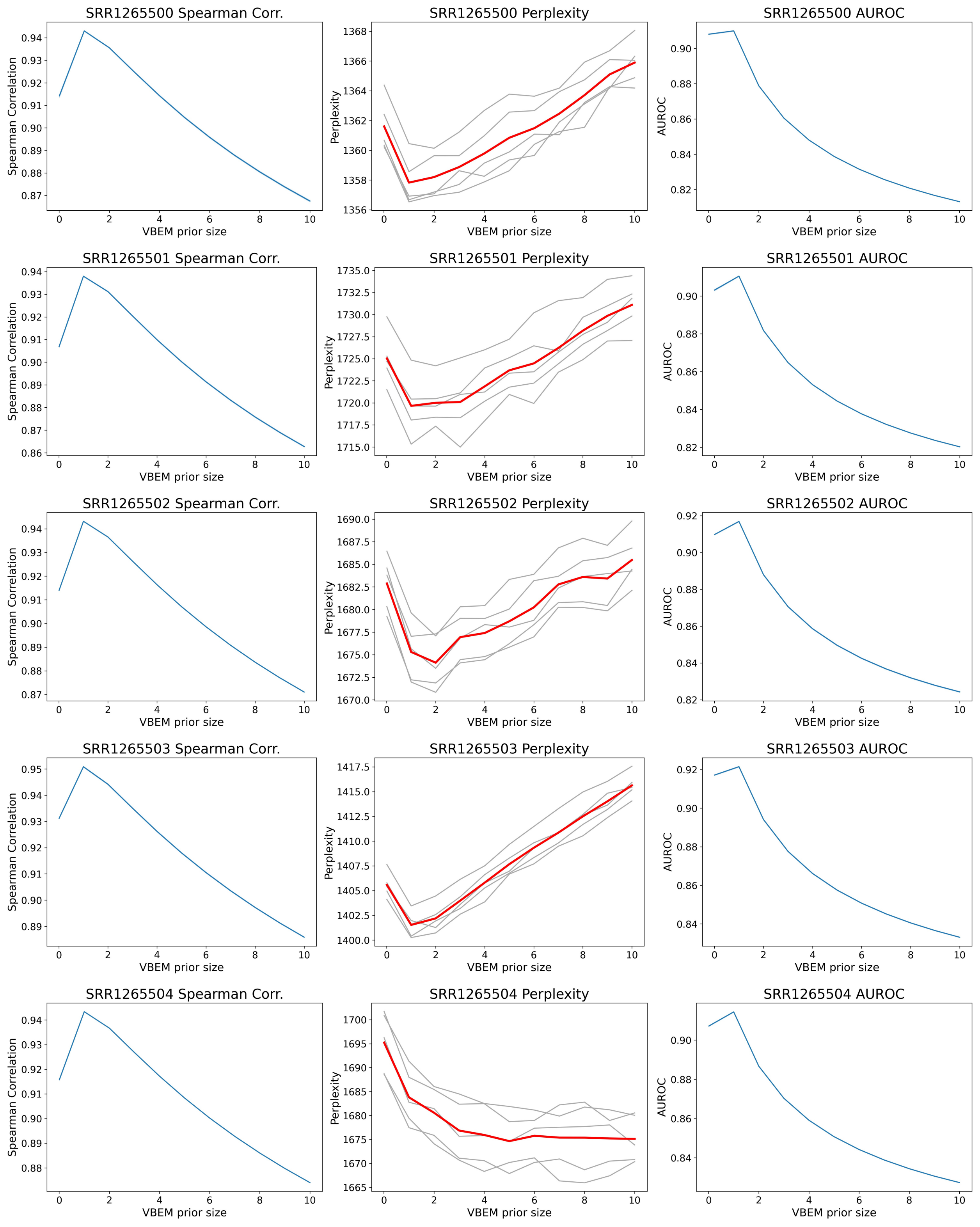


Figure 5.15: Quality of transcript abundance estimates as a function of VBEM per-nucleotide prior size for samples `SRR1265{500-504}`.

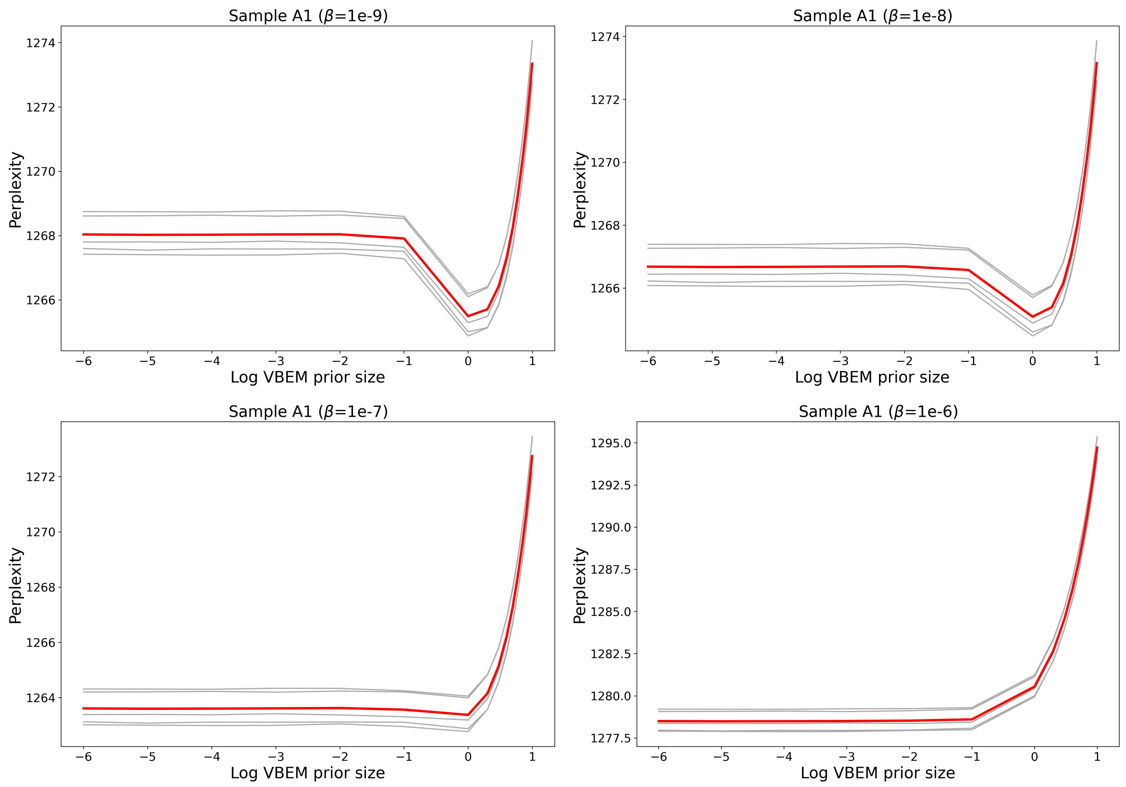


Figure 5.16: Perplexity plots for SEQC sample $A1$ at different smoothing parameter settings. Plots show perplexity versus VBEM reads-per-transcript prior size for SEQC samples. Mean perplexities across five folds are plotted in red, and gray perplexities for each fold are plotted are plotted in gray.

5.5 Discussion

In this work, we derive the smoothed perplexity metric, which, to our knowledge, is the first metric that enables the evaluation of the quality of transcript abundance estimates in the absence of ground truth.

In experimental data from the Sequencing Quality Control (SEQC) project [160], we show that the most accurate abundance estimates consistently have the lowest perplexity (lower is better) and demonstrate how quantitative model selection can be performed on input fragment sets directly and in the absence of ground truth. In simulated samples, we demonstrate a looser, but still useful, relationship between perplexity and measurements against ground truth. One possible explanation for the more erratic behavior and noisier perplexity plots for our simulated samples is due to these samples consisting of many fewer fragments than SEQC samples. On average, the simulated samples contain 17,410,732 fragments on average while the SEQC samples average 47,589,281 fragments.

Although we only demonstrate model selection with respect to only one hyperparameter (the VBEM prior size) in `salmon` using perplexity, model selection for other hyperparameters are possible with simple changes to the experimental protocols implemented here. For example, perplexity evaluated to choose the number of bins for the range-factorized likelihood approximation, or select between VBEM and EM models and optimization algorithms in `salmon`.

Notably, perplexity may be useful for investigating and comparing different abundance estimation models. In a proof-of-concept style experiment running `eXpress` [181], we demonstrate perplexity can be computed to verify theoretically predicted behavior. In doing so, we theoretically and empirically demonstrate that perplexity can be computed for *almost any* transcript abundance estimation model.

In future work, perplexity can perhaps be adapted and applied to other problem settings in bioinformatics where probabilistic models infer abundances. For example, perplexity may be useful in metagenomics where model selection (i.e., choosing confidence cutoffs for taxa identification, or selecting candidate reference genomes) can have a large effect on the quality of inferred abundances [18].

In sum, this work demonstrates that evaluation of transcript abundance estimates in the absence of ground truth is indeed possible. Perplexity is an example of a promising new direction in which estimated abundances can be evaluated and validated *directly* on input fragments themselves. This may prove fruitful not only for the re-analysis of previously published data where ground truth was absent, but also for current and future experimental settings where parallel experimental measurements complementary to RNASeq are too expensive or cumbersome to obtain.

Part III

Indexing

Chapter 6: Spectrum preserving tilings enable sparse and modular indexing

Disclosure

This Chapter presents first-author work presented at, and preprinted for, the 27th Annual International Conference on Research in Computational Molecular Biology [182], with minimal changes. Exciting directions for future work that are discussed in more detail in the supplement of the preprinted manuscript has been incorporated as the last section in this Chapter.

6.1 Background and Motivation

Indexing of genomic sequences is an important problem in modern computational genomics, as it enables the atomic queries required for analysis of sequencing data — particularly *reference guided* analyses where observed sequencing data is compared to known *reference* sequences. Fundamentally, analyses need to first rapidly locate short exact matches to reference sequences before performing other operations downstream. For example, for guided assembly of genomes, variant calling, and structural variant identification, seed sequences are matched to known references before novel sequences are arranged according to the seeds [183]. For RNA-seq, statistics for groups of related k-mers mapping to known transcripts or genes allow algorithms to infer the activity of genes in single-cell and bulk gene-expression analyses [20, 184, 185].

Recently, researchers have been interested in indexing collections of genomes for metagenomic and pan-genomic analyses. There have been two main types of approaches: full-text indexes, and hashing based approaches that typically index the *de Bruijn graph* (dBG). With respect to full-text indexes, researchers have developed tools that use the *r-index* [186] to compute matching statistics and locate maximal exact matches for large reference collections [187, 188]. For highly repetitive collections, such as many genomes from the same species, r-index based approaches are especially space efficient since they scale linearly to the number of runs in the *Burrows-Wheeler Transform* (BWT) [189] and not the length of the reference text. With respect to hashing based approaches, tools restrict queries to fixed length k-mers [183, 190] and index the dBG. These tools achieve faster exact queries but typically trade off space. In other related work, graph-based indexes that compactly represent genomic variations as paths on graphs have also been developed [191, 192]. However, these indexes require additional work to project queries landing on graph-based coordinates to linear coordinates on reference sequences.

Many tools have been developed to efficiently build and represent the dBG [193, 194]. Recently, Khan et al. introduced a pair of methods to construct the compacted dBG from both assembled references [195] and read sets [196]. Ekim et al. [197] introduced the minimizer-space dBG — a highly effective lossy compression scheme that uses minimizers as representative sequences for nodes in the dBG. Karasikov et al. developed the Counting dBG [198] that stores differences between adjacent nodes in the dBG to compress metadata associated with nodes (and sequences) in a dBG. Encouragingly, much recent work on *Spectrum Preserving String Sets* (SPSS) that compactly index the set-membership of k-mers in reference texts has been introduced [21, 22, 23, 196, 199, 200, 201]. Although these approaches do not tackle the *locate* queries directly, they do suggest that even more efficient solutions for reference indexing are possible.

In this work, we extend these recent ideas and introduce the concept of a *Spectrum Preserving Tiling* (SPT) which encodes how and where k-mers in an SPSS occur in a reference text. In introducing the SPT, this work makes two key observations. First, a hashing based solution to the reference indexing problem for k-mers does not necessitate a de Bruijn graph but instead requires a *tiling* over the input reference collection — the SPT formalizes this. Second, the reference indexing problem for k-mers queries can be cleanly decomposed into a *k-mer-to-tile* query and a *tile-to-occurrence* query. Crucially, SPTs enable the implementation and analysis of a general class of modular indexes that can exploit efficient implementations introduced in prior work.

***.** Contributions. We focus our work on considering how indexes can, *in practice*, efficiently support the two composable queries — the *k-mer-to-tile* query and the *tile-to-occurrence* query. We highlight this work's key contributions below. We introduce:

1. The *spectrum preserving tiling* (SPT). An SPT is a general representation that explicitly encodes how shared sequences — *tiles* — repeatedly occur in a reference collection. The SPT enables an entire *class* of sparse and modular indexes that support exact locate queries for k-mers.

2. An algorithm for sampling and compressing an indexed SPT built from unitigs that *samples* unitig-occurrences. For some small constant "sampling rate", s, our algorithm stores the positions of only $\approx 1/s$ occurrences and encodes all remaining occurrences using a small *constant* number of bits.

3. `Pufferfish2`: a practical index and implementation of the introduced sampling scheme. We highlight the critical engineering considerations that make `pufferfish2` effective in practice.

6.2 Problem definition and preliminaries

***.** The mapped reference position (MRP) query. In this work we consider the *reference indexing problem for k-mers*. Given a collection of references $\mathcal{R} = \{R_1, \dots, R_N\}$, where each reference is a string over the DNA alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}$, we seek an index that can efficiently compute the *mapped reference position* (MRP) query for a fixed k-mer size k. Given any k-mer x, the MRP query enumerates the positions of all occurrences of x in $\mathcal{R}$. Precisely, each returned occurrence is a tuple (n, p) that specifies that k-mer, x, occurs in reference n at position p where $R_n[p : p+k] = x$. If a k-mer does not occur in some $R_n \in \mathcal{R}$, the MRP query returns an empty list.

***.** Basic notation. Strings and lists are zero-indexed. The length of a sequence S is denoted $|S|$. The i-th character of a string S is $S[i]$. A k-mer is a string of length k. A sub-string of length ℓ in the string S starting at position i is notated $S[i : i + \ell]$. The prefix and suffix of length i is denoted $S[: i]$ and $S[|S| - i :]$, respectively. The concatenation of strings A and B is denoted $A \circ B$.

We define the *glue* operation, $A \oplus_k B$, to be valid for any pair of strings A and B that overlap by $(k-1)$ characters. If the $(k-1)$-length suffix of A is equal to the $(k-1)$-length prefix of B, then $A \oplus_k B := A \circ B[(k-1) :]$. When k clear from context, we write $A \oplus B$ in place of $A \oplus_k B$.

***.** Rank and select queries over sequences. Given a sequence S, the *rank* query given a character α and position i, written $\mathtt{rank}_\alpha(S, i)$, is the number of occurrences of α in $S[: i]$ The *select* query $\mathtt{select}_\alpha(S, r)$ returns the position of the r-th occurrence of symbol α in S. The *access* query $\mathtt{access}(S, i)$ returns $S[i]$. For a sequence of length n over an alphabet of size σ, these can be computed in $O(\lg \sigma)$ time using a *wavelet matrix* that requires $n \lg \sigma + o(n \lg \sigma)$ bits [202].

6.3 Spectrum preserving tilings

In this section, we introduce the *spectrum preserving tiling*, a representation of a given reference collection $\mathcal{R}$ that specifies how a set of *tiles* containing k-mers repeatedly occur to spell out the constituent reference sequences in $\mathcal{R}$. This alternative representation enables a modular solution to the reference indexing problem, based on the interplay between two mappings — a k-mer-to-tile mapping and a tile-to-occurrence mapping.

6.3.1 Definition

Given a k-mer length k and an input reference collection of genomic sequences $\mathcal{R} = \{R_1, \dots, R_N\}$, a spectrum preserving tiling (SPT) for $\mathcal{R}$ is a five-tuple $\Gamma := (\mathcal{U}, \mathcal{T}, \mathcal{S}, \mathcal{W}, \mathcal{L})$:

- **Tiles**: $\mathcal{U} = \{U_1, \dots, U_F\}$. The set of *tiles* is a spectrum preserving string set, i.e., a set of strings such that each k-mer in $\mathcal{R}$ occurs in some $U_i \in \mathcal{R}$. Each string $U_i \in \mathcal{U}$ is called a *tile*.
- **Tiling sequences**: $\mathcal{T} = \{T_1, \dots, T_N\}$ where each T_n corresponds to each reference $R_n \in \mathcal{R}$. Each tiling sequence is an ordered sequence of tiles $T_n = [T_{n,1}, \dots, T_{n,M_n}]$, of length M_n, with each $T_{n,m} = U_i \in \mathcal{U}$. We term each $T_{n,m}$ a *tile-occurrence*.
- **Tile-occurrence lengths**: $\mathcal{L} = \{L_1, \dots, L_N\}$, where each $L_n = [l_{n,1}, \dots, l_{n,M_n}]$ is a sequence of lengths.
- **Tile-occurrence offsets**: $\mathcal{W} = \{W_1, \dots, W_N\}$, where each $W_n = [w_{n,1}, \dots, w_{n,M_n}]$ is an integer-sequence.
- **Tile-occurrence start positions**: $\mathcal{S} = \{S_1, \dots, S_N\}$, where each $S_n = [s_{n,1}, \dots, s_{n,M_n}]$ is

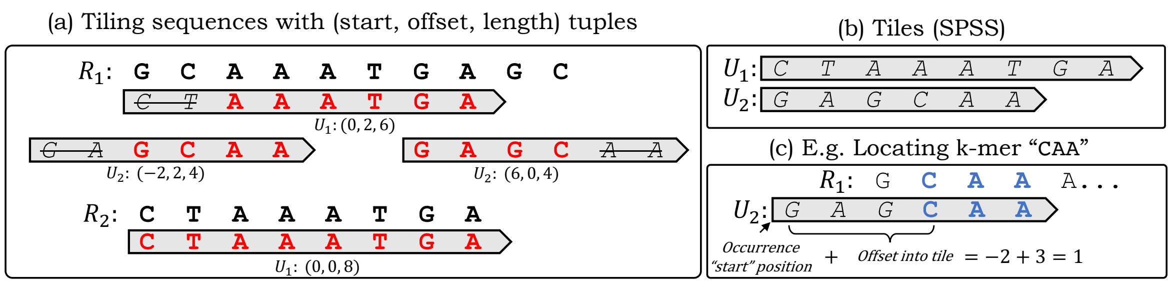


Figure 6.1: (a) A spectrum preserving tiling (SPT) with $k = 3$, (b) with tiles (an SPSS) that contain all k-mers in references. (c) The SPT explicitly encodes where each k-mer occurs.

an integer-sequence.

A valid SPT must satisfy the *spectrum preserving tiling property*, that every reference sequence R_n can be reconstructed by gluing together *substrings of tiles* at offsets W_n with lengths L_n:

$$R_n = T_{n,1}[w_{n,1} : w_{n,1} + l_{n,1}] \oplus ... \oplus T_{n,M_n}[w_{n,M_n} : w_{n,M_n} + l_{n,M_n}].$$

Specifically, the SPT encodes how redundant sequences — *tiles* — repeatedly occur in the reference collection $\mathcal{R}$. We illustrate how an ordered sequence of start-positions, offsets, and lengths explicitly specify how redundant sequences tile a pair of references in Fig. 6.1. More succinctly, each tile-occurrence $T_{n,m}$ with length $l_{n,m}$ tiles the reference sequence R_n as:

$$R_n[s_{n,m} + w_{n,m} : s_{n,m} + w_{n,m} + l_{n,m}] = T_{n,m}[w_{n,m} : w_{n,m} + l_{n,m}].$$

In the same way a small SPSS compactly determines the *presence* of a k-mer, a small SPT compactly specifies the *location* of a k-mer. For this work, we consider SPTs where any k-mer occurs only once in the set of tiles $\mathcal{U}$. The algorithms and ideas introduced in this paper still work with SPTs where a k-mer may occur more than once in $\mathcal{U}$ (some extra book-keeping of a one-to-many k-mer-to-tile mapping would be needed, however). For ease of exposition, we ignore tile orientations here. We completely specify the SPT with orientations, allowing tiles to simultaneously represent reverse-complement sequences, in Appendix A.2.

6.3.2 A general and modular index over spectrum preserving tilings

Any SPT is immediately amenable to indexing by an entire *class* of algorithms. This is because an SPT yields a natural decomposition of the MRP query (defined in Section 6.2) where k-mers first map to the tiles and tile-occurrences then map to positions in references. To index a reference collection, a data structure need only compose a query for the positions where k-mers occur on tiles in a SPSS with a query for the positions where tiles cover the input references.

Ideally, an index should find a small SPT where k-mers are compactly represented in the set of tiles where tiles are "long" and tiling sequences are "short". Compact tilings exist for almost all practical applications since the amount of *unique* sequence grows much more slowly than the *total* length of reference sequences. Finding a small SPSS where k-mers occur only once has been solved efficiently [21, 22, 199]. However, it remains unclear if a small SPSS induces a small SPT, since an SPT must additionally encode tile-occurrence positions. Currently, tools like `pufferfish` index reference sequences using an SPT built from the *unitigs* of the compacted de Bruijn graph (cdBG) constructed over the input sequences, which has been found to be sufficiently compact for practical applications. Though the existence of SPSSs smaller than cdBGs suggest that smaller SPTs might be found for indexing, we leave the problem of finding small or even optimal SPTs to future work. Here, we demonstrate how indexing any given SPT is *modular* and possible in general.

Given an SPT, the MRP query can be decomposed into two queries that can each be supported by sparse and efficient data structures. These queries are:

- **The kmer-to-tile query**: Given a k-mer x, $\texttt{k2tile}(x)$ returns (i, p) — the identity of the tile U_i that contains x and the offset (position) into the tile U_i where x occurs. That is, $\texttt{k2tile}(x) = (i, p)$ iff $U_i[p : p + k] = x$. If x is not in $\mathcal{R}$, $\texttt{k2tile}(x)$ returns $\emptyset$.

- **The tile-to-occurrence query**: Given the r-th occurrence of the tile U_i, $\texttt{tile2occ}(i, r)$ returns the tuple (n, s, w, l) that encodes how U_i tiles the reference R_n. When $\texttt{tile2occ}(i, r) = (n, s, w, l)$, the r-th occurrence of U_i occurs on R_n at position $(s + w)$, with the sequence $U_i[w : w + l]$. Let the r-th occurrence of U_i be $T_{n,m}$ on $\mathcal{T}$, then $\texttt{tile2occ}(i, r)$ returns $(n, s_{n,m}, w_{n,m}, l_{n,m})$.

When these two queries are supported, the MRP query can be computed by Algorithm 1. By adding the offset of the queried k-mer x in a tile U_i to the positions where the tile U_i occurs, Algorithm 1 returns all positions where a k-mer occurs. Line 10 checks to ensure that any occurrence of the queried k-mer is returned only if the corresponding tile-occurrence of U_i contains that k-mer. We note that storing the number of occurrences of a tile and returning $\texttt{num-occs}(U_i)$ requires negligible computational overhead. In practice, the length of tiling sequences, $\mathcal{T}$, are orders of magnitude larger than the number of unique tiles. In this work, we shall use occ_i, to denote the number of occurrences of U_i in tiling sequences $\mathcal{T}$.

Algorithm 1:

```
def mrp($x$):
    $tup \leftarrow$ k2tile($x$)
    if $tup = \emptyset$ then
        return [ ]
    $(i, p) \leftarrow tup$
    $occ_i \leftarrow$ num-occs($U_i$)
    $ans \leftarrow$ [ ]
    for $r \leftarrow 0$ to $occs_i$ do
        $(n, s, w, l) \leftarrow$ tile2occ($i, r$)
        if $w \leq p \leq (w + l - k)$ then
            $ans$.append($n, s + p$)
    return ans
```

6.3.3 "Drop in" implementations for efficient k-mer-to-tile queries

Naturally, prior work for indexing and compressing spectrum preserving string sets (SPSS) can be applied to implement the k-mer-to-tile query. When `pufferfish` was first developed, the data structures required to support the k-mer-to-tile query dominated the size of moderately sized indexes. Thus, Almodaresi et al. [190] introduced a sampling scheme that samples k-mer positions in unitigs. Recently, Pibiri [23, 200] introduced `SSHash`, an efficient k-mer hashing scheme that exploits minimizer based partitioning and carefully handles highly-skewed distributions of minimizer occurrences. When built over an SPSS, `SSHash` stores the k-mers by their order of appearance in the strings (which we term tiles) of an SPSS and thus allows easy computation of a k-mer's offset into a tile. Other methods based on the Burrows-Wheeler transform (BWT) [189], such as the Spectral BWT [201] and BOSS [203], could also be used. However, these methods implicitly sort k-mers in lexicographical order and would likely need an extra level of indirection to implement `k2tile`. Unless a compact scheme is devised, this can outweigh the space savings offered by the BWT.

6.3.4 Challenges of the tile-to-occurrence query

The straightforward solution to the tile-to-occurrence query is to store the answers in a table,`utab`, where `utab`$[i]$ stores information for all occurrences of the tile U_i and computing `tile2occ`(i, r) amounts to a simple lookup into `utab`$[i][r]$. This is the approach taken in the `pufferfish` index and has proven to be effective for moderately sized indexes. This implementation is output optimal and is fast and cache-friendly since all occ_i occurrences of a tile U_i can be accessed contiguously. However, writing down all start positions of tile-occurrences in `utab` is impractical for large indexes.

For larger indexes (e.g. metagenomic references, many human genomes), explicitly storing `utab` becomes more costly than supporting the k-mer-to-tile query. This is because, as the number of indexed references grow, the number of distinct k-mers grows sub-linearly whereas the number of occurrences grows with the (cumulative) reference length. Problematically, the number of start positions of tile-occurrences grows *at least* linearly. For a reference collection with total sequence length L, a naive encoding for `utab` would take $O(L \lg L)$ bits, as each position require $\lceil \lg L \rceil$ bits and there can be at most L distinct tiles.

Other algorithms that support "locate" queries suffer from a similar problem. To answer queries in time proportional to the number of occurrences of a query, data structures must explicitly store positions of occurrences and access them in constant time. However, storing *all* positions is impractical for large reference texts or large k-mer-sets. To address this, some algorithms employ a scheme to *sample* positions at some small sampling rate s, and perform $O(s)$ work to retrieve not-sampled positions. Since s is usually chosen to be a small constant, this extra $O(s)$ work only imposes a slight overhead.

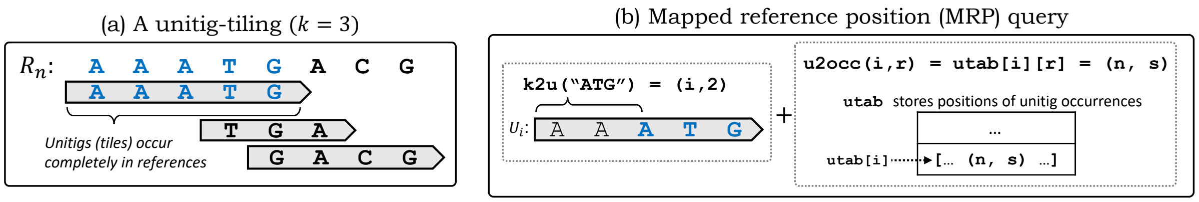
One may wonder if `utab` — which is an *inverted index* — can be compressed using the techniques developed in the Information Retrieval field [204]. For biological sequences, a large proportion of `utab` consists of very short inverted lists (e.g., unique variants in indexed genomes) that are not well-compressible. In fact, these short lists occur at a rate that is much higher than for inverted indexes designed for natural languages. So, instead applying existing compression techniques, we develop a novel *sampling* scheme for `utab` and the tile-to-occurrence query that exploits the properties of genomic sequences.

6.4 Pufferfish2

Below, we introduce `pufferfish2`, an index built over an SPT consisting of *unitigs*. `Pufferfish2` applies a sampling scheme to sparsify the tile-to-occurrence query of a given `pufferfish` index [190].

6.4.1 Interpreting `pufferfish` as an index over a unitig-based SPT

Though not introduced this way by Almodaresi et al., `pufferfish` is an index over a *unitig-tiling* of an input reference collection [190]. A *unitig-tiling* is an SPT which satisfies the property that all tiles always occur completely in references where, for every tile-occurrence $T_{n,m} = U_i$, offset $w_{n,m} = 0$ and length $l_{n,m} = |U_i|$. When this property is satisfied, we term tiles *unitigs*.


(a) A unitig-tiling (k = 3)
Rn: A A A T G A C G
A A A T G
T G A
G A C G
Unitigs (tiles) occur completely in references
(b) Mapped reference position (MRP) query
k2u("ATG") = (i,2)
Ui: A A A T G
+
u2occ(i,r) = utab[i][r] = (n, s)
utab stores positions of unitig occurrences
...
utab[i] [... (n, s) ...]
Figure 6.2: (a) A *unitig-tiling* is an SPT where tiles, *unitigs*, always occur completely in the reference sequences. (b) The MRP query is performed by computing a k-mer's offset into a unitig (`k2u`), then adding the offset to the positions where *unitig-occurrences* appear in indexed reference sequences (`u2occ`). To naively support the unitig-to-occurrence query, positions of all unitig-occurrences are stored in a table, `utab`.

An index built over unitig-tilings does not need to store tile-occurrence offsets, $\mathcal{W}$, or tile-occurrence lengths $\mathcal{L}$ since all tiles have the same offset (zero) and occur with maximal length. For indexes constructed over unitig-tilings, we shall use `k2u` to mean `k2tile`, and `u2occ` to be `tile2occ` with one change. That is, `u2occ` omits offsets and lengths of tile occurrences since they are uninformative for unitig-tilings and returns a tuple (n, s) instead of (n, s, w, l), In prose, we shall refer to these queries as the k-mer-to-unitig and unitig-to-occurrence queries.

The MRP query over unitig-tilings can be computed with Algorithm 4 (in Appendix A.1) where Line 10 is removed from Algorithm 1. We illustrate the MRP query and an example of a unitig-tiling in Fig. 6.2.

6.4.2 Sampling unitigs and traversing tilings to sparsify the unitig-to-occurrence query

`Pufferfish2` implements a sampling scheme for *unitig-occurrences* on a unitig-tiling. For some small constant s, our scheme samples $1/s$ rows in `utab` each corresponding to *all* occurrences of a unique unitig. In doing so, it sparsifies the `u2occ` query and `utab` by only storing positions for a subset of *sampled* unitigs. To compute unitig-to-occurrence queries, it traverses unitig-occurrences

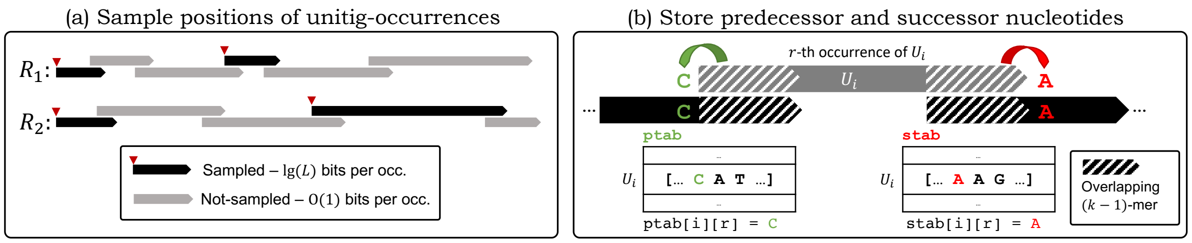

(a) Sample positions of unitig-occurrences
R_1:
R_2:
Sampled – $\lg(L)$ bits per occ.
Not-sampled – $O(1)$ bits per occ.
(b) Store predecessor and successor nucleotides
r-th occurrence of U_i
U_i
C
A
ptab
stab
U_i [... C A T ...]
U_i [... A A G ...]
ptab[i][r] = C
stab[i][r] = A
Overlapping $(k-1)$-mer

Figure 6.3: (a) `Pufferfish2` samples unitigs and their occurrences on a unitig-tiling. Only the positions of the occurrences of the *sampled* unitigs (black) are stored in `utab`. Positions of the *not-sampled* unitigs (gray) can be computed relative to the positions of sampled unitigs by traversing backwards on the visualized tiling of references. Sampling the zero-th unitig-occurrence on every reference sequence guarantees that traversals terminate. (b) Predecessor and successor nucleotides are obtained from adjacent unitig occurrences and are stored in the order in which they appear on the references. These nucleotides for the r-th occurrence of U_i is stored in `ptab`$[i][r]$ and `stab`$[i][r]$, respectively.

on an indexed unitig-tiling.

Notably, `pufferfish2` traverses unitig-tilings that are *implicitly* represented. For unitig-tilings with positions stored in `utab`, there exists no contiguous sequence in memory representing occurrences that is obvious to traverse. However, when viewed as an SPT, *unitig-occurrences* have *ranks* on a tiling and traversals are possible because tiling sequences map uniquely to a sequence of unitig-rank pairs.

Specifically, we define the `pred` query — an atomic traversal step that enables traversals of arbitrary lengths over reference tilings. Given the r-th occurrence of the unitig U_i, the `pred` query returns the identity and rank of the *preceding* unitig. Let tile $T_{n,m}$ be the r-th occurrence of the unitig U_i on all tiling sequences $\mathcal{T}$. Then, `pred`(i, r) returns (j, q) indicating that $T_{n,m-1}$, the *preceding* unitig-occurrence, is the q-th occurrence of the unitig U_j. If there is no preceding occurrence and $m = 1$, `pred`(i, r) returns the sentinel value $\emptyset$.

When an index supports `pred`, it is able to traverse "backwards" on a unitig-tiling. Successively calling `pred` yields the identities of unitigs that form a tiling sequence. Furthermore, since `pred` returns the identity j *and* the rank q of a preceding unitig-occurrence, accessing data associated with each visited occurrence is straightforward in a table like `utab` (i.e., with `utab`$[j][q]$).

Given the unitig-set $\mathcal{U}$, `pufferfish2` first samples a subset of unitigs $\mathcal{U}_S \subseteq \mathcal{U}$. For each sampled unitig $U_i \in \mathcal{U}_S$, it stores information for unitig-occurrences identically to `pufferfish` and records, for *all* occurrences of a sampled unitig U_i, a list of reference identity and position tuples in `utab`$[i]$.

To recover the position of the r-th occurrence a not-sampled unitig U_i and to compute `u2occ`(i, r), the index traverses the unitig-tiling and iteratively calls `pred` until an occurrence of a sampled unitig is found — let this be the q-th occurrence of U_j. During the traversal, `pufferfish2` accumulates number of nucleotides covered by the traversed unitig-occurrences. Since U_j is a sampled unitig, the position of the q-th occurrence can be found in `utab`$[j][q]$. To return `u2occ`(i, r), `pufferfish2` adds the number of nucleotides traversed to the start position stored at `utab`$[j][q]$, the position of a preceding occurrence of the sampled unitig U_j.

This procedure is implemented in Algorithm 2 and visualized in Fig. 6.3. Traversals must account for $(k-1)$ overlapping nucleotides of unitig-occurrences that tile a reference (Line 5). Storing the length of the unitigs is negligible since the number of unique unitigs is much smaller than the number of occurrences.

***.** On the termination of traversals. Any unitig that occurs as the zero-th occurrence (i.e., with rank zero) of a tiling-sequence is always sampled. This way, backwards traversals terminate because every occurrence of a not-sampled unitig occurs after a sampled unitig. This can be seen from Fig. 6.3. Concretely, if $T_{n,1} = U_i$ for some tiling-sequence T_n, then the unitig U_i must always be sampled.

Algorithm 2:

```
def u2occ($i, r$):
    $l \leftarrow 0$
    while $!isSamp[i]$ do
        $(i, r) = \texttt{pred}(i, r)$
        $l \leftarrow l + |U_i| - k + 1$
    $(n, s) \leftarrow \texttt{utab}[i][r]$
    return $(n, s + l)$
```

Algorithm 3:

```
def pred($i, r$):
    $p \leftarrow \texttt{ptab}[i][r]$
    $y \leftarrow p \circ U_i[: k - 1]$
    $(j, \_) \leftarrow \texttt{k2u}(y)$
    $s \leftarrow U_i[k]$
    $t \leftarrow \texttt{rank}_p(\texttt{ptab}[i], r)$
    $q \leftarrow \texttt{select}_s(\texttt{stab}[j], t)$
    return $(j, q)$
```

6.4.3 Implementing the `pred` query with `pufferfish2`

`Pufferfish2` computes the `pred` query in constant time while requiring only constant space per unitig-occurrence by carefully storing *predecessor* and *successor* nucleotides of unitig-occurrences.

***.** Predecessor and successor nucleotides. Given the tiling sequence $T_n = \left[T_{n,1}, \ldots, T_{n,M_n}\right]$, we say that a unitig-occurrence $T_{n,m}$ is *preceded* by $T_{n,m-1}$, and that $T_{n,m-1}$ is *succeeded* by $T_{n,m}$. Suppose $T_{n,m} = U_i$, and $T_{n,m-1} = U_j$, and let the unitigs have lengths ℓ_i and ℓ_j, respectively.

We say that, $T_{n,m-1}$ precedes $T_{n,m}$ with predecessor nucleotide p. The predecessor nucleotide is the nucleotide that precedes the unitig-occurrence $T_{n,m}$ on the reference sequence R_n. Concretely, p is the first nucleotide on the last k-mer of the preceding unitig, i.e., $p = T_{n,m-1}[\ell_j - k]$. We say that, $T_{n,m}$ succeeds $T_{n,m-1}$ with successor nucleotide s. Accordingly, the successor nucleotide, s, is the last nucleotide on the first k-mer of the succeeding unitig, i.e., $s = T_{n,m}[k]$.

Abstractly, the preceding occurrence $T_{n,m-1}$ can be "reached" from the succeeding occurrence

$T_{n,m}$ by prepending its predecessor nucleotide to the $(k-1)$-length prefix of $T_{n,m}$. Given $T_{n,m}$ and its predecessor nucleotide p, the k-mer y that is the last k-mer on the preceding occurrence $T_{n,m-1}$ can be obtained with $y = p \circ T_{n,m}[: k-1]$. Given an occurrence $T_{n,m}$, let the functions $\mathrm{pred}_{\mathrm{n}}\left(T_{n,m}\right)$ and $\mathrm{succ}_{\mathrm{n}}\left(T_{n,m}\right)$ yield the predecessor nucleotide and the successor nucleotide of $T_{n,m}$, respectively. If $T_{n,m}$ is the first or last unitig-occurrence pair on T_n, then $\mathrm{succ}_{\mathrm{n}}\left(T_{n,m}\right)$ and $\mathrm{pred}_{\mathrm{n}}\left(T_{n,m}\right)$ return the "null" character, "\$".

These notationally dense definitions can be more easily understood with a figure. Figure 6.3 shows how predecessor and successor nucleotides of a given unitig-occurrence on a tiling are obtained.

***.** Concrete representation. `Pufferfish2` first samples a set of unitigs $\mathcal{U}_S \subseteq \mathcal{U}$ from $\mathcal{U}$ and stores a bit vector, `isSamp`, to record if a unitig U_i is sampled where $\texttt{isSamp}[i] = 1$ iff $U_i \in \mathcal{U}_S$. `Pufferfish2` stores in `utab` the reference identity and position pairs for occurrences of *sampled* unitigs only.

After sampling unique unitigs, `pufferfish2` stores a *predecessor nucleotide table*, `ptab`, and a *successor nucleotide table*, `stab`. For each not-sampled unitig U_i *only*, $\texttt{ptab}[i]$ stores a list of predecessor nucleotides for each occurrence of U_i in the unitig-tiling. For *all* unitigs U_i, $\texttt{stab}[i]$ stores a list of successor nucleotides for each occurrence of U_i. Concretely, when the unitig-occurrence $T_{n,m}$ is the r-th occurrence of U_i,

$$\texttt{ptab}[i][r] = \mathrm{pred}_{\mathrm{n}}\left(T_{n,m}\right) \quad \text{and} \quad \texttt{stab}[i][r] = \mathrm{succ}_{\mathrm{n}}\left(T_{n,m}\right).$$

As discussed in Section 6.4.2, unitigs that occur as the zero-th element on a tiling is always sampled so that every occurrence of a not-sampled unitig has a predecessor. If $T_{n,m}$ has no successor and is the last unitig-occurrence on a tiling sequence, $\texttt{stab}[i][j]$ contains the sentinel symbol "\$".

Figure 6.3 illustrates how predecessor and successor nucleotides are stored.

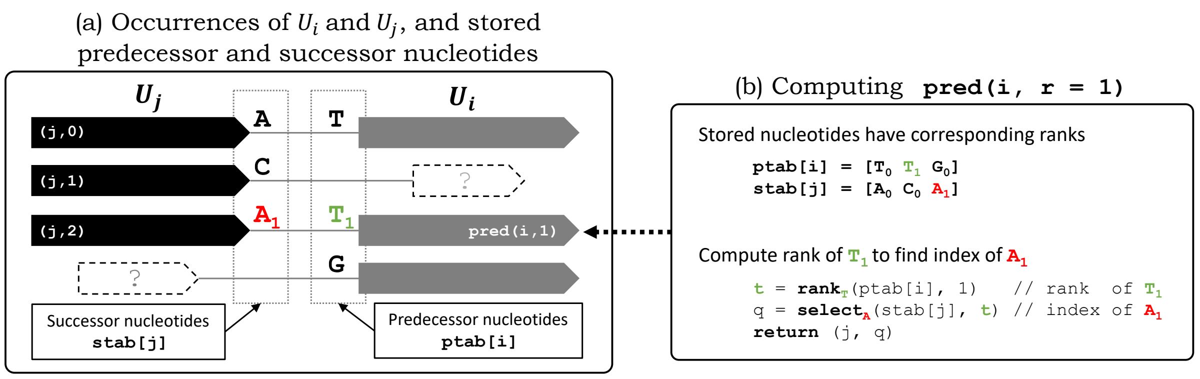
*. Computing the pred query.

Given the k-mer-to-unitig query, pufferfish2 supports the pred query for any unitig U_i that is not-sampled. When the r-th occurrence of U_i succeeds the q-th occurrence of U_j, it computes $\texttt{pred}(i, r) = (j, q)$ with Algorithm 3. To compute pred, it constructs a k-mer to find U_j, and then computes one rank and one select query over the stored lists of nucleotides to find the correct occurrence.

Pufferfish2 first computes j, the identity of the preceding unitig. The last k-mer on the preceding unitig must be the first $(k-1)$-mer of U_i *prepended* with predecessor nucleotide of the r-th occurrence of U_i. Given $\texttt{ptab}[i][r] = p$, it constructs the k-mer, $y = p \circ U_i[: k-1]$, that must be the last k-mer on U_j. So on Line 4, it computes $\texttt{k2u}(y)$ to obtain the identity of the preceding unitig U_j.

It then computes the unitig-rank, q, of the preceding unitig-occurrence of U_j. Each time U_i is preceded by the nucleotide p, it must be preceded by the *same* unitig U_j since any k-mer occurs in only one unitig. Accordingly, each occurrence U_j that is succeeded by U_i must always be succeeded by the *same* nucleotide s equal to the k-th nucleotide of U_i, $U_i[k]$. For the preceding occurrence of U_j that the algorithm seeks to find, the nucleotide s is stored at some unknown index q in $\texttt{stab}[j]$ — the list of successor nucleotides of U_j.

Whenever an occurrence of U_i succeeds an occurrence of U_j, so do the corresponding pair predecessor and successor nucleotides stored in $\texttt{ptab}[i]$ and $\texttt{stab}[j]$. Since $\texttt{ptab}[i]$ and $\texttt{stab}[j]$ store predecessor and successor nucleotides in the order in which unitig-occurrences appear in the tiling sequences, the following *ranks* of stored *nucleotides* must be equal: (1) the rank of the nucleotide $p = \texttt{ptab}[i][r]$ at index r in the list of predecessor nucleotides, $\texttt{ptab}[i]$, of the succeeding unitig U_i,


(a) Occurrences of U_i and U_j, and stored predecessor and successor nucleotides
U_j
U_i
(j,0)
(j,1)
(j,2)
A
C
A_1
T
T_1
G
?
pred(i,1)
Successor nucleotides stab[j]
Predecessor nucleotides ptab[i]
(b) Computing pred(i, r = 1)
Stored nucleotides have corresponding ranks
Compute rank of T_1 to find index of A_1

```
ptab[i] = [T0 T1 G0]
stab[j] = [A0 C0 A1]

t = rankT(ptab[i], 1)   // rank  of T1
q = selectA(stab[j], t) // index of A1
return (j, q)
```

Figure 6.4: Visualizing the `pred` query that finds the occurrence of U_j that precedes the queried occurrence of U_i with rank 1. (a) All occurrences of U_i and U_j are visualized (in sorted order) with their preceding and succeeding unitig occurrences, respectively. The figure shows stored successor nucleotides for U_j, and predecessor nucleotides for U_i. Whenever an occurrence of U_j precedes an occurrence of U_i, a corresponding pair of nucleotides "`A`" and "`T`" occur and are stored in `stab`$[j]$ and `ptab`$[i]$ respectively. (b) Their *ranks* (annotated with subscripts) of the corresponding predecessor-successor nucleotide pair *match* in `ptab`$[i]$ and `stab`$[j]$, but the *indices* do not. A rank query for predecessor nucleotide "`T`" at index $r = 1$ yields the matching rank of the successor nucleotide "`A`". A select query for the nucleotide "`A`" with rank 1 yields the *index* and occurrence of the predecessor U_j.

and (2) the rank of the nucleotide $s = U_i[k]$ at index q in the list of successor nucleotides, `stab`$[j]$, of the preceding unitig U_j. We illustrate this correspondence between ranks in Fig. 6.4. So to find q, the rank of the preceding unitig-occurrence, `pufferfish2` computes the rank of the predecessor nucleotide, $t = \texttt{rank}_p(\texttt{ptab}[i], r)$. Then, computing $\texttt{select}_s(\texttt{stab}[i], t)$, the index where the t-th rank successor nucleotide of U_j occurs must yield q.

*. Time and space analysis.

`Pufferfish2` computes the `pred` query in constant time. The k-mer for the query `k2u` is assembled in constant time, and the `k2u` query itself is answered in constant time, as already done in the `pufferfish` index [190].

For not-sampled unitigs, `pufferfish2` does not store positions of unitig-occurrences in `utab`. Instead, it stores nucleotides in tables `stab` and `ptab`. These tables are implemented by *wavelet matrices* that support rank, select, and access operations in $O(\lg \sigma)$ time on sequences with alphabet size σ while requiring only $\lg \sigma + o(\lg \sigma)$ bits per element [202].

As explained in Section 6.3.1, we have avoided the treatment of *orientations* of nucleotide sequences for brevity. In actuality, unitigs may occur in a *forward* or a *backwards* orientation (i.e., with a reverse complement sequence). When considering orientations, pufferfish2 implements the pred query by storing and querying over lists of *nucleotide-orientation* pairs. In this case, ptab and stab instead store predecessor-orientation and successor-orientation pairs. Accordingly, wavelet matrices are then built over alphabets of size 8 and 9 respectively — deriving from eight nucleotide-orientation pairs and one sentinel value for unitig-occurrences that have no predecessor. Thus, ptab and stab in total require ≈ 7 bits per unitig-occurrence (since $7 = \lceil \lg 8 \rceil + \lceil \lg 9 \rceil$). We describe how the pred query is implemented with orientations in Appendix A.3.

***.** Construction. The current implementation of pufferfish2 sparsifies the unitig-to-occurrence query and compresses the table of unitig occurrences, utab, of an existing pufferfish index, and inherits its k-mer-to-unitig mapping. In practice, sampling and building a pufferfish2 index always takes less time than the initial pufferfish index construction. In brief, building pufferfish2 amounts to a linear scan over an SPT. We describe how pufferfish2 in constructed in more detail in Appendix A.4.

6.4.4 A random sampling scheme to guarantee short backwards traversals

Even with a constant-time pred query, computing the unitig-to-occurrence query is fast only if the length of backwards traversals — the number of times pred is called — is small. So for some small constant s, a sampling scheme should sample $1/s$ of *unique* unitigs, store positions of only $1/s$ of unitig-*occurrences* in utab, and result in traversal lengths usually of length s.

At first, one may think that a greedy sampling scheme that traverses tiling sequences to sample unitigs could be used to bound traversal lengths to some given maximum length, s. However, when tiling sequences become much longer than the number of unique unitigs, such a greedy scheme samples almost *all* unitigs and only somewhat effective in limited scenarios (see Appendix A.5). Thus, we introduce the *random* sampling scheme that samples $1/s$ of unitigs uniformly at random from $\mathcal{U}$. This scheme guarantees that traversals using the `pred` query terminate in s steps *in expectation* if each unitig-occurrence $T_{n,m}$ is independent and identically distributed and drawn from an arbitrary distribution. Then, backwards traversals until the occurrence of a sampled unitig is a series of Bernoulli trials with probability $1/s$, and traversal lengths follow a geometric distribution with mean s. Although this property relies on a simplifying assumption, the random sampling scheme works well in practice.

6.4.5 Closing the gap between a constant time `pred` query and contiguous array access

Even though the `pred` query is constant time and traversals are short, it is difficult to implement `pred` queries in with speed comparable to *contiguous array accesses* that are used to compute the `u2occ` for when `utab` is "dense" — i.e., uncompressed and not sampled. In fact, any compression scheme for `utab` would have difficulty contending with constant time contiguous array access regardless of their asymptotics since dense implementations are output optimal, very cache friendly, and simply store the answers to queries in an array. To close the gap between theory and practice, `pufferfish2` exploits several optimizations.

In practice, a small proportion of unique unitigs are "popular" and occur extremely frequently. Fortunately, the total number of occurrences of popular unitigs is small relative to other unitigs. To avoid an excessively large number of traversals from a not-sampled unitig, `pufferfish2` modifies the sampling scheme to always sample popular unitigs that occur more than a preset number, α, times. Better yet, we re-parameterize this optimization and set α so that the total number of occurrences of popular unitigs sum to a given proportion $0 < t \leq 1$ of the total occurrences of all the unitigs. For example, setting $t = 0.25$ restricts `pufferfish2` to sample from 75% of the total size of `utab` consisting of unitigs that occur most infrequently.

Also, the MRP and `pred` query are especially amenable to caching. Notably, `pufferfish2` caches and memoizes redundant `k2u` queries in successive `pred` queries. Also, it caches "streaming" queries to exploit the fact that successive queried k-mers (e.g., from the same sequenced read) likely land on the same unitig. We describe in more detail these and other important optimizations in Appendix A.6.

6.5 Experiments

We assessed the space-usage of the indexes constructed by `pufferfish2` from several different whole-genome sequence collections, as well as its query performance with different sampling schemes. Reported experiments were performed on a server with an Intel Xeon CPU (E5-2699 v4) with 44 cores and clocked at 2.20 GHz, 512 GB of memory, and a 3.6 TB Toshiba MG03ACA4 HDD.

***.** Datasets. We evaluated the performances on a number of datasets with varying attributes: (1) Bacterial collection: a random set of 4000 bacterial genomes from the NCBI microbial database; (2) Human collection: 7 assembled human genome sequences from [205]; and (3) Metagenomic collection: 30,691 representative sequences from the most prevalent human gut prokaryotic genomes

Dataset	**Sampling strategy**	`u2occ` **size (GB)**	**10M k-mers (secs)**	**100K reads (secs)**
7 Humans	None	16.8	86.1	139.4
	Random $(s = 3, t = .05)$	7.8 (0.46)	4159.1 (43.8×)	8092.8 (58.04×)
	Random $(s = 3, t = .25)$	9.9 (0.59)	681.1 (7.9×)	1466.2 (10.52×)
4000 Bacteria	None	7.7	35.5	12.6
	Random $(s = 3, t = .05)$	3.7 (0.48)	420.4 (11.9×)	15.6 (1.24×)
	Random $(s = 3, t = .25)$	4.7 (0.61)	323.8 (9.1×)	15.5 (1.23×)
30K Human gut	None	86.3	80.6	178.7
	Random $(s = 3, t = .05)$	45.6 (0.53)	439.4 (5.5×)	570.2 (3.19×)
	Random $(s = 3, t = .25)$	54.4 (0.63)	365.2 (4.5×)	576.9 (3.23×)
	Random $(s = 6, t = .05)$	34.6 (0.40)	1037.5 (12.9×)	644.8 (3.61×)
	Random $(s = 6, t = .25)$	45.6 (0.53)	614.0 (7.6×)	646.1 (3.56×)

Table 6.1: Size and speed of `pufferfish2` indexes querying 10 million random k-mers and 100,000 reads. Uncompressed, baseline implementations of the unitig-to-occurrence query (`pufferfish` indexes with the *sparse* `k2u` implementation [190]) are labeled with "None" sampling strategy. Relative sizes of compressed representations and relative slowdowns to the baseline are indicated in parentheses.
from [206].

***.** Results.

To emulate a difficult query workload, we queried the indexes with 10 million random *true positive* k-mers sampled uniformly from the indexed references. Our results from Table 6.1 show that sampling *popular* unitigs is critical to achieve reasonable trade-offs between space and speed. When indexing seven human genomes, the difference in space between always sampling using $t = 0.05$ and $t = 0.25$, is only 2.1GB (12.5% of the uncompressed `utab`). However, explicitly recording 2.1GB of positions of occurrences of popular unitigs, *substantially* reduces the comparative slowdown from 43.8× to 7.9×. This is because setting $t = 0.25$ instead of $t = 0.05$ greatly reduces the maximum number of occurrences of a *not-sampled* unitig — from ≈87,000 to ≈9,000 times, respectively. Here, setting $t = 0.25$ means that random k-mer queries that land in not-sampled unitigs perform many fewer traversals over reference tilings.

On metagenomic datasets, indexes are compressed to a similar degree but differences in query speed at different parameter settings are small. `Pufferfish2` is especially effective for a *large* collection of bacterial genomes. With the fastest parameter setting, it incurs only a $4.5\times$ slowdown for random queries while reducing the size of `utab` for the collection of 30,000 bacterial genomes by 37% (from 86.3GB to 54.4GB).

Apart from random lookup queries, we also queried the indexes with k-mers deriving from sequenced readsets [207, 208]. We measured the time to query and recover the positions of all k-mers on 100,000 reads. This experiment demonstrates how the slowdown incurred from sampling can (in most cases) be further reduced when queries are positionally coherent or miss. Successive k-mer queries from the same read often land on the same unitig and can thus be cached (see Section 6.4.5). *True negative* k-mers that do not occur in the indexed reference collection neither require traversals nor incur any slowdowns.

To simulate a metagenomic analysis, we queried reads from a human stool sample against 4,000 bacterial genomes. This is an example of a low hit-rate analysis where 18% of queried k-mers map to indexed references. In this scenario, `pufferfish2` reduces the size of `utab` by *half* but incurs only a $1.2\times$ slowdown. We also queried reads from the same human stool sample against the collection of 30,000 bacterial genomes representative of the human gut. Here, 88% of k-mers are found in the indexed references. At the sparsest setting, `pufferfish2` indexes incur only a $3.6\times$ slowdown while reducing the size of `utab` by 60%.

We observe that `pufferfish2`'s sampling scheme is less effective when indexing a collection of seven human genomes. When sampled with $s = 3$ and $t = 0.25$, `pufferfish2` incurs a $10.5\times$ slowdown when querying reads from a DNA-seq experiment in which 92% of queried k-mers occur in reference sequences. Interestingly, the slowdown when querying reads is larger than the slowdown when querying random k-mers. This is likely due to biases from sequencing that cause k-mers and reads to map to non-uniformly indexed references. Nonetheless, this result motivates future work that could design sampling schemes optimized for specific distributions of query patterns.

We expect to see less-pronounced slowdowns in practice than those reported in Table 6.1. This is because tools downstream of an index like `pufferfish2` almost always perform operations *much* slower after straightforward exact lookups for k-mers. For example, aligners have to perform alignment accounting for mismatches and edits. Also, our experiments pre-process random k-mer sets and read-sets so that no benchmark is I/O bound. Critically, the compromises in speed that `pufferfish2` makes are especially palatable because it trades-off speed in the *fastest* operations in analyses — *exact* k-mer queries — while substantially reducing the space required for the *most space intensive* operation.

***.** Using `SSHash` for even smaller indexes.

For convenience, we have implemented our SPT compression scheme within an index that uses the *specific* sparse `pufferfish` implementation for the k-mer-to-tile (k-mer-to-unitig) mapping [190]. However, the SPT enables the construction of modular indexes that use *various* data structures for the k-mer-to-tile mapping and the tile-to-reference mapping, provided only a minimalistic API between them. A recent representation of the k-mer-to-tile mapping that supports all the necessary functionality is `SSHash` [23]. Compared to the `k2u` component of `pufferfish`, `SSHash` is almost always substantially smaller. Further, it usually provides faster query speed compared to the *sparse* `pufferfish` implementation of the k-mer-to-tile query, especially when streaming queries are being performed.

In Table 6.2, we calculate the size of indexes if `SSHash` is used for the k-mer-to-tile mapping — rather than the *sparse* `pufferfish` implementation. These sizes then represent overall index sizes that would be obtained by pairing a state-of-the-art representation of the k-mer-to-tile mapping with a state-of-the-art representation of the tile-to-reference mapping (that we have presented in this work). Practically, the only impediment to constructing a fully-functional index from these components is that they are implemented in different languages (`C++` for `SSHash` and `Rust` for `pufferfish2`) — we are currently addressing this issue.

Importantly, these results demonstrate that, when `SSHash` is used, the representation of the tile-to-occurrence query becomes a bottleneck in terms of space, occupying an increasingly larger fraction of the overall index. Table 6.2 shows that, in theory, if one fully exploits the modularity of SPTs, new indexes that combine `SSHash` with `pufferfish2` would be *half* the space of the original `pufferfish` index. As of writing, with respect to an index over 30,000 bacterial genomes, the estimated difference in *monetary* cost of an AWS EC2 instance that can fit a new 55.6GB index versus a 131GB `pufferfish` index in memory is 300USD per month (see Appendix A.7).

***.** Comparing to MONI and the r-index. We compared `pufferfish2` to MONI, a tool that builds an r-index to locate maximal exact matches in highly repetitive reference collections [187]. In brief, `pufferfish2` is faster and requires less space than MONI for our benchmarked bacterial dataset. Our tool does so with some trade-offs. `Pufferfish2` supports rapid locate queries for k-mers of a *fixed* length, while r-index based approaches supports locate queries for patterns of any *arbitrary* length and can be used to find MEMs. Notably, it has been shown that both k-mer and MEM queries can be used for highly effective read-mapping and alignment [183, 187].

For reference, we built MONI on our collection of 4,000 bacterial genomes. Here, MONI required 51.0G of disk space to store which is 29% larger than the `pufferfish` index (39.5GB) with its *dense* `k2u` implementation — its *least* space-efficient configuration. The most space efficient configuration of the `pufferfish2` index (with s=3, t=.25) is 42% the size of MONI when built on from the same data and requires 21.7GB of space. Compared to a theoretically possible index

Dataset	**`u2occ` w/ `pufferfish2`**	**`k2u` w/ `SSHash`**	**New index**	**`pufferfish` index**
7 Human	9.9	3.2	**13.1**	28.0
4000 Bacteria	3.7	7.3	**11.0**	26.1
30K Human gut	34.6	22.0	**55.6**	131.7

Table 6.2: Sizes in GB of possible, new indexes — with `k2u` implemented by `SSHash` and `u2occ` by `pufferfish2` — compared to the size of original `pufferfish` indexes. Selected sampling parameters for datasets (top-to-bottom) are $(s = 3, t = 0.25)$, $(s = 3, t = 0.05)$, and $(s = 6, t = 0.05)$, respectively.

specified in Table 6.2 that would only require 11.0GB, MONI would need 4.6× more space.

We also performed a best-effort comparison of query speed between `pufferfish2` and MONI. Unfortunately, it is not possible to directly measure the speed of exact locate queries for MONI because it does not expose an interface for such queries. Instead, we queried MONI to find MEMs on true-positive k-mers treating each k-mer as unique read (encoded in FASTQ format as MONI requires). We argue that this is a reasonable proxy to exact locate queries because, for each true-positive k-mer deriving from an indexed reference sequence, the entire k-mer itself is the maximal exact match. For MONI, just like in benchmarks for in Table 6.1, we report the time taken for computing queries only and ignore time required for I/O operations (i.e., loading the index and quries, and writing results to disk).

We found that `pufferfish2` is faster than MONI when querying k-mers against our collection of 4,000 bacterial genomes. MONI required 1,481.7 seconds to query the same set of 10 million random true-positive k-mers queried in Table 6.1. When compared to the slowest built most space efficient configuration of `pufferfish2` benchmarked in Table 6.1, `pufferfish2` is 3.5× faster.

6.6 Discussion

In this work, we introduce the *spectrum preserving tiling* (SPT), which describes how a spectrum preserving string set (SPSS) tiles and "spells" an input collection of reference sequences. While considerable research effort has been dedicated to constructing space and time-efficient indexes for SPSS, little work has been done to develop efficient representations of the tilings themselves, despite the fact that these tilings tend to grow more quickly than the SPSS and quickly become the size bottleneck when these components are combined into reference indexes. We describe and implement a sparsification scheme in which the space required for representing an SPT can be greatly reduced in exchange for an expected constant-factor increase in the query time. We also describe several important heuristics that are used to substantially lessen this constant-factor in practice. Having demonstrated that modular reference indexes can be constructed by composing a k-mer-to-tile mapping with a tile-to-occurrence mapping, we have thus opened the door to exploring an increasingly diverse collection of related reference indexing data structures.

6.7 Future work

Despite the encouraging progress that has been made here, we believe that there is much left to be explored regarding the representation of SPTs, and that many interesting questions remain. Some of these questions are:

1. How would an algorithm sample individual unitig-occurrences instead of all occurrences of a unitig to *explicitly* bound the lengths of backwards traversals? Given the SPT definition, we have introduced strategies for sampling $\mathcal{U}$. That is, either a tile has *all* or *none* of its

occurrences sampled. Yet, nothing theoretically prevents one from instead sampling over $\mathcal{T}$, so that *occurrences* are sampled according to their position on a tiling regardless of their unitig-identities. This approach introduces some extra complications but provides the benefit of allowing sampling schemes to *trivially* bound the worst-case traversal length, while also directly controlling the fraction of sampled entries by sampling every s-th occurrence. The question of what sampling strategy works better in practice is an interesting open question.

2. Does a smaller SPSS imply a small SPT, and could one compute an optimally small SPT? Currently, this is not clear, since working with unitigs dispenses entirely the space required for $\mathcal{W}$, $\mathcal{L}$, and $\mathcal{S}$, so that a smaller SPSS may increase the space for representing the tiling given the need to encode $\mathcal{W}$, $\mathcal{L}$, and $\mathcal{S}$.

3. Given some distributional assumptions, can an algorithm sample SPTs to minimize the expected query time or to minimize index sizes? We have provided an intuitive but imprecise notion of what a "good" or "desirable" SPT ought to be. That is, an SPT amenable to indexing has few but long tiles and short tilings. Yet, rather than separating the problem of finding a set of tiles and then efficiently representing the tiling it induces, one could more precisely formulate a general optimization problem given a set of references $\mathcal{R}$.

4. In practice, how can an implemented tool combine our sampling scheme with existing compression algorithms for the highly skewed tile-to-occurrence query? We have implemented one, specific, sparsification and compression scheme to reduce the size of SPTs. However, as hybrid encoding strategies have proven successful in optimizing the representation of k-mer-to-tile mappings [23], we may expect the same to be true of the tile-to-occurrence map. For example, long occurrence lists may compress well with traditional information retrieval

compression schemes [209], and delta-encoding-like schemes may prove very effective in compressing the occurrence lists for tiles that almost always co-occur. In general, hybrid encoding and compression schemes likely hold great promise in tackling this problem.

5. Can a *lossy* index over an SPT be constructed and applied effectively in practical use cases? We have considered here only exact and lossless representation of SPTs. However, many successful indexing schemes for problems like read mapping avoid indexing all sub-words, instead, for example, indexing only minimizers [210] or altering the sampling strategy in highly-repetitive regions. Thus, for many important applications it may not be necessary to have a *complete* and *lossless* index over the underlying SPT and it is possible that a *lossy* index over an SPT could be made much smaller and faster still.

Chapter 7: Fulgor: A fast and compact k-mer index for large-scale matching and color queries

Disclosure

This Chapter presents first-author work presented at, and preprinted for,the 2023 Workshop on Algorithms in Bioinformatics [211], with minimal changes.

7.1 Background and motivation

At the core of many metagenomic and pan-genomic analyses is *read-mapping*, the atomic operation that assigns observed sequence reads to putative genome(s) of origin. A wide range of methods have been developed for mapping reads to large collections of reference genomes. Of note, alignment-based methods, though accurate [212, 213], are relatively computationally intensive as they must provide the ability to *locate* the read on each genome. A queried read must, with low edit-distance, be matched with a sub-string of some reference genome in the collection. For alignment, the index is also required to report the position of this match. As a matter of fact, alignment against hundreds or even tens of thousands of reference genomes can be impractically slow and simply require too much space in practice.

Fortunately, *alignment-free* techniques have become popular and widespread for metagenomic analyses [214, 215, 216, 217, 218]. These methods generally work by avoiding alignment altogether, and replacing it with strategies for matching (exactly or approximately) substrings, signatures, or sketches between the queries and the referenced sequences. Ideally, good matching heuristics can assign or match a query against the correct reference with high precision while also retaining high recall (i.e., being sensitive to sequencing error or small divergence between the query and the reference). One particular type of alignment-free method for assigning reads to compatible references that has recently gained substantial traction is *pseudoalignment* [184, 219, 220, 221]. While tremendous progress has been made in supporting alignment-free methods for metagenomic analyses, continued development of ever more efficient indexing methods is required for such analyses to scale to tens, even hundreds, of thousands of bacterial reference genomes.

A practical data structure that is suitable for alignment-free matching methods is the *colored de Bruijn graph*, a graph where each node corresponds to a k-mer in a reference collection and is annotated with a *color*, the set of references in which it occurs. Bifrost [222] and Metagraph [223] are two efficient approaches that index the colored de Bruijn graph and support the k-mer-to-color query. Recently, Alanko et al. [224] developed Themisto, an index for alignment-free matching (and specifically pseudoalignment) that substantially outperforms these prior methods in the context of indexing and mapping against large collections of genomes. Compared to Bifrost, Themisto uses practically the same space, but is faster to build and query. Compared to the fastest variant of Metagraph, Themisto offers similar query performance, but is much more space-efficient; on the other hand, Themisto is much faster to query than Metagraph-BRWT, the most-space efficient variant of Metagraph.

7.1.1 Contributions

We describe how recent advancements in associative, order-preserving, k-mer dictionaries [23, 200] can be combined with a compressed inverted index to implement a fast index over the *colored compacted* de Bruijn graph (ccdBG). Leveraging the *order-preserving* property of its dictionary, our index takes full advantage of the fact that unitigs in this variant of the ccdBG are *monochromatic* — i.e., all k-mers in a unitig have the same set of references of origin, or "colors". In fact, k-mers are kept in unitig order, and our index takes advantage of the ability of our associative dictionary to store the unitigs in any order. Reordering the unitigs so that all unitigs with the same color are adjacent in the index allows the construction of a map from k-mers to their corresponding colors that uses only $1 + o(1)$ bits per unitig. Our index combines this property with a simple but effective hybrid compression scheme for inverted lists (colors) to require little space. By storing unitigs and keeping k-mers in unitig order, our index also supports very fast streaming queries for consecutive k-mers in a read, and additionally allows efficient implementation of skipping heuristics that have previously been suggested to speed up pseudoalignment [184]. We implemented our index in a C++17 tool called `fulgor`, which is available at `https://github.com/jermp/fulgor`.

Compared to Themisto [224], the prior state of the art, `fulgor` indexes a heterogeneous collection of 30,691 bacterial genomes in $3.8\times$ less space, a collection of 150,000 *Salmonella enterica* genomes in approximately $2\times$ less space, is at least twice as fast at query time, and even $2-6\times$ faster to construct.

Perhaps unsurprisingly, the rapid development of novel indexing data structures has been accompanied by novel and custom strategies for matching and assigning reads to colors (i.e., reference sets) and algorithms that each make different design choices and trade-offs. Many of these strategies can be considered as a form of pseudoalignment. Having been iterated on since its introduction [184], the term “pseudoalignment” has come to describe a family of efficient heuristics for read-to-color assignment, rather than a single concept or algorithm. Prior methods have taken either *exhaustive* approaches that queries every k-mer on a read (previously termed *exact* pseudoalignment [221, 224]) or have implemented *skipping* based approaches that skip the query of “redundant” consecutive k-mers that likely map to the same set of reference genomes [184, 225]. To our knowledge, the precise details of the types of skipping heuristics used in the latter methods — including those adopted by the initial pseudoalignment method — have been discussed only in passing. Complete details, instead exist only in the source code of the corresponding tools. To shed light on these algorithms, we provide a more structured discussion of how these algorithms are designed. Using `fulgor`, we implement two previously proposed variants and benchmark them.

7.2 Preliminaries

In this section, we first formalize the problem under study here. We then describe a modular indexing layout that solves the problem using the interplay between two well-defined data structures. Lastly we describe the properties induced by the problem and how these are elegantly captured by the notion of *colored compacted de Bruijn graph*.

7.2.1 Problem definition

Problem 7.1 (Colored k-mer indexing problem)**.** *Let $\mathcal{R} = \{R_1, \dots, R_N\}$ be a collection of references. Each reference R_i is a string over the DNA alphabet $\Sigma = \{A, C, G, T\}$. We want to build a data structure (referred to as the* index*) that allows us to retrieve the set* $\mathsf{Color}(x) = \{i | x \in R_i\}$

as efficiently as possible for any k-mer $x \in \Sigma^k$. Note that $\mathsf{Color}(x) = \emptyset$ *if x does not occur in any reference.*

Hence, we call the set $\mathsf{Color}(x)$ the *color* of the k-mer x.

7.2.2 Modular indexing layout

In principle, Problem 7.1 could be solved using an old but elegant data structure: the *inverted index* [204, 226]. The inverted index, say $\mathcal{L}$, stores explicitly the ordered set $\mathsf{Color}(x)$ for each k-mer $x \in \mathcal{R}$. What we want is to implement the map $x \to \mathsf{Color}(x)$ as efficiently as possible in terms of both memory usage and query time. To this end, all the distinct k-mers of $\mathcal{R}$ are stored in an *associative* dictionary data structure, $\mathcal{D}$. Suppose the dictionary $\mathcal{D}$ stores n k-mers. To implement the map $x \to \mathsf{Color}(x)$, the operation that $\mathcal{D}$ is required to support is $\mathsf{Lookup}(x)$ which returns $\perp$ if k-mer x is not found in the dictionary or a unique integer identifier in $[n] = \{1, \ldots, n\}$ if x is found. Problem 7.1 can then be solved using these two data structures — $\mathcal{D}$ and $\mathcal{L}$ — thanks to the interplay between $\mathsf{Lookup}(x)$ and $\mathsf{Color}(x)$: logically, the index stores the sets $\{\mathsf{Color}(x)\}_{x \in \mathcal{R}}$ in compressed format in the order given by $\mathsf{Lookup}(x)$.

To our knowledge, all prior solutions proposed in the literature that fall under the "color-aggregative" classification [227], are incarnations of this *modular indexing framework* and, as such, require an efficient k-mer dictionary joint with a compressed inverted index. For example, Themisto [224] makes use of the *spectral* BWT (or SBWT) data structure [201] for its k-mer dictionary, whereas Metagraph [223] implements a general scheme to compress metadata associated to k-mers which is, in essence, an inverted index.

7.2.3 The colored compacted de Bruijn graph and its properties

Problem 7.1 has some specific properties that one would like to exploit to implement as efficiently as possible the modular indexing framework described in Section 7.2.2. First, consecutive k-mers share $(k-1)$-length overlaps; second, co-occurring k-mers have the same color. A useful, standard, formalism that describes these properties is the *colored compacted de Bruijn graph* (ccDBG).

Given the collection of references $\mathcal{R}$, the (node-centric) de Bruijn graph (dBG) of $\mathcal{R}$ is a directed graph whose nodes are all the distinct k-mers of $\mathcal{R}$ and there is an edge connecting node u to node v if the $(k-1)$-length suffix of u is equal to the $(k-1)$-length prefix of v. We refer to k-mers and nodes in a (node-centric) dBG interchangeably; likewise, a path in a dBG spells the string obtained by "gluing" together all the k-mers along the path. Thus, unary (i.e., non-branching) paths in the graph can be collapsed into single nodes spelling strings that are referred to as *unitigs*. The dBG arising from this compaction step is called the compacted dBG (cdBG). Lastly, the *colored* compacted dBG is obtained by logically annotating each k-mer x with its color, $\mathsf{Color}(x)$, and only collapsing non-branching paths with nodes having the same color.

Below, we notate n to be the number of distinct k-mers of $\mathcal{R}$ and m to be the number of unitigs $\{u_1, \ldots, u_m\}$ of the ccdBG induced by the k-mers of $\mathcal{R}$. The unitigs of the ccdBG that we consider have the following key properties.

1. *Unitigs are contiguous subsequences that spell references in $\mathcal{R}$.* Each distinct k-mer of $\mathcal{R}$ appears once, as sub-string of some unitig of the cdBG. By construction, each reference $R_i \in \mathcal{R}$ can be a *tiling* of the unitigs — a sequence of unitig occurrences that spell out R_i [182]. Joining together k-mers into unitigs reduces their storage requirements. In Sections 7.3.1 and 7.3.2, we show how this property can be exploited to make indexes compact.

In Section 7.4, we show how this property can be exploited to make queries fast.

2. *Unitigs are monochromatic.* The k-mers belonging to the same unitig u_i all have the same color. Thus, we shall use $\mathsf{Color}(u_i)$ to denote the color of each k-mer $x \in u_i$. We note that this property holds only if one considers k-mers appearing at the start or end of reference sequences to be *sentinel* k-mers that must terminate their containing unitig [193, 195, 196], and that such conventions are not always adopted [222, 228].

3. *Unitigs co-occur and share colors.* Unitigs often have the same color (i.e., occur in the same set of references) because they derive from conserved sequences in indexed references that are longer than the unitigs themselves. We indicate with M the number of distinct color sets $\mathcal{C} = \{C_1, \ldots, C_M\}$. Note that $M \leq m$ and that in practice there are dramatically more unitigs than there are distinct colors. We use $\mathsf{ColorID}(u_i) = j$ to indicate that unitig u_i has color C_j. As a consequence, each k-mer $x \in u_i$ has color C_j.

In this work our goal is to design an index that takes full advantage of these key properties.

7.3 Index description

In this section we describe a modular index that implements a colored compacted de Bruijn graph (ccdBG) and fully exploits its properties described in Section 7.2.3. We adopt the modular indexing framework from Section 7.2.2 — comprising a k-mer dictionary $\mathcal{D}$ and an inverted index $\mathcal{L}$ — to work seamlessly over the *unitigs* of the ccdBG. We extend the ideas from Fan et al. [182] for the modular indexing of k-mer positions to k-mer colors.

Our strategy is to first map k-mers to unitigs using a dictionary $\mathcal{D}$, and then map unitigs to their colors $\mathcal{C} = \{C_1, \ldots, C_M\}$. By *composing* these mappings, we obtain an efficient map directly from k-mers to their associated colors. The colors themselves in $\mathcal{C}$ are stored in compressed form in a inverted index $\mathcal{L}$. Figure 7.1 offers a pictorial overview of how we orchestrate these different components in the index. The goal of this section is to describe how these mapping steps can be performed efficiently and in small space.

7.3.1 The k-mer dictionary: mapping k-mers to unitigs with SSHash

For a k-mer dictionary, we use the SSHash data structure [23, 200], which fulfills the requirement described in Section 7.2.2, in that it implements the query $\mathsf{Lookup}(x)$ for any k-mer x efficiently and in compact space. This is achieved by storing the unitigs explicitly (i.e., as contiguous, 2-bit encoded strings) in some prescribed order so that a k-mer x occurring in some unitig u_i can be quickly located using a minimal perfect hash function [229] built for the set of the *minimizers* [230] of the k-mers. Laying out unitigs in this principled manner also enables very efficient streaming query. That is, when querying consecutive k-mers from input reads, the query for a given k-mer can often be answered very efficiently given the query result from its predecessor, since it often shares the same minimizer and frequently even occupies the very next position on the same unitig as its predecessor. We refer the interested reader to [23, 200] for a complete overview of SSHash.

Even more importantly for our purposes, a query into the SSHash dictionary returns, among other quantities, $\mathsf{UnitigID}(x) = i$, the ID of the unitig containing the k-mer x, as a byproduct of $\mathsf{Lookup}(x)$. For any k-mer occurring in $\mathcal{R}$, $\mathsf{UnitigID}(x) = i$ is an integer in $[1..m]$. This map from k-mers to unitigs will be exploited in the subsequent sections.

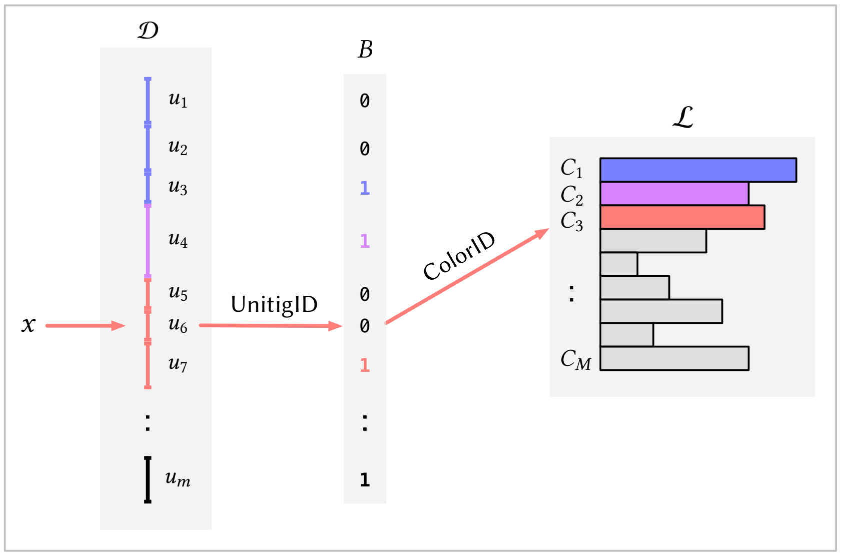


Figure 7.1: A schematic picture of the index described in Section 7.3, highlighting the interplay between the k-mer dictionary $\mathcal{D}$, the bit-vector B, and the inverted index $\mathcal{L}$. The red arrows show how the index is queried for a k-mer x, assuming that x occurs in unitig u_6 and has color C_3. The k-mer x is first mapped by $\mathcal{D}$ to its unitig u_6 via the query $\text{UnitigID}(x) = 6$. Then we compute $\text{ColorID}(u_6) = \text{Rank}_1(6, B) + 1 = 2 + 1 = 3$ and lastly retrieve C_3 from $\mathcal{L}$.

7.3.2 Mapping unitigs to colors

Now that we have an efficient map from k-mers to unitigs, i.e., the operation $\text{UnitigID}(x)$, we must subsequently map unitigs to distinct colors. That is, we have to describe how to implement the operation $\text{ColorID}(u_i)$ for each unitig u_i. Since each $\text{ColorID}(u_i)$ is an integer in $[1..M]$, we could implement $\text{ColorID}(u_i)$ just by storing $\text{ColorID}(u_1), \ldots, \text{ColorID}(u_m)$ explicitly in an array of $\lceil \log_2(M) \rceil$-bit integers. We show how to do this in just $1 + o(1)$ bits per unitig rather than $\lceil \log_2(M) \rceil$ bits per unitig.

We do so by exploiting another key property of SSHash: the unitigs it stores internally can be permuted in any desired order without impacting the correctness or efficiency of the dictionary. This was already noted and exploited in [200] to compress k-mer abundances. Similarly, here we sort the unitigs by $\text{ColorID}(u_i)$, so that all the unitigs having the same color are stored consecutively in SSHash. To compute $\mathsf{ColorID}(u_i)$, all that is now required is a Rank_1 query over a bit-vector $B[1..m]$ where:

- $B[i] = \mathtt{1}$ if $\mathsf{ColorID}(u_i) \neq \mathsf{ColorID}(u_{i+1})$ and $B[i] = \mathtt{0}$ otherwise, for $1 \leq i < m$;
- $B[m] = \mathtt{1}$.

It follows that B has exactly M bits set. The operation $\mathsf{Rank}_1(i, B)$ returns the number of ones in $B[1, i)$ and can be implemented in $O(1)$ time, requiring only $o(m)$ additional bits as overhead on top of the bit-vector [231, 232]. This means that $\mathsf{ColorID}(u_i)$ can be computed in $O(1)$ as $\mathsf{Rank}_1(i, B)+1$.

We illustrate this unitig to color ID mapping in Figure 7.1. In this toy example, $\mathsf{ColorID}(u_6) = 3$ can be computed with $\mathsf{Rank}_1(6, B) + 1 = 2 + 1$ because there are two bits set in $B[1, 6)$ — each marking where previous groups of unitigs with the same color end. Therefore, according to B, unitigs $\{u_1, u_2, u_3\}$ all have the same color as also $\{u_5, u_6, u_7\}$; u_4's color is not shared by any other unitig instead.

7.3.3 Compressing the colors

The inverted index $\mathcal{L}$ is a collection of sorted integer sequences $\{C_1, \dots, C_M\}$, whose integers are drawn from a universe of size N (the total number of references in the collection $\mathcal{R}$). There is a plethora of different methods that may be used to compress integer sequences (see, e.g., the survey [204]). Testing the many different techniques available on genomic data is surely an interesting benchmark study to carry out. Here, however, we choose to adopt a simple strategy based on the widespread observation that effective compression appears to require using different strategies based on the density of the sequence C_i to be compressed (ratio between $|C_i|$ and N) [204]. For example, for the colored k-mer indexing problem, Alanko et al. also observe and report highly skewed distributions of color densities [224].

We therefore implement the following *hybrid* compression scheme:

1. For a sparse color set C_i where $|C_i|/N < 0.25$, we adopt a delta-gap encoding: the differences between consecutive integers are computed and represented via the universal Elias' δ code [233].

2. For a dense color set C_i where $|C_i|/N > 0.75$, we first take the complementary set of C_i, that is, the set $\overline{C_i} = \{j \in [1..N] | j \notin C_i\}$, and then compress $\overline{C_i}$ as explained in 1. above.

3. Finally, for a color set C_i, that does not fall into either above density categories, we store a characteristic bit-vector encoding of C_i — a bit-vector $b[1..N]$ such that $b[j] = \mathtt{1}$ if $j \in C_i$ and $b[j] = \mathtt{0}$ otherwise.

The compressed representations of all sequences are then concatenated into a single bit-vector, say *sequences*. An additional sorted sequence, *offsets*$[1..M]$, is used to record where each sequence begins in the bit-vector *sequences*, so that the compressed representation of the i-th sequence begins at the bit-position *offsets*$[i]$ in *sequences*, $1 \leq i \leq M$. The *offsets* sequence is compressed using the Elias-Fano encoding [234, 235] and takes only a (very) small part of the whole space of $\mathcal{L}$ unless the sequences are very short.

This hybrid encoding scheme is similar in spirit to the one also used in Themisto which, in turn, draws inspiration from Roaring bitmaps [236]. However, our choice of switching to the complementary set when $|C_i|$ approaches N turns out to be a very effective strategy, especially for pan-genome data, where a striking fraction of integers in $\mathcal{L}$ are indeed covered by these extremely dense sets (see also Table 7.4 from Section 7.5).

7.3.4 Construction

`fulgor` is constructed by directly processing the output of GGCAT [228], an efficient algorithm to build ccdBGs using external memory and multiple threads. Importantly, GGCAT provides the ability to iterate over unitigs grouped by color. Therefore, `fulgor` construction just requires a single scan of the unitigs in the order given by GGCAT. SSHash is built on the set of unitigs, each distinct color is compressed as described in Section 7.3.3, and the bit-vector B is also built during the scan.

7.4 Pseudoalignment algorithms

The term *pseudoalignment*, originally coined by Bray et al. [184] and developed in the context of RNA-seq quantification, has been used to describe many different algorithms and approaches, several of which do not actually comport with the original definition. Specifically, Bray et al. [184] define a "pseudoalignment of a read to a set of transcripts, T" as "a subset, $S \subseteq T$, without specific coordinates mapping each base in the read to specific positions in each of the transcripts in S". The goal of such an approach then becomes to determine, for a given read, the *set* of indexed reference sequences with which the read is *compatible*, where, in the most basic scenario, the compatibility relation can be determined entirely by the presence/absence of k-mers in the read in specific references.

Given any index of k-mer colors, a variety of different pseudoalignment algorithms can be implemented that rapidly map given reads to compatible reference sequences according to a set of heuristics. Below, we review four pseudoalignment algorithms and describe their properties. Various existing tools implement a subset of these pseudoalignment strategies and `fulgor` implements all four. These algorithms fall into two categories: (1) *exhaustive* methods that retrieves the color of every k-mer on a given read (as described in [224]), and (2) *skipping* heuristics that skip or jump over k-mers during pseudoalignment that are likely to be *uninformative* (i.e., to have the same color as the k-mer that was just queried).

7.4.1 Exhaustive methods

For a given query sequence Q, exhaustive approaches return colors with respect to a set of k-mers of Q, $K(Q)$, that map to a non-empty color (i.e., each k-mer $x \in K(Q)$ if found in the dictionary $\mathcal{D}$).

Full-intersection. The first of the two exhaustive approaches, the *full-intersection* method, simply returns the intersection between all the colors of the k-mers in $K(Q)$. Algorithm B1 in the Appendix (page 216) shows how this query mode is implemented in `fulgor`. In the current implementation, `fulgor` has a generic intersection algorithm that can work over *any* compressed color sets, provided that an iterator over each color supports two primitives — Next and NextGEQ(x), respectively returning the integer immediately after the one currently pointed to by the iterator and the smallest integer which larger-than or equal-to x. (We point the reader to [237] and [204] for details.)

Threshold-union. The second algorithm, which we term the *threshold-union* approach, relaxes the full-intersection method to trade off precision for increased recall. Instead of requiring a reference to be compatible with *all* mapped k-mers, the threshold-union method requires a reference to be compatible with a user defined proportion of k-mers. Given a parameter $\tau \in (0, 1]$, this method returns the set of references that occur in *at least* $s \cdot \tau$ returned (i.e., non-empty) k-mer colors, where s can be either chosen to be $s = |K(Q)|$ (the number of positive k-mers only) or $s = |Q| - k + 1$ (the total number of k-mers in Q). Themisto [224] implements the variant with $s = |K(Q)|$ (called the "hybrid" method), whereas both Bifrost [222] and Metagraph [223] use $s = |Q| - k + 1$. In fact, the latter approach of simply looking up all of the k-mers in a query, and requiring a specified fraction of them to match, is a long-standing strategy that predates the notion of pseudoalignment [214, 238]. In the following, we assume $s = |K(Q)|$ is used by the threshold-union algorithm, unless otherwise specified. The pseudocode for this query mode is given in Algorithm B3 in the Appendix (page 218).

In practice, both the aforementioned exhaustive methods are efficient to compute for two reasons. First, intersections, thresholding, and unions are easy to compute because colors are encoded as monotonically increasing lists of reference IDs. Second, for `fulgor` in particular, querying *every* k-mer for its color can be performed in a highly-optimized way via *streaming* queries to SSHash. In the streaming setting, SSHash may skip comparatively slow hashing and minimizer lookup operations because it stores *unitig* sequences contiguously in memory. When sequentially querying adjacent k-mers on a read that are also likely adjacent on indexed unitigs, it can rapidly lookup and check k-mers that are cached and adjacent in memory (we refer the reader to [23] for more details).

7.4.2 *Skipping* heuristics

For even faster read mapping, pseudoalignment algorithms can implement heuristic *skipping* approaches that avoid exhaustively querying all k-mers on a given read. These skipping heuristics make the assumption that whenever a k-mer on a read is found to belong to a unitig, subsequent k-mers will likely map to the same unitig and can therefore be skipped, since they will be uninformative with respect to the final color assigned to the query (i.e., the intersection of the colors of the mapped k-mers).

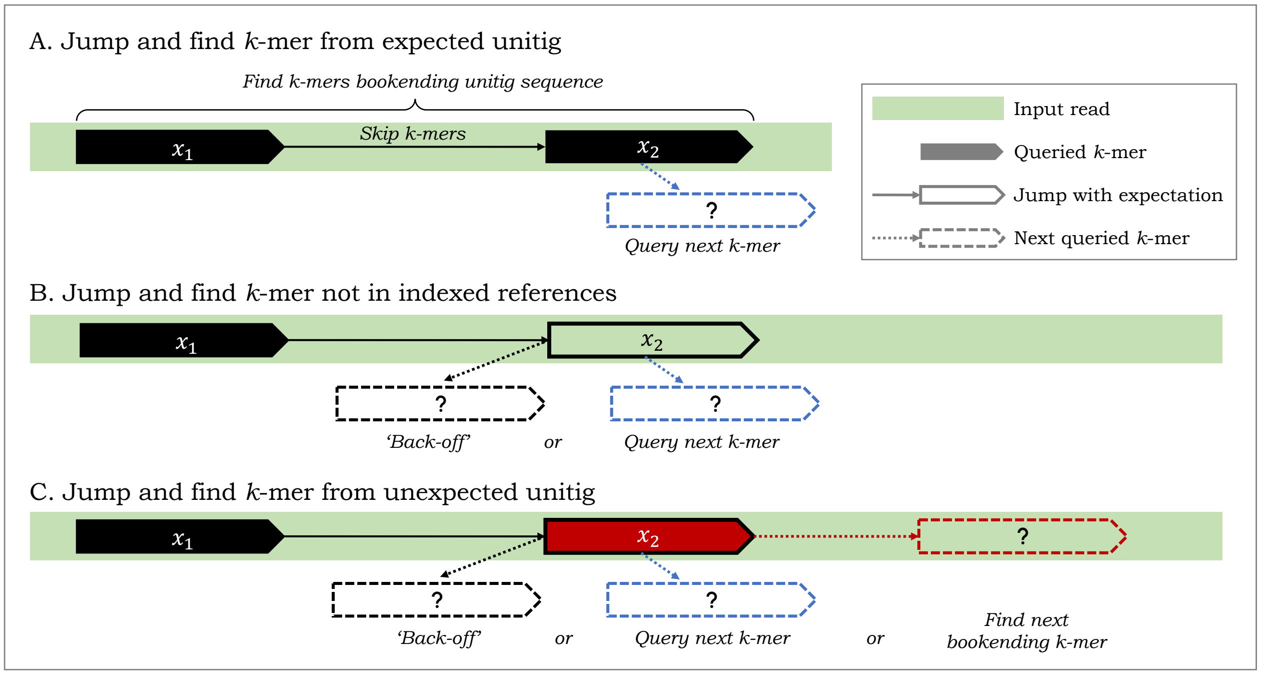

A. Jump and find k-mer from expected unitig
Find k-mers bookending unitig sequence
Skip k-mers
x_1
x_2
?
Query next k-mer
Input read
Queried k-mer
Jump with expectation
Next queried k-mer
B. Jump and find k-mer not in indexed references
x_1
x_2
?
?
'Back-off'
or
Query next k-mer
C. Jump and find k-mer from unexpected unitig
x_1
x_2
?
?
?
'Back-off'
or
Query next k-mer
or
Find next bookending k-mer

Figure 7.2: Some relevant design choices for pseudoalignment with skipping heuristics that *jump* and skip k-mers on a given read. After k-mer x_1 is queried and found to map to a "black" unitig, an algorithm can jump to query the k-mer x_2 on input read, where the number of k-mers skipped is given by the length of the black unitig. (A) In the ideal scenario, x_2 maps to the black unitig sequence and k-mers x_1 and x_2 are found to bookend this unitig sequence as it appears on the read. (B) If x_2 misses the index, an algorithm can *back-off* to an earlier k-mer on the read to find a k-mer bookending a shorter subsequence of the black unitig; or it may just query the next k-mer. (C) If x_2 maps to a different "red" unitig, an algorithm has an alternative, aggressive, heuristic option to jump and find the next k-mer bookending the red unitig sequence.

Bray et al. [184] first described such an approach, where a successful search that returns a unitig u triggers a skip that moves the search position forward to either the end of the query or the implied distance to the end of u (whichever is less). Subsequent searches follow the same approach as new unitigs are discovered and traversed in the query. Later, other tools extended or modified the proposed skipping heuristics, and introduced "structural constraints", which take into account the co-linearity and spacing between matched seeds on the query and on the references to which they map [225]. In contrast to Themisto, `fulgor` has rapid access to the topology of the ccdBG because its k-mer dictionary, SSHash, explicitly maps k-mers to unitig sequences that are stored contiguously in memory. `fulgor` thus permits efficient implementation of pseudoalignment algorithms with skipping heuristics since, due to the underlying capabilities provided by SSHash, it can rapidly find k-mers bookending unitig substrings because SSHash can explicitly map k-mers to their offsets (positions) in indexed unitig sequences.

In general, pseudoalignment methods that implement skipping heuristics must specify what steps the algorithm will take in *all* scenarios, not just what should happen when search proceeds as expected. In practice, implementations for resolution strategies are complicated and difficult to describe succinctly in prose, and prior work has only discussed these important details in passing. Here, using the depicted scenarios in Figure 7.2, we provide a more structured (though certainly not exhaustive) discussion of possible design choices that can be made. These design choices impact the performance of the pseudoalignment algorithm, both in terms of how many k-mers it queries (and, hence, its speed), and in how many distinct color sets it collects (and, hence, the actual compatibility assignment it makes).

Jump and find k-mer in expected unitig. Before the first matching k-mer of a read is found, there is relatively little difference between exhaustive and heuristic pseudoalignment approaches; subsequent k-mers are queried until the read is exhausted or some k-mer is found in the index. At this point, however, heuristic skipping methods diverge from the exhaustive approaches. At a high level, when a k-mer on a read is found to map to a unitig, skipping heuristics make an assumption that said unitig appears wholly on the read. A pseudoalignment algorithm then jumps, on the read, to what would be the last k-mer on the unitig sequence occurring on the given read (i.e., a bookending k-mer). Scenario A in Figure 7.2 depicts when this assumption is correctly made. Moving left-to-right on a given read, if a k-mer on the *left* is found to occur on the unitig depicted in black color in the figure (referred to as the "black" unitig henceforth), an algorithm can then skip a distance given by the length of the black unitig and jump to a k-mer to the right that also maps to the black unitig and bookends it. Doing so, an algorithm can assume that all k-mers bookended by these two queried k-mers map to the black unitig, avoid querying k-mers in-between, and instead continue to query the next k-mer on the read (indicated in dashed lines in blue).

Jump and miss k-mer. In practice however, the implemented skipping heuristics are not so simple. This is because, when skipping k-mers according to unitig lengths, the resulting k-mer that an algorithm jumps to may not necessarily map to the unitig it *expects*. In scenario B, an algorithm jumps to a k-mer on a read, expecting it to map to a black unitig, but finds that it does not correspond to any indexed k-mer. Here, an algorithm can make several choices, and in fact, current skipping heuristics make two distinct choices in this scenario. It can ignore this missed k-mer and simply query the next k-mer after the position that was jumped to (in blue). Or, it can take a more conservative approach and implement a *back-off* scheme to look for another k-mer that maps to the black unitig. An algorithm can back-off and jump a lesser distance, and such a back-off approach can happen once or can be recursive or iterative until some termination condition is satisfied.

Jump and find k-mer in *un*-expected unitig. In scenario C, an algorithm that jumps to a k-mer but finds that it maps to a *different* (red) unitig than expected. Here, we suggest three choices an algorithm can make. Like in scenario B, an algorithm can back-off to find another k-mer mapping to the black unitig or it can query the next k-mer after the jumped position. Alternatively, it can take a new more aggressive approach and jump to a k-mer on the read where it expects to find the end of an occurrence of the red unitig.

In this work, we have retrofitted the pseudoalignment with skipping algorithms from Kallisto [184][1]

[1]`https://github.com/jermp/fulgor/blob/main/kallisto_psa/psa.cpp`

and Alevin-fry [225][2] to make use of `fulgor`, rather than the distinct indexes atop which they were implemented in their original work. Using `fulgor`, we compare their resulting pseudoalignments, along with those from the full-intersection and threshold-union approaches, in a simple simulated scenario in Section 7.5.4.

7.5 Results

In this section, we report experimental results to assess `fulgor`'s construction time/space, index size, and query speed. Throughout the section, we compare `fulgor` to Themisto [224], which has been shown to outperform other methods that build similarly capable indexes (namely Bifrost [222] and Metagraph [223]) in terms of speed and space. We build Themisto indexes using the fastest configuration, i.e., without sampling of k-mer colors in the SBWT (build option `-d1`), as done by the authors in [224]. Not sampling k-mer colors yields slightly larger indexes but makes Themisto faster to query. For our largest benchmarked reference collection (150,000 genomes), potential space savings from sampling is not significant anyway because the space required to store distinct colors dominate the overall space. We also use Themisto's default color set representation (i.e., without Roaring bitmaps). For both `fulgor` and Themisto, we set the k-mer size to $k = 31$.

Datasets. We follow the experimental methodology of [224] and build `fulgor` over subsets of *Salmonella enterica* genomes (up to 150,000 genomes) from [239], to demonstrate `fulgor`'s effectiveness when indexing collections of similar reference sequences. We also consider a heterogeneous collection of 30,691 genomes of bacterial species representative of the human gut [206] (as also benchmarked in our previous work [182]). We report some summary statistics for the indexed ccdBGs in Table 7.1.

[2]`https://github.com/jermp/fulgor/blob/main/piscem_psa/hit_searcher.cpp`

Hardware and software. All experiments were run on a machine equipped with Intel Xeon Platinum 8276L CPUs (clocked at 2.20GHz), 500 GB of RAM running Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0). `fulgor` is available at `https://github.com/jermp/fulgor`. For the experiments reported here we use v1.0.0 of the software, compiled with gcc 11.1.0. For Themisto, we use the shipped compiled binaries (v3.1.1).

7.5.1 Construction time and space

Construction time and peak RAM usage is reported in Table 7.2 for the different datasets evaluated. Both tools use GGCAT to build the ccdBG. However, `fulgor` is $2-6\times$ faster, and typically consumes much less memory during construction. This is because Themisto spends most of its time and memory building the color mapping. However, the analogous component of `fulgor` is just a bit vector, demarcating groups of unitigs with the same color, that is built via a linear scan of the unitigs produced by GGCAT.

Figure 7.3 shows, instead, `fulgor`'s construction time breakdown for some illustrative datasets. We distinguish between three phases in the construction: (1) running GGCAT, (2) compressing the colors and, (3) building SSHash. While GGCAT and color compression take most of the construction time on the Salmonella pangenomes, building SSHash is the most expensive step on the Gut Bacteria collection. This is consistent with the statistics reported in Table 7.1. Here, there are far more integers to compress in the Salmonella collections whereas the Gut Bacteria collection contains one order of magnitude more k-mers. This suggests that one could achieve even faster construction for `fulgor` if the colors are compressed in parallel with the SSHash construction (currently, these two phases are sequential).

Table 7.1: Summary statistics for the tested collections. The row "Integers in colors" reports the total number of reference IDs that are required to encode all colors — i.e., the sum set sizes for all colors, $\sum_i |C_i|$.

	Salmonella					Gut Bacteria
Genomes	5,000	10,000	50,000	100,000	150,000	30,691
Distinct colors ($\times 10^6$)	2.69	4.24	13.92	19.36	23.61	227.80
Integers in colors ($\times 10^9$)	5.77	15.68	133.49	303.53	490.04	10.04
k-mers in dBG ($\times 10^6$)	104.69	239.88	806.23	1,018.69	1,194.44	13,936.86
Unitigs in dBG ($\times 10^6$)	4.95	8.24	30.64	41.16	49.60	566.39

7.5.2 Index size

When indexing collections of Salmonella genomes, `fulgor` is consistently $\approx$ $2\times$ smaller than Themisto as apparent from Table 7.3. For example, on the largest collection comprising 150,000 genomes, `fulgor` takes 70.66GB whereas Themisto takes 133.63GB. This remarkable space improvement is primarily due to the more effective color compression scheme adopted by `fulgor`. This leads to, for example, 48% less space to encode colors for the 150,000 collection of Salmonella genomes. Looking at Table 7.4, we highlight that for all indexed Salmonella reference collections, approximately 50% of all encoded integers in the distinct colors belong to colors that are *at least* 90% dense. For such extremely dense colors, the complementary encoding strategy described in Section 7.3.3 is very effective: only $\approx$ 0.2 bits/int (bpi) are required to encode them in all benchmarked indexes. In fact, even for our largest collection of 150,000 Salmonella genomes, encoding *all* integers in *all* colors requires only 1.120 bpi.

Unsurprisingly, `fulgor` also uses less space than Themisto to support the ColorID operation. We recall from Section 7.3.2 that `fulgor` requires only $1 + o(1)$ bits per unitig by design. This amounts to a negligible space usage compared to the overall index size. For example, while Themisto requires 7.26GB to map k-mers to color IDs for 150,000 Salmonella genomes, our strategy just takes

Table 7.2: Total index construction time and GB of memory (max. RSS), as reported by `/usr/bin/time` with option `-v`. The reported time includes the time taken by GGCAT to build the ccdBG (using 48 processing threads) and the time to serialize the index on disk.

	Genomes	`fulgor` hh:mm	`fulgor` GB	Themisto hh:mm	Themisto GB
Salmonella	5,000	00:04	12.91	00:11	12.97
	10,000	00:09	23.60	00:25	23.58
	50,000	01:13	43.76	02:32	96.00
	100,000	02:56	73.54	06:25	202.42
	150,000	04:36	136.94	10:00	323.10
Gut Bacteria	30,691	02:27	115.05	15:35	327.72

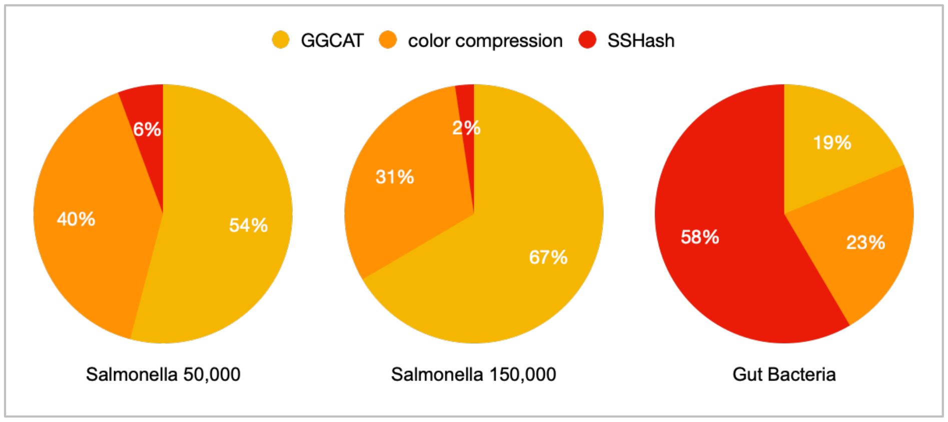


Figure 7.3: Construction time breakdown for `fulgor`.

7.75 MB.

When indexing a *heterogeneous* collection, e.g., the 30,691 bacterial genomes [206], with many more unique k-mers, the space advantage `fulgor` has over Themisto is even more apparent. First, the overall size of `fulgor` is 3.8$\times$ smaller (36.77GB versus 139.41GB). Second, `fulgor`'s near optimal approach of mapping *unitigs* to colors instead of k-mers to colors is dramatically more efficient, requiring only 88MB compared to Themisto's 91GB. Themisto, by using the SBWT, organizes k-mers based on their *colexicographical* order and requires $\lceil \log_2(M) \rceil$ bits per sampled k-mer to record the color IDs. Here, the SBWT must record colors for each of the 13.9 billion distinct k-mers *and* their reverse complement. In contrast, `fulgor` uses SSHash that maintains k-

Table 7.3: Index space in GB, broken down by space required for indexing the k-mers in a dBG (SSHash for `fulgor`, and the SBWT for Themisto); and data structures required to encode colors and map k-mers to colors.

	Genomes	`fulgor`			Themisto		
		dBG	Colors	Total	dBG	Colors	Total
Salmonella	5,000	0.16	0.59	0.75	0.14	1.82	1.96
	10,000	0.35	1.66	2.01	0.32	4.78	5.09
	50,000	1.26	17.03	18.30	1.07	36.89	37.96
	100,000	1.72	40.70	42.44	1.35	81.82	83.17
	150,000	2.03	68.60	70.66	1.58	132.05	133.63
Gut Bacteria	30,691	21.23	15.45	36.77	18.33	121.08	139.41

mers in unitig order and requires only $1 + o(1)$ bits per unitig to map *all* k-mers from the same unitig to a single color. Although not the default behavior, Themisto can optionally sample k-mer colors to avoid storing one color ID per each k-mer. Clearly, the sampling scheme reduces space usage at the expense of some overhead at query time by requiring an implicit walk in the dBG. While this sampling strategy can be quite effective when the underlying k-mer set induce long unitigs, allowing the sampling of the terminal k-mers of a non-branching path [224], it is unlikely to be similarly effective in a highly-branching and fragmented graph like the one underlying this heterogeneous dataset where unitigs are short. On the contrary, our index does not have this issue *by design* and can thus scale to more heterogenous collections using small space.

7.5.3 Query speed

To compare query speed, we benchmark `fulgor` and Themisto using both low- and high-hit rate read-sets, i.e., read-sets for which we have a low and high number of positive k-mers respectively. Precisely, we use the files containing the first read of the following paired-end libraries:

Table 7.4: Average bits/int (bpi) spent for representing colors whose density is $(i \times 10)\%$ of N, for $i = 1, \ldots, 10$. The first two columns for each collection, "lists" and "ints", report the percentage of lists (i.e., colors) and integers (stored reference identifiers) that belong to all colors within a given density. The last row, "Total bpi", is comprehensive of the space spent for the integer lists themselves and the space spent for the offsets delimiting the lists' boundaries.

Density	Salmonella															Gut Bacteria		
	$N = 5{,}000$			$N = 10{,}000$			$N = 50{,}000$			$N = 100{,}000$			$N = 150{,}000$			$N = 30{,}691$		
	lists	ints	bpi	lists	ints	bpi	lists	ints	bpi	lists	ints	bpi	lists	ints	bpi	lists	ints	bpi
0–10%	38.88	2.00	4.66	46.15	1.81	4.64	70.96	2.62	6.00	76.52	3.14	6.16	79.23	3.27	6.32	99.99	99.99	12.05
10–20%	6.04	2.11	2.66	4.83	1.93	2.66	3.74	2.84	3.05	2.82	2.61	3.77	2.54	2.68	3.92	0.00	0.00	0.00
20–30%	4.70	2.69	2.86	4.44	2.93	2.81	2.69	3.50	3.24	2.32	3.66	3.41	2.09	3.76	3.46	0.00	0.00	0.00
30–40%	3.13	2.55	2.88	4.27	4.02	2.87	1.90	3.43	2.89	1.57	3.49	2.88	1.40	3.51	2.88	0.00	0.00	0.00
40–50%	4.05	4.25	2.23	3.32	4.04	2.22	1.81	4.25	2.22	1.44	4.14	2.23	1.29	4.19	2.23	0.00	0.00	0.00
50–60%	4.13	5.30	1.83	3.54	5.29	1.81	1.82	5.24	1.82	1.42	4.99	1.82	1.24	4.94	1.82	0.00	0.00	0.00
60–70%	3.98	6.07	1.54	4.24	7.44	1.54	2.04	6.94	1.53	1.59	6.61	1.54	1.40	6.59	1.54	0.00	0.00	0.00
70–80%	5.53	9.72	0.94	4.86	9.91	0.93	2.33	9.13	1.08	1.87	8.96	1.14	1.64	8.91	1.15	0.00	0.00	0.00
80–90%	5.80	11.52	0.47	3.71	8.57	0.47	3.03	13.43	0.56	2.49	13.65	0.63	2.09	12.95	0.66	0.00	0.00	0.00
90–100%	23.77	53.80	0.15	20.65	54.07	0.14	9.67	48.63	0.19	7.94	48.76	0.21	7.07	49.21	0.21	0.00	0.00	0.00
Total bpi	0.817			0.848			1.020			1.072			1.120			12.32		

SRR896663[3] with 5.7×10^6 reads, SRR801268[4] with 6.6×10^6 reads, and ERR321482[5] with 6.8×10^6 reads.

In Table 7.5 we report the result of the comparison using the full-intersection method (Algorithm B1). We repeated the same experiment using the threshold-union method (Algorithm B3) with parameter $\tau = 0.8$ as this is the preferred query mode in Themisto. However, we did not observe any appreciable difference compared to the full-intersection method in terms of query speed.

In a low-hit rate workload where a small proportion of reads map to the indexed references, `fulgor` is much faster than Themisto. In this scenario, we expect many queried k-mers to not occur in the indexed references. When k-mers are absent from the index, no color needs to be retrieved and only the k-mer dictionary is queried. Here, `fulgor` is faster than Themisto because its reliance on the fast streaming query capabilities of SSHash. It is worth noting here that in any *streaming* setting

[3]https://www.ebi.ac.uk/ena/browser/view/SRR896663
[4]https://www.ebi.ac.uk/ena/browser/view/SRR801268
[5]https://www.ebi.ac.uk/ena/browser/view/ERR321482

where consecutive k-mers are queried, `fulgor` can fully exploit the monochromatic property of unitigs in ways which Themisto cannot. Queries to SSHash have very good locality compared to the SBWT because adjacent k-mers in unitigs are stored contiguously in memory. Further, streaming queries to SSHash can be very efficiently cached and optimized. When looking up consecutive k-mers, SSHash can entirely avoid computing its minimal perfect hash (a slow operation) and instead perform fast comparisons of k-mers stored in cached positions pointing to adjacent addresses in memory.

In a high-hit rate workload, `fulgor` also outperforms Themisto, but by a smaller margin, since most of the time is now spent in performing the intersection between colors. It is interesting to note that the workloads can be processed significantly faster (by both tools) on the Gut Bacteria collection: this is a direct consequence of the fact that the lists being intersected are much shorter on average for the Gut Bacteria compared to the Salmonella collections. This is evident from Table 7.4: essentially all lists are just 10% dense, i.e., have length at most $\lceil 30,691/10 \rceil < 3{,}070$.

We also note that part of the slowdown seen for Themisto is due to the time spent in loading the index from disk to RAM, which takes at least twice as `fulgor`'s because of its larger index size.

7.5.4 Comparison of pseudoalignment algorithms on simulated data

To analyze the accuracy of the underlying pseudoalignment algorithms, we perform additional testing with read sets simulated using the Mason [240] simulator. To analyze how mapping and hit rates affect query speed, we simulate a varying proportion of "positive" reads from indexed reference sequences and generate "negative" reads from the human chromosome 19 from the CHM13 v2.0 human genome assembly [241]. We use `fulgor` to compare the four mapping algorithms described in Section 7.4.

Table 7.5: Total query time as elapsed time reported by `/usr/bin/time`, using 16 processing threads for both indexes. The read-mapping output is written to `/dev/null` for this experiment. We also report the mapping rate in percentage (fraction of mapped read over the total number of queried reads). Results are relative to the full-intersection query mode. All reported timings are relative to a second run of the experiment, when the index is loaded faster from the disk cache. For each workload, we indicate the run accession number.

a low-hit, Salmonella, SRR896663

Genomes	Mapping rate	`fulgor`	Themisto
		mm:ss	mm:ss
5,000	1.27	00:09	00:32
10,000	13.86	00:10	00:36
50,000	32.61	00:25	01:05
100,000	34.09	00:45	01:39
150,000	34.01	01:06	05:02

b high-hit, Salmonella, SRR801268

Genomes	Mapping rate	`fulgor`	Themisto
		mm:ss	mm:ss
5,000	89.53	01:16	03:50
10,000	89.76	02:26	07:35
50,000	91.31	19:15	41:25
100,000	91.52	35:50	82:14
150,000	91.61	42:30	120:08

c low-hit, Gut Bacteria, SRR896663

Genomes	Mapping rate	`fulgor`	Themisto
		mm:ss	mm:ss
30,691	11.90	0:57	2:58

d high-hit, Gut Bacteria, ERR321482

Genomes	Mapping rate	`fulgor`	Themisto
		mm:ss	mm:ss
30,691	92.98	01:16	02:45

From Table 7.6, we see that at various proportions of ground truth positive reads (simulated reads deriving from indexed references), all mapping methods have a true positive rate (TPR), i.e., total reads correctly mapped over the total ground truth positives, greater than 95%. This high sensitivity for all four methods is to be expected since all methods simply check for k-mer's membership to references of origin and do not consider k-mer positions in references. One main drawback of eliding positions, heuristically avoiding "locate" queries, and entirely ignoring k-mers that are not present in the index, is also clear. All methods incur approximately a 30% false positive rate (FPR), i.e., total reads spuriously mapped over the total ground truth negatives. As is expected, the threshold-union method incurs a slightly higher FPR compared to other methods (30% compared to 27% for other methods) because of its less strict criteria only requiring references to be compatible with τ fraction of mapped k-mers instead of *all* k-mers.

Table 7.6: Quality of pseudoalignment algorithms querying 100,000 simulated reads against 50,000 Salmonella genomes indexed with `fulgor`. We vary the percentage of *positive* reads simulated from indexed Salmonella genomes by diluting queried read sets with *negative* reads simulated from a reference human transcriptome. We consider a mapped positive read (deriving from indexed references) to be a *true positive* if the reference of origin is in the returned set of compatible references; and a mapped negative read (deriving from human chromosome 19) to be a *false positive*. We denote true and false positive rates (%) to be TPR and FPR, respectively. For the threshold-union method, we use $\tau = 0.8$.

% Positive	Full-intersection		Threshold-union		Kallisto		Alevin-fry	
	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
90%	95.0	27.0	97.7	30.0	95.0	27.0	95.1	27.0
70%	95.1	27.0	97.7	30.0	95.1	27.0	95.1	27.0
25%	95.1	27.0	97.7	30.0	95.2	27.0	95.2	27.0
10%	95.5	27.0	97.8	30.0	95.5	27.0	95.5	27.0

In these benchmarks, we find very little difference in terms of TPR and FPR between the exhaustive methods and skipping heuristics. These results also gesture at one desirable and one undesirable quality of these methods. First, skipping heuristics correctly assume and successfully skip k-mers that likely occur on the same unitig and have the same color. Likewise, they have the *potential* to be even more sensitive than the full-intersection method, as they do not, in general, search for every k-mer in a query, and can thus avoid scenarios where variation or sequencing errors in a query cause spurious matches to the index, shrinking or eliminating the set of references appearing in the final color assigned to the query. In fact, in a small-scale test, Alanko et al. [224] report that Kallisto's skipping heuristic results in a small but persistent increase of approximately 0.03% in the mapping rate. However, all four of the pseudoalignment methods evaluated here suffer from a high FPR and low precision. Better algorithms to lower FPR and improve precision without lowering sensitivity too much should be investigated in future work. Such improvements may be possible by adding back information about the *reference positions* where k-mers from the query match, incorporating structural constraints [225] or other such restrictions atop the color intersection rule. Yet, those approaches are more computationally involved, require the index to support locate queries, and also substantially diverge from "pseudoalignment" as traditionally understood. Regardless, we highlight here that `fulgor` more easily enables implementing skipping and unitig-based heuristics compared to other methods that do not explicitly store unitig sequences and keep k-mers in unitig order. In fact, `fulgor` implicitly maintains additional information regarding the *local structural consistency* of k-mers. For example, with `fulgor`, one can easily check if consecutive k-mers are valid on an indexed unitig or check if consecutive unitigs on a read have valid overlaps, in an attempt to reduce the FPR.

7.6 Discussion

We introduce `fulgor`, a fast and compact index for the k-mers of a colored compacted de Bruijn graph (ccdBG). Using, SSHash, an order-preserving k-mer dictionary, `fulgor` fully exploits the monochromatic property of unitigs in ccdBGs. `fulgor` implements a very succinct map from unitigs to colors, taking only $1 + o(1)$ bits per unitig. Further, `fulgor` applies an effective hybrid compression scheme to represent the set of distinct colors. Across all benchmarked scenarios, `fulgor` outperforms Themisto, the prior state-of-the-art in terms of space *and* speed. There is still room for improvement in future work. We discuss some promising directions below.

In terms of speed, we remark that when processing a high-hit workload, the overall runtime is dominated by the time required to *intersect* the colors. As explained in Section 7.4.1, `fulgor` currently implements a generic intersection algorithm that only requires two primitive operations, namely Next and NextGEQ (see also Appendix B). But this is not the only paradigm available for efficient intersection. We could, for example, try approaches that exploit different indexing paradigms, such as Roaring [236] and Slicing [242], that are explicitly designed for fast intersections. These alternative approaches may be significantly faster especially on the high-hit workloads.

Another possible optimization is to implement a caching scheme for frequently occurring and/or recently intersected colors. Caching the uncompressed or intersection-optimized versions of frequently occurring color sets, or previously computed intersections, could speed up query processing substantially when many reads map to the same set of colors.

In terms of space, one property that `fulgor` does not yet exploit is the fact that *many* unitigs in the ccdBG share *similar colors* — i.e., co-occur in many reference sequences. This is so because unitigs arising from conserved genomic sequences will share similar occurrence patterns. In a related line of research, [243] developed a method that efficiently compresses distinct, but highly-correlated colors, through a variant of referential encoding. Specifically, they compute a minimum spanning tree (MST) on a subgraph of the color graph induced by the ccdBG, and encode a color by recording its differences with respect to its parent in the MST. This vastly reduces the space required to encode the color set when many similar colors exist, as we would expect in a pangenome, and fast query speed can be retained through color caching. Another related approach would be to resort to clustering similar colors and encoding all colors within a cluster with respect to a cluster representative color [244]. Likewise, although not specifically designed to compress colors, Metagraph and its variants can exploit similarity between colors using a general compression scheme that records differences in stored metadata (in this case, the colors) between adjacent k-mers [223]. We note that, since the colored k-mer indexing problem is *modular* (Section 7.2.2), novel relational compression techniques for the set of distinct colors can be developed and optimized independently of the other components of the index.

Finally, in our experiments with simulated data analyzing the quality of pseudoalignment algorithms from Section 7.5.4, we find higher than desirable false positive rates. This suggests that, at least for the metagenomic and pangenomic reference collections where many references share similar k-mer content, better read-mapping heuristics and algorithms that improve specificity (i.e., reduce the spurious mapping of reads not arising from indexed references) without trading-off too much recall are still sorely needed. Here, it will be desirable to search for methods that can improve specificity without the need to retain reference positions or issue locate queries for all k-mers. We suggest that there may be several promising directions. For example, one may consider enforcing local structural consistency among matched k-mers to potentially reduce spurious mapping. Likewise, one may consider filtering repetitive and low-complexity k-mers from contributing to the final pseudoalignment result. Finally, by analogy to BLAST [31], one may consider evaluating the likelihood that a pseudoalignment result is spurious by comparing the matching rate against against some null or background expectation to account for the fact that, in very large reference databases, a very small number of (potentially correlated) k-mers may be insufficient evidence to consider a query as compatible with a subset of references.

Chapter 8: How to implement fast and modular indexes *in practice*

In this Chapter, we put together theory and practice. First, we introduce an optimization to the query algorithm for `SSHash` [23]. Along the way, we define the minimizer of a *canonical* k-mer and show how ties in minimizer selection schemes for canonical k-mers must be broken carefully. Finally, we describe how we designed `mazu`[1], a Rust library that makes building and designing modular indexing schemes easy.

8.1 Replacing each scan over a candidate super-k-mers in `SSHash` with a single lookup

The linear scan over candidate matching k-mers in `SSHash` can be replaced by a single lookup at query time. With an MPHF, $f(\cdot)$ over all minimizers constructed, the original `SSHash` algorithm stores all super-k-mers[2] with a minimizer r, in a bucket B_r. More specifically, for each super-k-mer in the bucket B_r, `SSHash` stores the offset where said super-k-mer occurs in a reference unitig sequence. At query time, when a k-mer x has minimizer r, `SSHash` performs a linear scan over each candidate super-k-mer in bucket B_r to find the super-k-mer in which x occurs and returns an appropriate hash value.

Here, we introduce an optimization in which each linear scan of a candidate super-k-mer can be

[1] `https://github.com/COMBINE-lab/mazu`

[2] A super-k-mer is the sequence containing consecutive k-mers that have the same minimizer

replaced by a single lookup. In each bucket B_r, instead of storing the position of each super-k-mer, we store the position of each unique occurrence of the minimizer r. The key insight is that, at query time, the offset of the minimizer of the *queried* k-mer implies the offset of a candidate matching k-mer on a reference unitig sequence *with respect to* the corresponding minimizer's position on the reference.

Let a k-mer x have minimizer r with offset o — with $r = x[o : o + w]$ (where w is the minimizer size $< k$). On a reference unitig sequence, where x occurs at position p, the corresponding occurrence of the minimizer r must then occur at position $p + o$. Let the position $p_r = p + o$ of an occurrence of the minimizer sequence r be stored in bucket B_r. To find the corresponding occurrence of r on the reference unitig sequence that contains x, `SSHash` can simply check to see if the sequence of x matches the sequence at position $(p_r - o)$; thus eliminating a scan the k-mers of each candidate super-k-mer.

8.2 Canonical k-mers and their minimizers

To avoid duplicated indexing of both forward and reverse-complemented unitig sequences and repeated query of a k-mer and its reverse-complement, `SSHash` as well as other hashing based k-mer indexes (like `pufferfish` and `pufferfish2`) query *canonical* k-mers. Essentially, a k-mer x and its reverse complement $\overline{x}$ is uniquely represented by a single canonicalized sequence that is defined to be the lexicographically minimum string denoted $\widehat{x} = \min(x, \overline{x})$. At query time, the aforementioned algorithms compute one query for $\widehat{x}$, instead of two queries for both x and $\overline{x}$.

To apply the above optimization of `SSHash` to canonical k-mer queries, we must carefully define the *position* of a canonical k-mer's minimizer — especially in the event of ties where a minimizer sequence r occurs in more than one position on k-mer sequence x. Given a k-mer x, we define the minimizer of the canonical k-mer to be the minimum w-mer, r, on the canonical sequence $\hat{x}$. And we define the position, o, of said minimizer to be the position of the left-most occurrence of r on $\hat{x}$.

Importantly, the occurrence of the minimizer r at position o on the canonical sequence $\hat{x}$ on implies positions and occurrences of a corresponding w-mer on the uncanonicalized sequence x and its reverse complement $\overline{x}$. If $x = \hat{x}$, then the minimizer r of the canonical k-mer $\hat{x}$ occurs as the sequence r on $x[o : o+w]$. Here, the corresponding occurrence of $\overline{r}$ occurs on $\overline{x}[k-w-o : k-o]$. If $x \neq \hat{x}$ and the reverse-complement sequence is canonical (i.e., $\overline{x} = \hat{x}$) then the minimizer r of the canonical k-mer $\hat{x}$ occurs as the sequence $\overline{r}$ on $x[k-w-o : k-o]$. Here, r occurs on $\overline{x}[o : o+w]$.

We demonstrate how minimizers occur with an illustrative example with $k = 5$ and minimizer size $w = 2$. Consider the k-mer $x =$ `AATTT` which has canonical sequence $\hat{x} = \overline{x} =$ `AAATT` since $\overline{x} < x$. Notice that the minimizer `AA` is a substring that occurs in both x and $\overline{x}$ and occurs *multiple* times in $\hat{x}$. Our above definition ensures that the occurrences and positions of a minimizer on x, $\overline{x}$ and $\hat{x}$ exactly correspond and are the *same* w-mer. Here, the leftmost occurrence of $r =$ `AA` on $\hat{x}$ is selected as the minimizer of $\hat{x}$. This corresponds to the *rightmost* occurrence of `TT` at $x[3 : 5]$ on $x =$ `AATTT` since $\hat{x} = \overline{x}$. Accordingly, r occurs as the leftmost occurrence of `AA` on $\overline{x}$ at index 0.

To handle queries of a canonical k-mer, $\hat{x}$, `SSHash` stores the positions, on unitig sequence, of each minimizer of each *canonical* k-mer. At query time, `SSHash` has to compare a queried canonical k-mer with candidate k-mers on its stored uncanonicalized unitig sequences. Specifically, if the minimizer r of $\hat{x}$ occurs at some position p_r on the unitig sequence (as either r or the reverse-complement $\overline{r}$), `SSHash` checks not one, but two candidate corresponding that may match $\hat{x}$ — occurring at position $(p_r - o)$ or $(p_r - (k - w - o))$.

Other interesting properties and considerations.. There are some interesting and unintuitive properties of the above definition of minimizers of canonical k-mers when applied to extracting minimizers from consecutive and adjacent canonical k-mers of a nucleotide sequence. Consider a sequence S with adjacent k-mers $x_i = S[i : i + k]$ and $x_{i+1} = S[i + 1 : i + k + 1]$, the minimizer of the canonical-k-mer $\hat{x}_i$ on S may have *greater* index than the minimizer of $\hat{x}_{i+1}$ even though x_i occurs before x_{i+1} on S. In contrast to minimizers of uncanonicalized k-mers, minimizers of canonical k-mers may not necessarily have monotonically increasing positions. For example, consider $S = \texttt{AGGAAGA}$ with $k = 5$ and $w = 2$. For the zero-th canonical k-mer $\hat{x}_0 = \texttt{AGGAA}$, its minimizer is `AA` which occurs as $S[3 : 5] = \texttt{AA}$ on S. And for the next canonical k-mer $\hat{x}_1 = \texttt{CTTCC}$, its minimizer is `CC` which occurs in the reverse complement sequence as $S[1 : 3] = \texttt{GG}$ in an earlier position on S. Interestingly, $\hat{x}_2 = \texttt{GAAGA}$ has minimizer AA that also occurs on S at index 3.

This unintuitive edge case must be carefully handled when implementing algorithms that store unique minimizer positions. In the case of `SSHash`, one must decide how minimizer occurrences (with positions) are deduplicated (and stored in buckets) for groups of overlapping canonical k-mers occurring on a unitig sequence, S. One simple design choice to make construction simple would be to store the minimizer position for every group of *consecutive* canonical k-mers the same minimizer occurrence (with same position on S).

On a linear-time algorithm for enumerating minimizers of canonical k-mers.. The oft implemented algorithm for enumerating minimizer occurrences on a sequence S can be adapted for linear time enumeration of minimizers of consecutive canonical k-mers. The algorithm for uncanonicalized, consecutive k-mer sequences involves inserting each w-mer falling into a k-mer window into a double-ended queue while maintaining the following invariant. At the beginning of each iteration i, the minimizers of all suffixes of a k-mer $x_i = S[i : i + k]$ are stored in the queue sorted by their position on S. Ties are broken by selecting the *leftmost* minimizer on suffixes of x_i.

Thus, the minimizer for x_i at the front of the queue at each iteration.

For consecutive *canonical* k-mers on a string S, the algorithm maintains an additional queue for the minimizers of all suffixes of the reverse complement of the k-mer at position i on S, $\overline{x_i}$. But in this queue, ties are broken by selecting the *rightmost* minimizer on all suffixes of $\overline{x}_i$. To retrieve the minimizer for $\hat{x}_i$, if $x_i = \hat{x}_i$, then the minimizer at the head of the queue for x_i is returned, otherwise $\overline{x}_i = \hat{x}_i$ and the head of the queue for $\overline{x}_i$ is returned.

8.3 Designing mazu, a Rust library for modular indexing

In Chapter 6 we describe how, in theory, SPTs enable a generic class of modular indexes that decompose locate queries into a k-mer-to-unitig query and a unitig-to-occurrence query. Below, we show how, *in practice*, a Rust library can be designed to implement to fully realize the modular potential of our theoretical abstraction. By carefully defining shared behavior with Rust Traits, one can implement a single generic index implementation supporting the locate query. This generic implementation eliminates the need for almost all boilerplate and repeated implementations of the locate query. Doing so not only removes opportunities for programmer error but also makes it easy to generically implement validation, testing, and benchmarking of modularly composed indexes.

8.3.1 Toy library design

Below, we describe a simplified toy library design that closely resembles our implemented Rust library, `mazu`, for modular k-mer indexing. Our modular indexing library relies on shared behavior encoded by a pair of *Traits*, `K2U` and `U2Pos` — interfaces that exactly map to the k-mer-to-unitig and unitig-to-occurrence queries described in Section 6.4.

```
 1 pub trait K2U {
 2     fn k2u(&self, km: &CanonicalKmer) -> Option<K2UResult>;
 3 }
 4 
 5 pub trait U2Pos {
 6     fn u2pos(&self, unitig_id: &usize) -> Vec<UnitigOcc>;
 7 }
 8 
 9 pub struct SSHash { ... } // hidden
10 
11 impl K2U for SSHash {
12     fn k2u(&self, km: &CanonicalKmer) -> Option<K2UResult> { ... } // hidden
13  }
```

For example, the `impl` block that reads `impl K2U for SSHash` implements the Trait `K2U` for the `struct` `SSHash` that implements the `SSHash` algorithm.

Beginning on line 19, using Rust's Trait bounds, we can now write a *generic* implementation of the locate query for `ModIndex`. Specifically, we write an implementation of `ModIndex<H,T>` for any pair of types `H` that implements the Trait `K2U` and `T` that implements the Trait `U2Pos`.

On line 25 we implement the locate query for our genericized modular index, This modular implementation maps exactly to the modular index definition in Section 6.3.2. On line 26, with member `self.k2u` that implements `K2U`, our index computes the k-mer-to-unitig query. On line 28, with member `self.u2pos`, our index computes the unitig-to-occurrence query to retrieve all occurrences of a unitig containing the queried k-mer. On line 31, the offset of the queried k-mer in its corresponding unitig is projected onto the positions and occurrences of the unitig on the indexed reference sequences — completing the locate query.

```rust
pub struct ModIndex<H, T> {
    u2pos: T,
    k2u: H,
}

impl<H, T> ModIndex<H, T>
where
    H: K2U,
    T: U2Pos,
{
    // generic implemation for locate!
    pub fn locate(&self, km: &CanonicalKmer) -> Option<Hits> {
        let k2u_result = self.k2u.k2u(km)?;
        let unitig_id = k2u_result.unitig_id;
        let unitig_occs = self.u2pos.u2pos(unitig_id);

        // Add k-mer offset on unitig to unitig positions on occurrences
        let hits = project_hits(k2u_result, unitig_occs);
        Some(hits)
    }
}
```

Optimizations such as caching, as discussed in Section 6.4.5 and Appendix A.6 can also be generically implemented. In the example below, the struct `CachingIndex` holds references to a `H` and `T` respectively and *owns* a `Cache` that can be mutated at query time. In line 41, we implement a cheap reference-to-reference conversion of a `&ModIndex` to a `CachingIndex`, allocating only the cache that a new `CachingIndex` needs to own. The a caching query can again be generically implemented given the Trait bound `H: K2U` and `T: U2Pos`.

```
35 pub struct CachingIndex<'a, H, T> {
36     cache: Cache,
37     k2u: &'a H,
38     u2pos: &'a T
39 }
40 
41 impl<'a, H, T> ModIndex<H, T> {
42     fn as_caching_index(&'a self) -> CachingIndex<'a, H, T> {
43         CachingIndex {
44             cahe: Cache::new(),
45             k2u: &self.k2u,
46             u2pos: &self.u2pos,
47         }
48     }
49 }
50 
51 impl<'a, H, T> CachingIndex<'a, H, T> where
52     H: K2U,
```

```rust
    T: U2Pos
{
    // Generic implementation of locate!
    pub fn locate(&mut self, km: &CanonicalKmer) -> Option<Hits> { ... }
}
```

With locate queries implemented generically for `ModIndex<H,T>`, benchmarking, testing, and validation functionalities can also be implemented generically.

8.3.2 Building and using indexes with `mazu`

We have implemented, `mazu`, a Rust library for modular indexing. This library enables us to modularly construct indexes described in Chapter 6, and allows researchers to easily compose new indexes when new and improved data-structures that support the k-mer-to-unitig query, unitig-to-occurrence query, or unitig-to-color mappings are developed in the future.

Below, we highlight the ease-of-use and interoperability of our implemented library with an illustrative snippet. In this snippet, a `pufferfish` index is loaded, its k-mer-to-unitig mapping is swapped out and replaced by an instance of `SSHash`, a streaming cache is attached, and the index is validated against the reference sequence that it indexes.

- We demonstrate load-only compatibility with the previously published implementation of `pufferfish` [190]. Here, `DenseIndex` is a type alias for a `ModIndex` with matching implementations of `pufferfish`'s minimum perfect hash function and encoding of unitig occurrences. On line 61 a `pufferfish` index, previously built and serialized by the C++ implementation is deserialized into a `ModIndex`.

- On line 62 the deserialized `pufferfish` index is converted to a "streaming index" that implements a caching scheme optimized for querying consecutive k-mers on a given sequence. And on line 64 the index is validated against the reference sequence from which it was built. The function `validate_fasta` queries each k-mer in the given FASTA file and validates that each queried k-mer is mapped to the appropriate position.

- On lines 66 to 78, we demonstrate how a new index can be easily composed by swapping out the `pufferfish` index's k-mer-to-unitig mapping with `SSHash`. We extract the unitig set indexed by the deserialized `pufferfish` index, and construct an `SSHash` on line 69. On line 73, we create a new modular index dropping in `SSHash`. Conversions to streaming indexes and `validate_fasta` are implemented generically. Thus, a cache for streaming queries can be immediately attached and the indexed FASTA sequence immediately validated on line 81.

```
58 fn main() {
59     let fp = YEAST_CHR01_INDEX;
60     let fasta = YEAST_CHR01_FASTA;
61     let pi = DenseIndex::deserialize_from_cpp(fp).unwrap();
62     let streaming_index = pi.as_streaming();
63 
64     streaming_index.validate_fasta(fasta);
65 
66     let minimizer_w = 15;
67     let build_hasher = WyHashState::defualt();
68     let unitig_set = pi.as_ref().clone();
69     let sshash = SSHash::from_unitig_set(unitig_set,
```

```rust
                                            minimizer_w, 32,
                                            build_hasher).unwrap();

    let new_index = ModIndex::from_parts(
        pi.base.clone(),
        sshash,
        pi.as_u2pos().clone(),
        pi.as_refseqs().clone(),
    );

    new_index.validate_fasta(fasta);
    new_index.as_streaming().validate_fasta(fasta);
}
```

Part IV

Conclusion and Back Matter

Chapter 9: Conclusion

In this dissertation, I have developed a series of methods that interrogate and improve genomics analyses at different scales — from factorizing biological networks to indexing huge collections of genomic sequences. One guiding principle throughout has been to not only exploit theoretically interesting opportunities but also to embrace empirically effective methods. For example, in Chapter 3, our simple but thoughtfully constructed matrix factorization models could be more effectively trained and thus outperformed a prior deep-learning model; in Chapter 5 I designed and applied perplexity to enable model selection and assessment of quality in experimental RNA-seq analysis; in Chapter 6 empirical optimizations (e.g., caching) made a theoretically novel sampling scheme also fast in practice; and in Chapter 7 careful and modular composition of arguably simple compression schemes allowed us to build a small and fast state-of-the-art index.

There are two major directions that, in my view, are particularly worth exploring in future work. First, the straightforward next step is to design and implement new and improved algorithms for indexing k-mers and (almost arbitrary) metadata — many interesting algorithmic questions are enumerated in Sections 6.6 and 7.6. There are two kinds of possible, new indexes that are particularly interesting. One is to index *approximate* positions of reference sequences. The indexes `pufferfish2` and `fulgor` are solutions that index the two extremes of positional metadata: one indexes the exact location of each k-mer in each reference and the other indexes only the set of references in which a

k-mer appears. A useful intermediate solution would be to index k-mers binned into quantized and approximate positions on references. Such an index could support new pseudoalignment algorithms that enforce more informative structural constraints to achieve improved specificity at scale. Another is to develop learned optimizations, in terms of both compressibility and speed, for indexes built for specific domains, applications, and workflows. Given expected patterns of queries and workload, one can imagine indexes that better trade-off query time versus space by learning more "optimal" representations of the indexed data. For example, for a frequently repeated workflow that queries k-mers from scATAC-seq data of a particular tissue, one could use `pufferfish2`'s sampling scheme to more densely sample unitigs that correspond to open chromatin regions that are commonly sequenced. Such an index would better compress occurrences of infrequently hit unitigs but maintain fast queries for frequently occurring k-mers.

Second, the possible translational opportunities provided by small efficient indexes ought to be further explored. In Part III of this dissertation, I developed indexing algorithms and demonstrated how they scale to huge collections of reference sequences — I have argued for their utility by benchmarking huge instances on machines with hundreds of gigabytes of memory. However, one unexplored consequence of having efficient indexes is that they may make analysis of high-throughput sequencing data *accessible*. Namely, our methods can index moderately sized collections (e.g., on the order of hundreds or few thousand genomes) with indexes that are only tens of gigabytes large. These indexes can be queried and loaded by modern laptops and are not restricted to use in large compute nodes. One should explore how such indexes can be deployed for applications, researchers, and practitioners in resource limited settings. These indexes may benefit application areas in which decentralized and rapidly deployable tools are favorable — for example in the early detection of disease in wastewater [245], or in the monitoring for global food safety [246]. Importantly, small and efficient indexes enable the application of methods in resource limited settings and may help make collaborative science also more equitable and help mitigate the practice of “parachute science” [247].

It is my hope that this dissertation, culminating in a modular library for implementing fast and efficient k-mer indexes, will help researchers correctly implement and explore new algorithms and develop more accessible tools. Perhaps, these efficient k-mer indexes and tools will make next generation’s NGS analysis possible not only on compute clusters of prestigious institutions, but also on moderately priced laptops of researchers in the lab and practitioners in the field.

Acknowledgements and Funding

Funding

The past and ongoing work would not be possible without the support of various funding agencies and organizations to JF and collaborators.

The majority of past and ongoing work was made in large part possible by funding from the National Science Foundation (NSF). In particular, JF was funded by National Science Foundation (NSF) award DGE-1632976 in 2020 as part of the UMD *Computation and Mathematics for Biological Networks* (COMBINE) program — a NSF-funded Research Traineeship program; and by NSF award DGE-1840340 from 2020 to 2023 as part of the NSF Graduate Research Fellowship Program (NSF GRFP).

Additional funding and collaborator contribution declarations

Fan developed and completed [13] under the co-supervision of Crovella, Hescott, and Leiserson. We sincerely thank our collaborators Cannistra, Fried, Lim, and Schaffner who developed and devised the initial methods and experiments prior to the study's completion. This work was funded in part: by NSF IIS-1421759 and CNS-1618207 to Crovella; and Boston University Undergraduate Research Opportunities Program to Lim. The funding for open access was provided by UMD, including in part by the UMD Open Access Publishing Fund.

Fan developed and implemented [14] under the supervision of Leiserson. This study was additionally supervised by Crovella, and was implemented in collaboration with Li.This work was also supported by NSF award CNS-1618207 to M.C., and in part by grant number [2018-182608] from the Chan Zuckerberg Initiative Donor-Advised Fund, an advised fund of Silicon Valley Community Foundation.

Franzese devised and implemented of [17]. The method and manuscript for this study was developed in collaboration with Fan. The study was supervised by Leiserson and Sharan. The study was supported in part, by a grant from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel; NSF GRFP award DGE-1842165 to NF; and by the Intramural Research Program of the National Cancer Institute, Center for Cancer Research, NIH.

Fan developed and implemented [16] under the supervision of Patro. Chen implemented some experiments. This work was also supported by NIH R01 HG009937, and by NSF CCF-1750472, and CNS-1763680.

Fan developed and implemented [182] under the supervision of Patro. This work is supported by the NIH under grant award numbers R01HG009937 to Patro; the NSF awards CCF-1750472 and CNS-1763680 to Patro; and NSF award No. DGE-1840340 to Fan. Pibiri was partially supported by the project MobiDataLab (EU H2020 RIA, grant agreement №101006879).

Fan protoyped and developed [182] under the supervision of Patro and Pibiri. The wonderfully fast implementation was implemented by Pibiri. This work is supported by the NIH under grant award numbers R01HG009937 to R.P.; the NSF awards CCF-1750472 and CNS-1763680 to R.P. Funding for this research has also been provided by the European Union's Horizon Europe research and innovation program (EFRA project, Grant Agreement Number 101093026). This work was also partially supported by DAIS — Ca' Foscari University of Venice within the IRIDE program.

Conflicts of interest

MDML supervised [14] and was a paid consultant for Microsoft during the time when part of the study was performed. Patro is a co-founder of Ocean Genomics Inc.

Appendix A: Spectrum preserving tilings and `pufferfish2`

A.1 The mapped position query (MRP) for unitig-tilings

Algorithm 4: The MRP query for unitig-tilings

```
def mrp($x$):
    $tup \leftarrow$ k2u($x$)
    if $tup = \emptyset$ then
        return [ ]
    $(i, p) \leftarrow tup$
    $occ_i \leftarrow$ num-occs($U_i$)
    $ans \leftarrow$ [ ]
    for $r \leftarrow 0$ to $occs_i$ do
        $(n, s) \leftarrow$ u2occ($i, r$)
        $ans[r] = (n, s + p)$
    return ans
```

A.2 Spectrum preserving tilings with *orientations*

We extend the definition of spectrum preserving tilings (without orientations) given in Section 6.3.1, to formally define spectrum preserving tilings (SPT) *with orientations*. An SPT with orientation allows tiles (members of a spectrum preserving string set) to occur in either a *forward* orientation as stored in memory as a nucleotide sequence, or a *backwards* orientation as the *reverse complement* of the stored sequence.

With respect to representing reference genomic sequences, using SPTs with orientations is particularly useful because it avoids redundantly encoding and storing occurrences of a k-mer *and* the reverse complement of said k-mer. Furthermore, since most sequencing technologies are agnostic to strands of DNA sequences, considering orientations enables the simultaneous and canonical representation of both corresponding strands of an indexed genomic sequence.

Also, as in [190], we consider only *odd* k-mer sizes so that no k-mer is its own reverse complement.

***.** Tiling sequences of tile *and orientation* pairs. Given a fixed k-mer size, k, a tiling sequence T_n in $\mathcal{T}$ is instead sequences of *tile-orientation* pairs where each occurrence is defined to be $T_{n,m} = (U_i, o)$, for some unitig $U_i \in \mathcal{U}$ and an orientation $o \in \{0, 1\}$. Here, $o = 1$ indicates that the unitig U_i occurs in a forward orientation and $o = 0$ indicates that it occurs in the *backwards* orientation with reverse complement sequence $\overline{U_i}$. Notationally, $\overline{U_i}$ is the string that is the reverse complement of U_i where U_i is reversed and each nucleotide is replaced with its complement.

Let us define the $\texttt{spell}(U_i, o)$ function for a unitig orientation pair to return the forward sequence U_i if $o = 1$ and the backwards, reverse complement sequence $\overline{U_i}$ otherwise. Abusing some notation, when $T_{n,m} = (U_i, o)$, let $\texttt{spell}(T_{n,m}) = \texttt{spell}(U_i, o)$.

Then formally, a spectrum preserving tiling with orientations for a reference collection $\mathcal{R} = \{R_1, \dots, R_N\}$ tiles each reference sequence R_n with sequences *spelled by* occurrences of tile-orienation pairs. Specifically, each R_n can be reconstructed by gluing together the sequences that tile-orientation pairs *spell*. For each R_n that is tiled by M_n tile occurrences, the SPT satisfies the property that:

$$R_n = \texttt{spell}(T_{n,1})[w_{n,1} : w_{n,1} + l_{n,1}] \oplus_k \dots \oplus_k \texttt{spell}(T_{n,M_n})[w_{n,M_n} : w_{n,M_n} + l_{n,M_n}]$$

A.2.1 Returning queries with orientations

Accordingly, when indexing an SPT with orientations the mapped reference position query, k-mer-to-tile query, and tile-to-occurrence query also return orientations. Here, we extend and reintroduce the queries defined in Section 6.3.

1. **The mapped reference position (MRP) query** Given any k-mer x, the MRP query enumerates the positions and orientations of all occurrences of x in $\mathcal{R}$. Precisely, each returned occurrence is a tuple (n, p, o), that specifies that k-mer x occurs in reference n at position p with orientation o. That is, if $o = 1$, then x occurs in the forward orientation as $R_n[p : p + k] = x$. Otherwise, the reverse complement occurs as $R_n[p : p + k] = \overline{x}$. If a k-mer does not occur in some $R_n \in \mathcal{R}$, the query returns an empty list.

2. **The kmer-to-tile query**: Given a k-mer x, $\texttt{k2tile}(x)$ returns (i, p, o) — the identity of the tile U_i that contains x, the offset (position) into the tile U_i where x occurs, and the *orientation* of how x occurs. That is, $\texttt{k2tile}(x) = (i, p, 1)$ if $U_i[p : p + k] = x$, and $\texttt{k2tile}(x) =$

$(i, p, 0)$ if $U_i[p : p + k] = \overline{x}$ where x occurs in the backwards orientation as the reverse complement. If x is not in $\mathcal{R}$, `k2tile`, `k2tile`(x) returns $\emptyset$.

3. **The tile-to-occurrence query**: Given the r-th occurrence of the tile U_i, `tile2occ`(i, r) returns the tuple (n, o, s, w, l) that encodes how *and in what orientation* U_i tiles the reference R_n. Let the r-th occurrence of U_i be a tile-occurrence $T_{n,m}$ on $\mathcal{T}$ where $T_{n,m} = U_i, o$ for some orientation o. Then `tile2occ`(i, r) returns $(n, o, s_{n,m}, w_{n,m}, l_{n,m})$.

   When `tile2occ`$(i, r) = (n, o, s, w, l)$ and $o = 1$, the r-th occurrence of U_i occurs on R_n at position $(s + w)$, with the sequence $U_i[w : w + l]$. When `tile2occ`$(i, r) = (n, o, s, w, l)$ and $o = 0$, the r-th occurrence of U_i occurs on R_n at position $(s + w)$, with the sequence $\overline{U_i}[w : w + l]$.

With some arithmetic bookkeeping considering orientations and lengths, the MRP query with orientations can again be decomposed into the two corresponding k-mer-to-tile and tile-to-occurrence queries that also return orientations. Although not introduced with respect to an SPT, the `pufferfish` index developed by Almodaresi et al. [190] is implemented exactly this way as an index over an SPT *with orientations* of unitigs.

A.3 `Pufferfish2`: the `pred` query with orientations

`Pufferfish2`'s sampling scheme and the `pred` query can be applied when considering orientations — our implemented tool does exactly this. Below, we extend Section 6.4 to fully specify the introduced sampling scheme and the `pred` query when orientations are considered.

A.3.1 Predecessor and successor nucleotides

When SPT references, predecessor and successor nucleotides are defined and obtained with respect to sequences on the *references*. Specifically, the predecessor nucleotide is the first nucleotide of the last k-mer on of the preceding unitig-occurrence *as spelled with the corresponding orientation* of the occurrence. The successor nucleotide is defined in the same manner.

Suppose $T_{n,m} = (U_i, o)$, and $T_{n,m-1} = (U_j, \omega)$, and let the unitigs have lengths ℓ_i and ℓ_j, respectively. We say that, $T_{n,m-1}$ precedes $T_{n,m}$ with predecessor nucleotide p *and orientation* o. Concretely, p is the first nucleotide on the last k-mer of the preceding unitig, with $p = \texttt{spell}(T_{n,m-1})[\ell_j - k]$. We say that, $T_{n,m}$ succeeds $T_{n,m-1}$ with successor nucleotide s *and orientation* ω. Accordingly, the successor nucleotide, s, is the last nucleotide on the first k-mer of the succeeding unitig, with $s = \texttt{spell}(T_{n,m})[k]$.

A.3.2 Storing nucleotide-orientation pairs in `ptab` and `stab`

Instead of storing only nucleotides, `pufferfish2` stores nucleotide-orientation pairs in implementation. That is, for each occurrence $T_{n,m} = (U_i, o)$ that is the r-th occurrence of a not-sampled unitig U_i,

$$\texttt{ptab}[i][r] = (\text{pred}_{\text{n}}(T_{n,m}), o).$$

And for each occurrence $T_{n,m} = (U_i, o)$ that is the r-th occurrence of *any* unitig U_i,

$$\texttt{stab}[i][r] = (\text{succ}_{\text{n}}(T_{n,m}), o).$$

In summary, `ptab` and `stab` store for each corresponding unitig-occurrence, the nucleotides that succeed and precede it as they occur on a tiled reference, *and* the orientation of said occurrence.

A.3.3 Computing the `pred` query by matching ranks of predecessor-orientation and successor-orientation pairs

When orientations are considered, computing the `pred` query requires matching ranks of predecessor-orientation and successor-orientation pairs. Critically, any time a pair of unitigs occur as a successor-predecessor pair in *fixed* orientations, the corresponding pair of predecessor and successor nucleotides are *consistent* and also *fixed*. Furthermore, if an occurrence of the U_j in orientation ω precedes a unitig U_i with orientation o, *any other* occurrence of U_j that precedes U_i with orientation o must also occur with orientation ω. We state and prove this property with Theorem A.1 and illustrate examples of both possible and impossible unitig-occurrences with Fig. A.1.

Theorem A.1 guarantees that whenever U_i occurs with orientation o with predecessor nucleotide p preceded by U_j, U_j must occur with fixed orientation ω with a fixed successor nucleotide s. We illustrate this correspondence in Fig. A.1. Algorithm 5 implements `pred` with orientations considered.

To find the identity, j, of the preceding unitig occurrence, Algorithm 5 must construct the last k-mer of the corresponding occurrence U_j as it appears on the reference. Specifically, in Line 3 it spells U_i before extracting the overlapping $(k-1)$-mer. Here, `k2u` returns orientation, ω, of the queried k-mer on U_j, which must also be the orientation of the preceding unitig occurrence on the reference. Furthermore, the successor nucleotide, s, for the preceding occurrence of U_j must be the k-th nucleotide on U_i spelled with orientation o (Line 5).

Now, Algorithm 5 has all it needs to compute q, the unitig-rank of the preceding occurrence of U_j. Computing the rank of (p, o) in ptab$[i]$ yields the rank of the corresponding successor-orientation pair stored for the preceding unitig-occurrence. Finally, selecting for the successor-orientation pair (s, ω) in stab$[j]$ yields q.

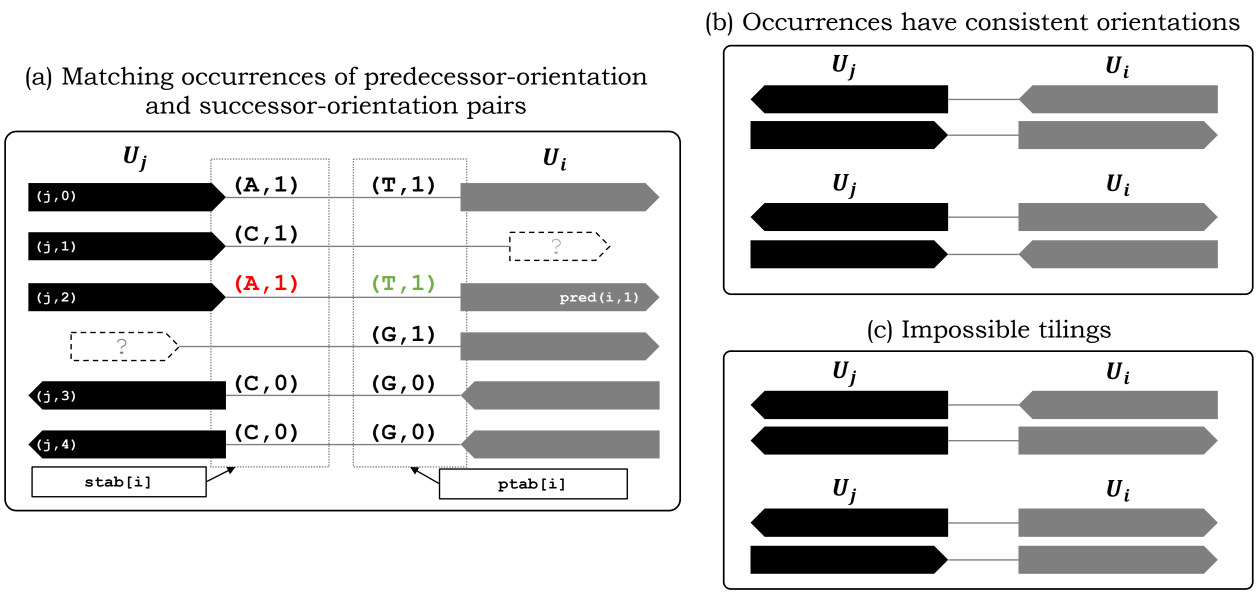


Figure A.1: Properties of the pred query for unitig-tilings with orientations. (a) Adjacent pairs of successor and predecessor unitigs have consistent and unique co-occurring pairs of predecessor nucleotide-orientation successor nucleotide-orientation pairs. (b) Whenever a pair of unitigs occur adjacently on the tiling, the orientation of *one* fixes the orientation of the other (for odd k-mer sizes). (c) That is, if U_j with orientation ω precedes a unitig U_i with fixed orientation o once, it cannot precede another occurrence (of U_i with orientation o) in the opposite orientation.

Algorithm 5: The `pred` query with orientations

```
def pred($i, r$):
    $(p, o) \leftarrow \texttt{ptab}[i][r]$
    $y \leftarrow p \circ \texttt{spell}(U_i, o)[: k-1]$
    $(j, \_, \omega) \leftarrow \texttt{k2u}(y)$
    $s \leftarrow \texttt{spell}(U_i, o)[k]$
    $t \leftarrow \texttt{rank}_{(p,o)}(\texttt{ptab}[i], r)$
    $q \leftarrow \texttt{select}_{(s,\omega)}(\texttt{stab}[j], t)$
    return $(j, q)$
```

A.3.4 Unitig-unitig occurrences have consistent orientations and predecessor-successor nucleotides

The key to the correctness of `pufferfish2`'s reference tiling traversal, by way of successor-orientation and predecessor-orientation pairs, is that predecessor-successor nucleotide pairs for adjacent unitig-occurrences are consistent and unique up to orientation. Whenever unitigs U_a and U_b overlap and tile with some given *fixed* orientations, corresponding successor and predecessor nucleotides are consistent and always the same. Below, we prove Theorem A.1 that formally states this property.

Theorem A.1. *Let unitigs U_a and U_b overlap and tile in orientations o and ω, with successor and predecessor nucleotides p and s. If any occurrence of U_a with orientation o is preceded by the nucleotide p, it must always be preceded by the same unitig U_b in the same orientation ω. Simultaneously, if any unitig U_b with orientation ω is succeeded by the nucleotide s, it must always* *be succeeded by the same unitig U_a in the same orientation o.*

Proof. Theorem A.1 is result of the lemmas proved below. Lemmas A.1 and A.2 state that with fixed orientations and predecessor and successor nucleotides, the identities of successor-predecessor unitig pairs must be unique. Lemmas A.3 and A.4 state that with fixed predecessor and successor nucleotides for fixed unitig identities, the orientations of a successor-predecessor unitig pair must be unique. □

Lemma A.1. *Consider unitigs $U_i, U_j, U_k \in \mathcal{U}$. Let adjacent unitig occurrences $T_{a,b} = (U_i, o)$ and $T_{a,b+1} = (U_j, \omega)$ occur with successor nucleotide s. For any c, d, there does not exist another pair of adjacent occurrences $T_{c,d} = (U_i, o)$ and $T_{c,d+1} = (U_k, \omega')$ with the same succeeding nucleotide s but with $U_j \neq U_k$.*

Proof. Let us assume the contrary. Let z be the last $(k-1)$-mer on $\texttt{spell}(T_{a,b})$, which is the same as $\texttt{spell}(T_{c,d})$. Then the k-mer $z \circ s$ occurs on different unitigs U_j and U_k. However, this is a contradiction since any unique k-mer occurs in only one unique unitig. □

Lemma A.2. *Consider unitigs $U_i, U_j, U_k \in \mathcal{U}$. Let the occurrences $T_{a,b} = (U_i, o)$ and $T_{a,b-1} = (U_j, \omega)$ occur with preceding nucleotide p. There does not exist another pair $T_{c,d} = (U_i, o)$, $T_{c,d-1} = (U_k, \omega')$ in $\mathcal{R}$ where $U_j \neq U_k$, with the same preceding nucleotide s.*

Proof. This is symmetrical to Lemma A.1. □

Lemma A.3. *Let $\{U_i, U_j\} \in \mathcal{U}$ Given unitig occurrences $T_{a,b} = (U_i, o)$ and $T_{a,b+1} = (U_j, 1)$ that tile R_a with successor nucleotide s. There does not exist another pair $T_{c,d} = (U, o)$, $T_{c,d+1} = (U_j, 0)$ in $\mathcal{R}$ with the same successor nucleotide s.*

Proof. Let us assume the contrary. Let z be the last $(k-1)$-mer on $T_{a,b}$ and $T_{c,d}$. Suppose $z \circ s$ is the first k-mer on U_i. The tiling on R_c implies that $\overline{z \circ s}$ is the first k-mer on $\overline{U}_j$ and that $z \circ s$ is the last k-mer on U_j. But the tiling on R_a implies that $z \circ s$ is the first k-mer on U_j. If $|U_j| = k$ and U_j is itself a k-mer, then the above implies $U_j = \overline{U_i}$. This cannot be the case, since we consider only odd-length k-mers, and no odd length k-mer can be equal to its reverse complement. If $|U_j| > k$, then z occurs in two distinct positions in U_j, this is again a contradication since any unique k-mer occurs in only one unique unitig. □

Lemma A.4. *Let $\{U_i, U_j\} \in \mathcal{U}$ Given unitig occurrences $T_{a,b} = (U, o)$ and $T_{a,b-1} = (V, 1)$ that tile R_a with predecessor nucleotide p. There does not exist another pair $T_{c,d} = (U, o)$, $T_{c,d-1} = (V, 0)$ in $\mathcal{R}$ with the same precedecessor nucleotide p.*

Proof. This is symmetrical to Lemma A.3. □

A.4 Constructing `pufferfish2` from `pufferfish`

Building `pufferfish2` requires a linear scan over the n total unitig-occurrences in the tiling sequences indexed by a given `pufferfish` index to collect predecessor and successor nucleotides. The construction process is dominated by the time it takes to build the pair of wavelet matrices over all the predecessor and successor nucleotides of every unitig-occurrence. We note that since `pufferfish2` sparsifies and compresses and existing `pufferfish` index, it adopts `pufferfish`'s upstream preprocessing of unknown bases, where each `N` is replaced by a pseudo-random nucleotide. Constructing a wavelet matrix over an alphabet of size σ requires $O(n \lg \sigma)$ time and amounts to successive stable partitions of the encoded characters according to their bitwise representations. In the future, we plan to update `pufferfish2` to index input reference sequences directly.

A.5 Greedy unitig sampling with bounded traversal length s

Here we describe a *greedy* sampling scheme that greedily bounds traversal lengths to be at most of length s. To ensure that all backwards traversals terminate, the greedy sampling with integer paramater s first samples all unitigs that occur as the first occurrence of a tiling sequence, adds them to the set of sampled unitigs $\mathcal{U}_S$, and sets the corresponding bits in `isSamp` to 1 and all other bits to zero. Then, traversing tiling sequences in the order in which they appear, the greedy scheme maintains a counter of the distance to the last sampled unitig-occurrence. At each unitig occurrence $T_{n,m} = U_i$, if U_i is already sampled (i.e., `isSamp`$[i]$ is 1), the greedy scheme resets the counter to zero. Otherwise, the greedy scheme increases the counter by one. When the counter is greater than s, it samples the current unitig and resets the counter.

Although the greedy scheme is able to explicitly bound the traversal length it samples almost *all* unitigs when the length of tiling sequences become much larger than the number of unique unitigs. This is because, as implemented, `pufferfish2` samples *all occurrences* of a unitig, if said unitig is sampled. For example, when applying this sampling scheme to index a collection of seven human genomes, a greedy scheme with $s = 3$ samples 40% of unique unitigs that constitute more than 70% of unitig-*occurrences*. In this example, over 70% of `utab` must then be kept and uncompressed.

A.6 Optimizations for `pufferfish2`

***.** Caching traversals. When enumerating all positions of a unitig with `u2occ`, `pufferfish2` caches the `k2u` query — the empirically slowest constant-time operation in the `pred` query (Line 4 in Algorithm 3). The purpose of this `k2u` query is only to find the *identity* of the preceding unitig given a unitig-occurrence's predecessor nucleotide. While a unitig may be preceded by *many* occurrences, preceding occurrences can have *at most* four unique unitig identities — one for each possible nucleotide. If U_i occurs more than once with U_j preceding it, U_j must always precede U_i with the same fixed predecessor nucleotide each time. For MRP queries, `pufferfish2` can avoid executing the redundant `k2u` queries (within `pred` queries) when the *same* nucleotide is prepended to different occurrences of U_i. Specifically, during MRP queries where the `pred` query is executed, `pufferfish2` caches the mapping from predecessor nucleotides to preceding unitig identities. In practice, `pufferfish2` maintains an efficient LRU cache to memoize Lines 3 and 4 in Algorithm 3.

*. Caching streaming MRP queries. In practice, a *stream* of successive MRP queries for different k-mers often land in the same unitig (e.g., when querying k-mers on a sequenced read). So, instead of performing redundant `u2occ` queries that may perform backwards traversals for the same unitig, `pufferfish2` maintains a cache for the `u2occ` query. When successive k-mers are found to be in the same unitig via the `k2u` query, `pufferfish2` checks a "streaming cache" to avoid performing repeated `u2occ` queries for the same unitig. This caching scheme for "streaming" queries is also employed in [23].

*. Exiting early. In practice, programs such as read-mappers and aligners can exit early from the mapped reference position query if a queried k-mer is uninformative and occurs too frequently. With `pufferfish2`, instead of always computing the `u2occ` query for every occurrence in loop starting on Line 8, a caller of the MRP query can exit before the loop and avoid traversals altogether.

*. Interpolating between wavelet matrices and short linear scans. In practice, computing rank and select for short predecessor and successor nucleotide sequences (see Lines 6 and 7) is faster with a linear scan in an array than an operation in the wavelet matrix. So, for unitigs that occur at most 64 times, `pufferfish2` stores corresponding lists of nucleotides in packed arrays instead of

wavelet matrices.

A.7 Cost estimate for Amazon Web Services (AWS) EC2 instances

Estimated prices for AWS EC2 instances in the "US East" region are obtained from `https://calculator.aws/#/estimate`. EC2 instances were specified with 500Gb of storage and 8 CPUs for 10 hrs per week of usage. Recommended EC2 `x2gd.4xlarge` instances have 258GiB of memory whereas `x2gd.2xlarge` instances have 128GiB of memory. As of writing, estimated cost per month for `x2gd.4xlarge` and `x2gd.2xlarge` are 651USD and 351USD, respectively.

Appendix B: Pseudoalginment algorithms

Algorithm B1: The Full-Intersection algorithm for a query sequence Q. The algorithm uses the three index components: $\mathcal{D}$ (the dictionary, mapping k-mers to unitigs), B (the bit-vector mapping from unitigs to colors), and $\mathcal{L}$ (the inverted index storing the compressed colors). As discussed in Section 7.3.1, the dictionary $\mathcal{D}$ can stream through the query sequence Q and collect unitig ids. The inverted index $\mathcal{L}$, instead, returns an iterator over a color set given the color id c as Iterator(c).

```
def Full-Intersection(Q):
    if $|Q| < k$ then
        return $\emptyset$
    $U = \mathcal{D}$.Stream-Through$(Q)$;                // $U$ is the set of unitig ids.
    Deduplicate$(U)$
    $C = \emptyset$ ;                                      // $C$ is the set of color ids.
    for $u \in U$ do
        $c = B$.Color-ID$(u)$
        $C$.Add$(c)$
    Deduplicate$(C)$ $I = \emptyset$ ;          // $I$ is the set of iterators over colors.
    for $c \in C$ do
        $i = \mathcal{L}$.Iterator$(c)$
        $I$.Add$(i)$
    $R =$ Intersect$(I)$ ;                  // $R$ is the result set of reference ids.
```

Algorithm B2: The Intersect algorithm for a set of iterators $I = \{i_1, ..., i_p\}$. An iterator object supports three primitive operations: Value(), returning the value currently pointed to by the iterator; Next(), returning the value immediately after the one currently pointed to by the iterator; Next-GEQ(x), returning the smallest value that is larger-than or equal-to x. We assume that if i is an iterator over color C_j then calling i.Next() for more than $|C_j|$ times will return the (invalid) reference id $N + 1$.

```
def Intersect(I):
    if $I = \emptyset$ then
        return $\emptyset$
    $R = \emptyset$
    $candidate = i_1$.Value()
    $j = 2$
    while candidate $\leq N$ do
        for ; $j \leq p$; $j = j + 1$ do
            $i_j$.NextGEQ($candidate$)
            $v = i_j$.Value()
            if v $\neq$ candidate then
                $candidate = v$
                $j = 1$
                break
    return $R$
```

Algorithm B3: The Threshold-Union algorithm for a query sequence Q. Differently from the Full-Intersection method (Algorithm B1), here U, C, and I, are sets of pairs. The first component of a pair is a unitig id, a color id, or an iterator, respectively if the pair is in U, C, or U. The second component, read by calling the method Score() in the pseudocode, is the number of positive k-mers that have a given unitig id or have a given color. The score of iterator i is the score of the color id c if $i = \mathcal{L}$.Iterator(c). Clearly, when deduplicating the sets U and C, the scores of equal unitig or color ids must be summed.

```
def Threshold-Union($Q, \tau$):
    if $|Q| < k$ then
        return $\emptyset$
    $U = \mathcal{D}$.Stream-Through$(Q)$ ;                    // $U$ is the set of unitig ids.
    $|K(Q)| = \sum_{u \in U} u$.Score() ;      // $|K(Q)|$ is the number of positive hits.
    Deduplicate-And-Sum-Scores$(U)$
    $C = \emptyset$ ;                                 // $C$ is the set of color class ids.
    for $u \in U$ do
        $c = B$.Color-ID$(u)$
        $C$.Add$(c)$
    Deduplicate-And-Sum-Scores$(C)$
    $I = \emptyset$ ;                        // $I$ is the set of iterators over color sets.
    for $c \in C$ do
        $i = \mathcal{L}$.Iterator$(c)$
        $I$.Add$(i)$
    $t = |K(Q)| \times \tau$ ; // A reference is returned iff it contains at least $t$
    $k$-mers.
    $R =$ Union$(I, t)$ ;                  // $R$ is the result set of reference ids.
    return $R$
```

Algorithm B4: The Union algorithm for a set of iterators $I = \{i_1, \dots, i_p\}$ and minimum score t.

```
def Union:
    $I, t$
if $I = \emptyset$ then
    return $\emptyset$
$R = \emptyset$
candidate $= \min\{i_1.\text{Value}(), \dots, i_p.\text{Value}()\}$
while candidate $\leq N$ do
    min $= N + 1$
    score $= 0$
    for $j = 1$; $j \leq p$; $j = j + 1$ do
        if $i_j$.Value() $=$ candidate then
            score $=$ score $+ i_j$.Score()
            $i_j$.Next()
        if $i_j$.Value() $<$ min then
            min $= i_j$.Value()
    if score $\geq t$ then
        $R$.Add(candidate)
    candidate $=$ min
return $R$
```

Bibliography

[1] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011):931–945, 2004. ISSN 0028-0836. doi: 10.1038/nature03001.

[2] The GTEx Consortium. The GTEx Consortium atlas of genetic regulatory effects across human tissues. *Science*, 369(6509):1318–1330, 2020. doi: 10.1126/science.aaz1776. URL `https://www.science.org/doi/abs/10.1126/science.aaz1776`.

[3] Katherine A. Hoadley, Christina Yau, Toshinori Hinoue, Denise M. Wolf, Alexander J. Lazar, Esther Drill, et al. Cell-of-origin patterns dominate the molecular classification of 10,000 tumors from 33 types of cancer. *Cell*, 173(2):291–304.e6, 2018. ISSN 0092-8674. doi: 10.1016/j.cell.2018.03.022.

[4] Wouter Saelens, Robrecht Cannoodt, Helena Todorov, and Yvan Saeys. A comparison of single-cell trajectory inference methods. *Nature Biotechnology*, 37(5):547–554, 2019. ISSN 1087-0156. doi: 10.1038/s41587-019-0071-9.

[5] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, et al. Gene Ontology: tool for the unification of biology. 25(1):25–29. ISSN 1061-4036. doi: 10.1038/75556. URL `http://dx.doi.org/10.1038/75556`.

[6] Rohit Singh, Jinbo Xu, and Bonnie Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. 105(35):12763–12768.

[7] Chung-Shou Liao, Kanghao Lu, Michael Baym, Rohit Singh, and Bonnie Berger. Isorankn: spectral methods for global alignment of multiple protein networks. 25(12):i253–i258.

[8] Tijana Milenković, Weng Leong Ng, Wayne Hayes, and Nataša Pržulj. Optimal network alignment with graphlet degree vectors. 9:121–137. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2901631/`.

[9] Noël Malod-Dognin and Nataša Pržulj. L-graal: Lagrangian graphlet-based network aligner. page btv130.

[10] Behnam Neyshabur, Ahmadreza Khadem, Somaye Hashemifar, and Seyed Shahriar Arab. Netal: a new graph-based method for global alignment of protein–protein interaction networks. 29(13):1654–1662.

[11] Somaye Hashemifar and Jinbo Xu. Hubalign: an accurate and efficient method for global alignment of protein–protein interaction networks. 30(17):i438–i444.

[12] Rob Patro and Carl Kingsford. Global network alignment using multiscale spectral signatures. *Bioinformatics*, 28(23):3105–3114, 10 2012. ISSN 1367-4803. doi: 10.1093/bioinformatics/bts592. URL `https://doi.org/10.1093/bioinformatics/bts592`.

[13] Jason Fan, Anthony Cannistra, Inbar Fried, Tim Lim, Thomas Schaffner, Mark Crovella, Benjamin Hescott, and Mark D M Leiserson. Functional protein representations from biological networks enable diverse cross-species inference. *Nucleic Acids Research*, 47(9): gkz132–, 2019. ISSN 1362-4962. doi: 10.1093/nar/gkz132.

[14] Jason Fan, Xuan Cindy Li, Mark Crovella, and Mark D M Leiserson. Matrix (factorization) reloaded: flexible methods for imputing genetic interactions with cross-species and side information. *Bioinformatics*, 36(Supplement_2):i866–i874, 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btaa818.

[15] Mark D M Leiserson, Fabio Vandin, Hsin-Ta Wu, Jason R Dobson, Jonathan V Eldridge, Jacob L Thomas, Alexandra Papoutsaki, Younhun Kim, Beifang Niu, Michael McLellan, Michael S Lawrence, Abel Gonzalez-Perez, David Tamborero, Yuwei Cheng, Gregory A Ryslik, Nuria Lopez-Bigas, Gad Getz, Li Ding, and Benjamin J Raphael. Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes. *Nature Genetics*, 47(2):106–114, 2015. ISSN 1061-4036. doi: 10.1038/ng.3168.

[16] Jason Fan, Skylar Chan, and Rob Patro. Perplexity: evaluating transcript abundance estimation in the absence of ground truth. *Algorithms for Molecular Biology*, 17(1):6, 2022. ISSN 1748-7188. doi: 10.1186/s13015-022-00214-y.

[17] Nicholas Franzese, Jason Fan, Roded Sharan, and Mark D.M. Leiserson. Scalpelsig designs targeted genomic panels from data to detect activity of mutational signatures. *Journal of Computational Biology*, 29(1):56–73, 2022. doi: 10.1089/cmb.2021.0453. URL `https://doi.org/10.1089/cmb.2021.0453`. PMID: 34986026.

[18] Daniel J. Nasko, Sergey Koren, Adam M. Phillippy, and Todd J. Treangen. RefSeq database growth influences the accuracy of k-mer-based lowest common ancestor species identification. *Genome Biology*, 2018. doi: 10.1186/s13059-018-1554-6.

[19] Avi Srivastava, Laraib Malik, Hirak Sarkar, Mohsen Zakeri, Fatemeh Almodaresi, Charlotte Soneson, Michael I. Love, Carl Kingsford, and Rob Patro. Alignment and mapping methodology influence transcript abundance estimation. *Genome Biology*, 21(1):239, 2020. ISSN 1474-7596. doi: 10.1186/s13059-020-02151-8.

[20] Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4): 417–419, 2017. ISSN 1548-7091. doi: 10.1038/nmeth.4197.

[21] Amatur Rahman and Paul Medevedev. Representation of k-mer sets using spectrum-preserving string sets. *Journal of Computational Biology*, 28(4):381–394, 2021. doi: 10.1089/cmb.2020.0431.

[22] Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de Bruijn graphs. *Genome Biology*, 22(1):1–24, 2021.

[23] Giulio Ermanno Pibiri. Sparse and skew hashing of k-mers. *Bioinformatics*, 38 (Supplement_1):i185–i194, 06 2022. ISSN 1367-4803. doi: 10.1093/bioinformatics/btac245.

[24] The UniProt Consortium. Uniprot: the universal protein knowledgebase. 45(D1):D158–D169. doi: 10.1093/nar/gkw1099. URL `http://dx.doi.org/10.1093/nar/gkw1099`.

[25] Sebastian M.B. Nijman. Synthetic lethality: General principles, utility and detection using genetic screens in human cells. 585(1):1–6. ISSN 1873-3468. doi: 10.1016/j.febslet.2010.11.024. URL `http://dx.doi.org/10.1016/j.febslet.2010.11.024`.

[26] Nigel J O'Neil, Melanie L Bailey, and Philip Hieter. Synthetic lethality and cancer. *Nature Reviews Genetics*, 2017. ISSN 1471-0056. doi: 10.1038/nrg.2017.47. URL `http://dx.doi.org/10.1038/nrg.2017.47`.

[27] Kriston L McGary, Tae Park, John O Woods, Hye Cha, John B Wallingford, and Edward M Marcotte. Systematic discovery of nonobvious human disease models through orthologous phenotypes. 107(14):6544–6549. ISSN 0027-8424. doi: 10.1073/pnas.0910200107. URL `http://dx.doi.org/10.1073/pnas.0910200107`.

[28] Adam Frost, Marc G Elgort, Onn Brandman, Clinton Ives, Sean R Collins, et al. Functional repurposing revealed by comparing S. pombe and S. cerevisiae genetic interactions. 149 (6):1339–1352. ISSN 0092-8674. doi: 10.1016/j.cell.2012.04.028. URL `http://dx.doi.org/10.1016/j.cell.2012.04.028`.

[29] Assen Roguev, Sourav Bandyopadhyay, Martin Zofall, Ke Zhang, Tamas Fischer, et al. Conservation and rewiring of functional modules revealed by an epistasis MAP in fission yeast. *Science*, 322(5900):405–410, 8 2008. ISSN 0036-8075. doi: 10.1126/science.1162609. URL `http://dx.doi.org/10.1126/science.1162609`.

[30] Mark G. F. Sun, Martin Sikora, Michael Costanzo, Charles Boone, and Philip M. Kim. Network evolution: rewiring and signatures of conservation in signaling. 8(3):e1002411, . ISSN 1553-734X. doi: 10.1371/journal.pcbi.1002411. URL `http://dx.doi.org/10.1371/journal.pcbi.1002411`.

[31] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[32] Eric W Sayers, Tanya Barrett, Dennis A Benson, Evan Bolton, Stephen H Bryant, et al. Database resources of the national center for biotechnology information. 39(suppl 1):D38–D51.

[33] Hitomi Hasegawa and Liisa Holm. Advances and pitfalls of protein structural alignment. 19 (3):341–348. ISSN 0959-440X. doi: 10.1016/j.sbi.2009.04.003. URL `http://dx.doi.org/10.1016/j.sbi.2009.04.003`.

[34] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[35] Aditya Grover and Jure Leskovec. Node2Vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939754. URL `http://doi.acm.org/10.1145/2939672.2939754`.

[36] Hyunghoon Cho, Bonnie Berger, and Jian Peng. Compact integration of multi-network topology for functional analysis of genes. 3(6):540–548.e5. ISSN 2405-4712. doi: 10.1016/j.cels.2016.10.017. URL `http://dx.doi.org/10.1016/j.cels.2016.10.017`.

[37] Vladimir Gligorijevic, Meet Barot, and Richard Bonneau. deepNF: Deep network fusion for protein function prediction. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty440. URL `http://dx.doi.org/10.1093/bioinformatics/bty440`.

[38] Alexandra Jacunski, Scott J Dixon, and Nicholas P Tatonetti. Connectivity homology enables inter-species network models of synthetic lethality. *PLOS Computational Biology*, 11(10): e1004506, 2015. ISSN 1553-734X. doi: 10.1371/journal.pcbi.1004506. URL `http://dx.doi.org/10.1371/journal.pcbi.1004506`.

[39] Vladimir Gligorijević, Noël Malod-Dognin, and Nataša Pržulj. Fuse: multiple network alignment via data fusion. 32(8):1195–1203. doi: 10.1093/bioinformatics/btv731. URL `http://dx.doi.org/10.1093/bioinformatics/btv731`.

[40] Vikram Khurana, Jian Peng, Chee Chung, Pavan K Auluck, Saranna Fanning, et al. Genome-scale networks link neurodegenerative disease genes to α-synuclein through specific molecular pathways. 4(2):157–170.e14. ISSN 2405-4712. doi: 10.1016/j.cels.2016.12.011. URL `http://dx.doi.org/10.1016/j.cels.2016.12.011`.

[41] Lenore Cowen, Trey Ideker, Benjamin J Raphael, and Roded Sharan. Network propagation: a universal amplifier of genetic associations. *Nature Reviews Genetics*, 2017. ISSN 1471-0056. doi: 10.1038/nrg.2017.38. URL `http://dx.doi.org/10.1038/nrg.2017.38`.

[42] Risi Imre Kondor and John D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the 19th International Conference on Machine Learning*, ICML '02, page 315–322, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1558608737.

[43] François Fouss, Marco Saerens, and Masashi Shimbo. *Algorithms and models for network data and link analysis*. Cambridge University Press, 2016.

[44] Kevin R Brown and Igor Jurisica. Unequal evolutionary conservation of human protein interactions in interologous networks. 8(5):R95.

[45] Stanley Letovsky and Simon Kasif. Predicting protein function from protein/protein interaction data: a probabilistic approach. *Bioinformatics*, 19(suppl 1):i197–i204, 2003. doi: 10.1093/bioinformatics/btg1026. URL `http://bioinformatics.oxfordjournals.org/content/19/suppl_1/i197.abstract`.

[46] Fabio Vandin, Eli Upfal, and Benjamin J. Raphael. Algorithms for detecting significantly mutated pathways in cancer. *Journal of Computational Biology*, 18(3):507–522, 11 2011. doi: 10.1089/cmb.2010.0265. URL `http://dx.doi.org/10.1089/cmb.2010.0265`.

[47] Charles Boone, Howard Bussey, and Brenda J Andrews. Exploring genetic interactions and networks with yeast. 8(6):437–449. ISSN 1471-0056. doi: 10.1038/nrg2085. URL `http://dx.doi.org/10.1038/nrg2085`.

[48] Florian L. Muller, Elisa A. Aquilanti, and Ronald A. DePinho. Collateral lethality: a new therapeutic strategy in oncology. 1(3):161–173. ISSN 2405-8033. doi: 10.1016/j.trecan.2015.10.002. URL `http://dx.doi.org/10.1016/j.trecan.2015.10.002`.

[49] Andrea Franceschini, Damian Szklarczyk, Sune Frankild, Michael Kuhn, Milan Simonovic, Alexander Roth, Jianyi Lin, Pablo Minguez, Peer Bork, Christian von Mering, and Lars J. Jensen. STRING v9.1: protein-protein interaction networks, with increased coverage and integration. 41(D1):D808–D815. doi: 10.1093/nar/gks1094. URL `http://nar.oxfordjournals.org/content/41/D1/D808.abstract`.

[50] C. Stark, B.J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: a general repository for interaction datasets. 34(Database Issue):D535–D539.

[51] Sean R Collins, Kyle M Miller, Nancy L Maas, Assen Roguev, Jeffrey Fillingham, et al. Functional dissection of protein complexes involved in yeast chromosome biology using a genetic interaction map. *Nature*, 446(7137):806–810, 7 2007. ISSN 0028-0836. doi: 10.1038/nature05649. URL `http://dx.doi.org/10.1038/nature05649`.

[52] Philip Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. 11:95–130.

[53] Catia Pesquita, Daniel Faria, André O. Falcão, Phillip Lord, and Francisco M. Couto. Semantic similarity in biomedical ontologies. 5(7):1–12. doi: 10.1371/journal.pcbi.1000443.

[54] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. 22(10):1345–1359. ISSN 1041-4347. doi: 10.1109/TKDE.2009.191. URL `http://dx.doi.org/10.1109/TKDE.2009.191`.

[55] Michael Costanzo, Benjamin VanderSluis, Elizabeth N Koch, Anastasia Baryshnikova, Carles Pons, et al. A global genetic interaction network maps a wiring diagram of cellular function. *Science*, 353(6306):aaf1420, 2016. ISSN 0036-8075. doi: 10.1126/science.aaf1420. URL `http://dx.doi.org/10.1126/science.aaf1420`.

[56] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2058–2065. AAAI Press, . URL `http://dl.acm.org/citation.cfm?id=3016100.3016186`.

[57] G. R. Lanckriet, M. Deng, N. Cristianini, M. I. Jordan, and W. S. Noble. Kernel-based data fusion and its application to protein function prediction in yeast. In *Pacific Symposium on Biocomputing*, pages 300–311.

[58] Mengfei Cao, Hao Zhang, Jisoo Park, Noah Daniels, Mark Crovella, et al. Going the distance for protein function prediction: A new distance metric for protein interaction networks. 8(10):e76339. doi: 10.1371/journal.pone.0076339. URL `http://www.cs.bu.edu/faculty/crovella/paper-archive/dsd-plos-one.pdf`.

[59] Christina S. Leslie, Eleazar Eskin, Adiel Cohen, Jason Weston, and William Stafford Noble. Mismatch string kernels for discriminative protein classification. 20(4):467. doi: 10.1093/bioinformatics/btg431. URL `http://dx.doi.org/10.1093/bioinformatics/btg431`.

[60] Elena Kuzmin, Benjamin VanderSluis, Wen Wang, Guihong Tan, Raamesh Deshpande, et al. Systematic analysis of complex genetic interactions. *Science*, 360(6386), 2018. ISSN 0036-8075. doi: 10.1126/science.aao1729. URL `http://dx.doi.org/10.1126/science.aao1729`.

[61] Clyde A. Hutchison, Ray-Yuan Chuang, Vladimir N. Noskov, Nacyra Assad-Garcia, Thomas J. Deerinck, Mark H. Ellisman, John Gill, Krishna Kannan, Bogumil J. Karas, Li Ma, James F. Pelletier, Zhi-Qing Qi, R. Alexander Richter, Elizabeth A. Strychalski, Lijie Sun, Yo Suzuki, Billyana Tsvetanova, Kim S. Wise, Hamilton O. Smith, John I. Glass, Chuck Merryman, Daniel G. Gibson, and J. Craig Venter. Design and synthesis of a minimal bacterial genome. *Science*, 351(6280), 2016. ISSN 0036-8075. doi: 10.1126/science.aad6253. URL `https://science.sciencemag.org/content/351/6280/aad6253`.

[62] Alan Ashworth and Christopher J. Lord. Synthetic lethal therapies for cancer: what's next after PARP inhibitors? *Nature Reviews Clinical Oncology*, 15(9):564–576, 2018. ISSN 1759-4782. doi: 10.1038/s41571-018-0055-6. URL `https://doi.org/10.1038/s41571-018-0055-6`.

[63] Joo-Sang Lee, Avinash Das, Livnat Jerby-Arnon, Rand Arafeh, Noam Auslander, et al. Harnessing synthetic lethality to predict the response to cancer treatment. *Nature Communications*, 9(1):2546, 2018. doi: 10.1038/s41467-018-04647-1.

[64] Kristin L. Patrick, Jason A. Wojcechowskyj, Samantha L. Bell, Morgan N. Riba, Tao Jing, Sara Talmage, Pengbiao Xu, Ana L. Cabello, Jiewei Xu, Michael Shales, David Jimenez-Morales, Thomas A. Ficht, Paul de Figueiredo, James E. Samuel, Pingwei Li, Nevan J. Krogan, and Robert O. Watson. Quantitative yeast genetic interaction profiling of bacterial effector proteins uncovers a role for the human retromer in salmonella infection. *Cell Systems*, 7(3):323 – 338.e6, 2018. ISSN 2405-4712. doi: https://doi.org/10.1016/j.cels.2018.06.010. URL `http://www.sciencedirect.com/science/article/pii/S2405471218302771`.

[65] Michael Costanzo, Elena Kuzmin, Jolanda van Leeuwen, Barbara Mair, Jason Moffat, Charles Boone, and Brenda Andrews. Global genetic networks and the genotype-to-phenotype relationship. *Cell*, 177(1):85 – 100, 2019. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2019.01.033. URL `http://www.sciencedirect.com/science/article/pii/S0092867419300960`.

[66] Maya Schuldiner, Sean R. Collins, Natalie J. Thompson, Vladimir Denic, Arunashree Bhamidipati, et al. Exploration of the function and organization of the yeast early secretory

pathway through an epistatic miniarray profile. *Cell*, 123(3):507–519, 5 2005. ISSN 0092-8674. doi: 10.1016/j.cell.2005.08.031. URL `http://dx.doi.org/10.1016/j.cell.2005.08.031`.

[67] Colm J. Ryan, Assen Roguev, Kristin Patrick, Jiewei Xu, Harlizawati Jahari, et al. Hierarchical modularity and the evolution of genetic interactomes across species. *Molecular Cell*, 46(5):691–704, 12 2012. ISSN 1097-2765. doi: 10.1016/j.molcel.2012.05.028. URL `http://dx.doi.org/10.1016/j.molcel.2012.05.028`.

[68] Atray Dixit, Oren Parnas, Biyu Li, Jenny Chen, Charles P. Fulco, Livnat Jerby-Arnon, Nemanja D. Marjanovic, Danielle Dionne, Tyler Burks, Raktima Raychowdhury, Britt Adamson, Thomas M. Norman, Eric S. Lander, Jonathan S. Weissman, Nir Friedman, and Aviv Regev. Perturb-seq: Dissecting molecular circuits with scalable single-cell rna profiling of pooled genetic screens. *Cell*, 167(7):1853 – 1866.e17, 2016. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2016.11.038. URL `http://www.sciencedirect.com/science/article/pii/S0092867416316105`.

[69] Sharyl L. Wong, Lan V. Zhang, Amy H. Y. Tong, Zhijian Li, Debra S. Goldberg, Oliver D. King, Guillaume Lesage, Marc Vidal, Brenda Andrews, Howard Bussey, Charles Boone, and Frederick P. Roth. Combining biological networks to predict genetic interactions. *Proceedings of the National Academy of Sciences*, 101(44):15682–15687, 2004. ISSN 0027-8424. doi: 10.1073/pnas.0406614101. URL `https://www.pnas.org/content/101/44/15682`.

[70] Sri R. Paladugu, Shan Zhao, Animesh Ray, and Alpan Raval. Mining protein networks for synthetic genetic interactions. *BMC Bioinformatics*, 9(1):426, 2008. ISSN 1471-2105. doi: 10.1186/1471-2105-9-426. URL `https://doi.org/10.1186/1471-2105-9-426`.

[71] Gaurav Pandey, Bin Zhang, Aaron N. Chang, Chad L. Myers, Jun Zhu, Vipin Kumar, and Eric E. Schadt. An integrative multi-network and multi-classifier approach to predict genetic interactions. *PLOS Computational Biology*, 6(9):1–14, 09 2010. doi: 10.1371/journal.pcbi.1000928. URL `https://doi.org/10.1371/journal.pcbi.1000928`.

[72] Min Wu, Xuejuan Li, Fan Zhang, Xiaoli Li, Chee-Keong Kwoh, and Jie Zheng. In silico prediction of synthetic lethality by meta-analysis of genetic interactions, functions, and pathways in yeast and human cancer. *Cancer informatics*, 13(Suppl 3):71–80, Nov 2014. ISSN 1176-9351. doi: 10.4137/CIN.S14026. URL `https://pubmed.ncbi.nlm.nih.gov/25452682`. cin-suppl.3-2014-071[PII].

[73] Graeme Benstead-Hume, Xiangrong Chen, Suzanna R. Hopkins, Karen A. Lane, Jessica A. Downs, and Frances M. G. Pearl. Predicting synthetic lethal interactions using conserved patterns in protein interaction networks. *PLOS Computational Biology*, 15(4):1–25, 04 2019. doi: 10.1371/journal.pcbi.1006888. URL `https://doi.org/10.1371/journal.pcbi.1006888`.

[74] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the 12th International Conference on Information and Knowledge Management*, CIKM '03, pages 556–559, New York, NY, USA, 2003. Association for Com-

puting Machinery. ISBN 1581137230. doi: 10.1145/956863.956972. URL `https://doi.org/10.1145/956863.956972`.

[75] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150 – 1170, 2011. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2010.11.027. URL `http://www.sciencedirect.com/science/article/pii/S037843711000991X`.

[76] Michael Costanzo, Anastasia Baryshnikova, Jeremy Bellay, Yungil Kim, Eric D Spear, et al. The genetic landscape of a cell. *Science*, 327(5964):425–431, 10 2010. ISSN 0036-8075. doi: 10.1126/science.1180823. URL `http://dx.doi.org/10.1126/science.1180823`.

[77] Igor Ulitsky, Nevan J Krogan, and Ron Shamir. Towards accurate imputation of quantitative genetic interactions. *Genome Biology*, 10(12):1–18, 9 2009. ISSN 1474-760X. doi: 10.1186/gb-2009-10-12-r140. URL `http://dx.doi.org/10.1186/gb-2009-10-12-r140`.

[78] The Gene Ontology Consortium. The Gene Ontology Resource: 20 years and still GOing strong. *Nucleic Acids Research*, 47(D1):D330–D338, 11 2018. ISSN 0305-1048. doi: 10.1093/nar/gky1055. URL `https://doi.org/10.1093/nar/gky1055`.

[79] Jianzhu Ma, Michael Ku Yu, Samson Fong, Keiichiro Ono, Eric Sage, et al. Using deep learning to model the hierarchical structure and function of a cell. *Nature Methods*, 15(4), 2018. ISSN 1548-7105. doi: 10.1038/nmeth.4627. URL `http://dx.doi.org/10.1038/nmeth.4627`.

[80] Michael Yu, Michael Kramer, Janusz Dutkowski, Rohith Srivas, Katherine Licon, Jason F Kreisberg, Cherie T Ng, Nevan Krogan, Roded Sharan, and Trey Ideker. Translation of genotype to phenotype by a hierarchy of cell subsystems. *Cell Systems*, 2(2):77–88, 2016. ISSN 2405-4712. doi: 10.1016/j.cels.2016.02.003. URL `http://dx.doi.org/10.1016/j.cels.2016.02.003`.

[81] Elizabeth N Koch, Michael Costanzo, Jeremy Bellay, Raamesh Deshpande, Kate Chatfield-Reed, Gordon Chua, Gennaro D'Urso, Brenda J Andrews, Charles Boone, and Chad L Myers. Conserved rules govern genetic interaction degree across species. *Genome Biology*, 13 (7):R57, 2012. ISSN 1465-6906. doi: 10.1186/gb-2012-13-7-r57.

[82] Steffen Rendle, Li Zhang, and Yehuda Koren. On the difficulty of evaluating baselines: A study on recommender systems. *CoRR*, abs/1905.01395, 2019. URL `http://arxiv.org/abs/1905.01395`.

[83] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

[84] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, page 1257–1264, Red Hook, NY, USA, 2007. Curran Associates Inc. ISBN 9781605603520.

[85] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug 2009. ISSN 1558-0814. doi: 10.1109/MC.2009.263.

[86] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717, 4 2009. ISSN 1615-3383. doi: 10.1007/s10208-009-9045-5. URL `https://doi.org/10.1007/s10208-009-9045-5`.

[87] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999. ISSN 1476-4687. doi: 10.1038/44565. URL `https://doi.org/10.1038/44565`.

[88] Genevieve L. Stein-O'Brien, Raman Arora, Aedin C. Culhane, Alexander V. Favorov, Lana X. Garmire, Casey S. Greene, Loyal A. Goff, Yifeng Li, Aloune Ngom, Michael F. Ochs, Yanxun Xu, and Elana J. Fertig. Enter the matrix: factorization uncovers knowledge from omics. *Trends in Genetics*, 34(10):790–805, Oct 2018. ISSN 0168-9525. doi: 10.1016/j.tig.2018.07.003. URL `https://doi.org/10.1016/j.tig.2018.07.003`.

[89] Tinghui Zhou, Hanhuai Shan, Arindam Banerjee, and Guillermo Sapiro. *Kernelized probabilistic matrix factorization: exploiting graphs and side information*, pages 403–414. 2012. doi: 10.1137/1.9781611972825.35. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611972825.35`.

[90] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the 2002 Pacific Symposium on Biocomputing*, 2002.

[91] Marinka Zitnik and Blaž Zupan. Data imputation in epistatic MAPs by network-guided matrix completion. *Journal of Computational Biology*, 22(6):595–608, 2015. doi: 10.1089/cmb.2014.0158. URL `https://doi.org/10.1089/cmb.2014.0158`. PMID: 25658751.

[92] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[93] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, November 2016. USENIX Association. ISBN 978-1-931971-33-1. URL `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`.

[94] Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B. Hall, Christopher H. Tomkins-Tinch, Vanessa Sochat, et al. Sustainable data analysis with Snakemake. *F1000Research*, 10:33, 2021. doi: 10.12688/f1000research.29032.1.

[95] Anastasia Baryshnikova, Michael Costanzo, Scott Dixon, Franco J. Vizeacoumar, Chad L. Myers, Brenda Andrews, and Charles Boone. Chapter 7 - Synthetic genetic array (SGA) Analysis in Saccharomyces cerevisiae and Schizosaccharomyces pombe. In *Guide to Yeast*

Genetics: Functional Genomics, Proteomics, and Other Systems Analysis, volume 470 of *Methods in Enzymology*, pages 145 – 179. Academic Press, 2010. doi: https://doi.org/10.1016/S0076-6879(10)70007-0. URL `http://www.sciencedirect.com/science/article/pii/S0076687910700070`.

[96] J. Michael Cherry, Eurie L. Hong, Craig Amundsen, Rama Balakrishnan, Gail Binkley, Esther T. Chan, Karen R. Christie, Maria C. Costanzo, Selina S. Dwight, Stacia R. Engel, Dianna G. Fisk, Jodi E. Hirschman, Benjamin C. Hitz, Kalpana Karra, Cynthia J. Krieger, Stuart R. Miyasato, Rob S. Nash, Julie Park, Marek S. Skrzypek, Matt Simison, Shuai Weng, and Edith D. Wong. Saccharomyces Genome Database: the genomics resource of budding yeast. *Nucleic Acids Research*, 40(D1):D700–D705, 11 2011. ISSN 0305-1048. doi: 10.1093/nar/gkr1029. URL `https://doi.org/10.1093/nar/gkr1029`.

[97] Antonia Lock, Kim Rutherford, Midori A Harris, Jacqueline Hayles, Stephen G Oliver, Jürg Bähler, and Valerie Wood. PomBase 2018: user-driven reimplementation of the fission yeast database provides rapid and intuitive access to diverse, interconnected information. *Nucleic Acids Research*, 47(D1):D821–D827, 10 2018. ISSN 0305-1048. doi: 10.1093/nar/gky961. URL `https://doi.org/10.1093/nar/gky961`.

[98] Rose Oughtred, Chris Stark, Bobby-Joe Breitkreutz, Jennifer Rust, Lorrie Boucher, Christie Chang, Nadine Kolas, Lara O'Donnell, Genie Leung, Rochelle McAdam, Frederick Zhang, Sonam Dolma, Andrew Willems, Jasmin Coulombe-Huntington, Andrew Chatr-aryamontri, Kara Dolinski, and Mike Tyers. The BioGRID interaction database: 2019 update. *Nucleic Acids Research*, 47(D1):D529–D541, 11 2018. ISSN 0305-1048. doi: 10.1093/nar/gky1079. URL `https://doi.org/10.1093/nar/gky1079`.

[99] Mehmet Gönen and Ethem Alpaydin. Multiple kernel learning algorithms. *J. Mach. Learn. Res.*, 12:2211–2268, July 2011. ISSN 1532-4435.

[100] Levi A. Garraway and Eric S. Lander. Lessons from the cancer genome. *Cell*, 153(1):17–37, 2013. ISSN 0092-8674. doi: 10.1016/j.cell.2013.03.002.

[101] Moritz Gerstung, Clemency Jolly, Ignaty Leshchiner, Stefan C. Dentro, Santiago Gonzalez, et al. The evolutionary history of 2,658 cancers. *Nature*, 578(7793):122–128, 2020. ISSN 0028-0836. doi: 10.1038/s41586-019-1907-7.

[102] Cyriac Kandoth, Michael D McLellan, Fabio Vandin, Kai Ye, Beifang Niu, Charles Lu, Mingchao Xie, Qunyuan Zhang, Joshua F McMichael, Matthew A Wyczalkowski, Mark DM Leiserson, Christopher A Miller, John S Welch, Matthew J Walter, Michael C Wendl, Timothy J Ley, Richard K Wilson, Benjamin J Raphael, and Li Ding. Mutational landscape and significance across 12 major cancer types. *Nature*, 502(7471):333–339, 2013. ISSN 0028-0836. doi: 10.1038/nature12634.

[103] Michael R Stratton, Peter J Campbell, and Andrew P Futreal. The cancer genome. *Nature*, 458(7239):719, 2009. ISSN 1476-4687. doi: 10.1038/nature07943.

[104] Douglas Hanahan and Robert A Weinberg. Hallmarks of cancer: The next generation. *Cell*, 144(5):646–674, 2011. ISSN 0092-8674.

[105] Michael S Lawrence, Petar Stojanov, Craig H Mermel, James T Robinson, Levi A Garraway, et al. Discovery and saturation analysis of cancer genes across 21 tumour types. *Nature*, 505 (7484):495–501, 2014. ISSN 0028-0836.

[106] Jon Zugazagoitia, Cristiano Guedes, Santiago Ponce, Irene Ferrer, Sonia Molina-Pinelo, and Luis Paz-Ares. Current challenges in cancer treatment. *Clinical Therapeutics*, 38(7):1551–1566, 2016. ISSN 0149-2918.

[107] Thomas Helleday, Saeed Eshtad, and Serena Nik-Zainal. Mechanisms underlying mutational signatures in human cancers. *Nature Reviews Genetics*, 15(9):585–598, 2014. ISSN 1471-0064. doi: 10.1038/nrg3729.

[108] Serena Nik-Zainal, Jill E Kucab, Sandro Morganella, Dominik Glodzik, Ludmil B Alexandrov, Volker M Arlt, Annette Weninger, Monica Hollstein, Michael R Stratton, and David H Phillips. The genome as a record of environmental exposure. *Mutagenesis*, 30(6):763–770, 2015. ISSN 0267-8357. doi: 10.1093/mutage/gev073.

[109] Anthony Tubbs and André Nussenzweig. Endogenous DNA Damage as a source of genomic instability in cancer. *Cell*, 168(4):644–656, 2017. ISSN 0092-8674. doi: 10.1016/j.cell.2017.01.002.

[110] Arne Hoeck, Niels H Tjoonk, Ruben van Boxtel, and Edwin Cuppen. Portrait of a cancer: mutational signature analyses for cancer diagnostics. *BMC Cancer*, 19(1):457, 2019. doi: 10.1186/s12885-019-5677-2.

[111] Dung T Le, Jennifer N Durham, Kellie N Smith, Hao Wang, Bjarne R Bartlett, Laveet K Aulakh, et al. Mismatch repair deficiency predicts response of solid tumors to PD-1 blockade. *Science*, 357(6349):409–413, 2017. ISSN 0036-8075.

[112] Hannah Farmer, Nuala McCabe, Christopher J Lord, Andrew NJ Tutt, Damian A Johnson, Tobias B Richardson, et al. Targeting the DNA repair defect in BRCA mutant cells as a therapeutic strategy. *Nature*, 434(7035):917–921, 2005. ISSN 0028-0836.

[113] Christopher J Lord and Alan Ashworth. PARP inhibitors: Synthetic lethality in the clinic. *Science*, 355:1152–1158, 2017.

[114] Serena Nik-Zainal, Ludmil B Alexandrov, David C Wedge, Peter Van Loo, Christopher D Greenman, Keiran Raine, et al. Mutational processes molding the genomes of 21 breast cancers. *Cell*, 149(5):979–993, 2012. ISSN 0092-8674.

[115] Ludmil B. Alexandrov, Serena Nik-Zainal, David C. Wedge, Samuel A. J. R. Aparicio, Sam Behjati, Andrew V. Biankin, et al. Signatures of mutational processes in human cancer. *Nature*, 500(7463):415–421, 2013. ISSN 1476-4687.

[116] Ludmil B Alexandrov, Serena Nik-Zainal, David C Wedge, Peter J Campbell, and Michael R Stratton. Deciphering signatures of mutational processes operative in human cancer. *Cell Reports*, 3(1):246–259, 2013. ISSN 2211-1247.

[117] Jaegil Kim, Kent W Mouw, Paz Polak, Lior Z Braunstein, Atanas Kamburov, Grace Tiao, et al. Somatic ERCC2 mutations are associated with a distinct genomic signature in urothelial tumors. *Nature Genetics*, 48(6):600–606, 2016. ISSN 1061-4036.

[118] Ludmil B Alexandrov, Jaegil Kim, Nicholas J Haradhvala, Mi Huang, Alvin Ng, Yang Wu, et al. The repertoire of mutational signatures in human cancer. *Nature*, 578(7793):94–101, 2020. ISSN 0028-0836.

[119] Arne Van Hoeck, Niels H Tjoonk, Ruben van Boxtel, and Edwin Cuppen. Portrait of a cancer: mutational signature analyses for cancer diagnostics. *BMC Cancer*, 19(1):457, 2019.

[120] Helen Davies, Dominik Glodzik, Sandro Morganella, Lucy R Yates, Johan Staaf, Xueqing Zou, et al. HRDetect is a predictor of BRCA1 and BRCA2 deficiency based on mutational signatures. *Nature Medicine*, 23(4):517–525, 2017.

[121] Donovan T Cheng, Talia N Mitchell, Ahmet Zehir, Ronak H. Shah, Ryma Benayed, Aijazuddin Syed, et al. Memorial Sloan Kettering-Integrated Mutation Profiling of Actionable Cancer Targets (MSK-IMPACT): A Hybridization Capture-Based Next-Generation Sequencing Clinical Assay for Solid Tumor Molecular Oncology. *Journal of Molecular Diagnostics*, 17 (3):251–264, 2015.

[122] Garrett M Frampton, Alex Fichtenholtz, Geoff A Otto, Kai Wang, Sean R Downing, Jie He, et al. Development and validation of a clinical cancer genomic profiling test based on massively parallel DNA sequencing. *Nature Biotechnology*, 31(11):1023–1031, 2013. ISSN 1087-0156.

[123] Serena Nik-Zainal, Yasin Memari, and Helen R. Davies. Holistic cancer genome profiling for every patient. *Swiss Medical Weekly*, 150(0304):w20158, 2020.

[124] Brittany B Campbell, Nicholas Light, David Fabrizio, Matthew Zatzman, Fabio Fuligni, Scott Davidson, et al. Comprehensive Analysis of Hypermutation in Human Cancer. *Cell*, 171(5):1042–1056, 2017. ISSN 0092-8674.

[125] Rachel Rosenthal, Nicholas McGranahan, Javier Herrero, Barry S Taylor, and Charles Swanton. deconstructSigs: delineating mutational processes in single tumors distinguishes DNA repair deficiencies and patterns of carcinoma evolution. *Genome Biology*, 17(1):31, 2016. ISSN 1474-760X.

[126] Doga C Gulhan, Jake June-Koo Lee, Giorgio E M Melloni, Isidro Cortés-Ciriano, and Peter J Park. Detecting the mutational signature of homologous recombination deficiency in clinical samples. *Nature Genetics*, 51(5):912–919, 2019.

[127] Itay Sason, Yuexi Chen, Mark D.M. Leiserson, and Roded Sharan. A mixture model for signature discovery from sparse mutation data. *Genome Medicine*, 13(1):173, Nov 2021. ISSN 1756-994X. doi: 10.1186/s13073-021-00988-7. URL `https://doi.org/10.1186/s13073-021-00988-7`.

[128] Daniel Temko, Ian PM Tomlinson, Simone Severini, Benjamin Schuster-Böckler, and Trevor A Graham. The effects of mutational processes and selection on driver mutations across cancer types. *Nature Communications*, 9(1):1857, 2018. doi: 10.1038/s41467-018-04208-6.

[129] Paz Polak, Jaegil Kim, Lior Z. Braunstein, Rosa Karlic, Nicholas J. Haradhavala, Grace Tiao, et al. A mutational signature reveals alterations underlying deficient homologous recombination repair in breast cancer. *Nature Genetics*, 49(10):1476–1486, 2017. ISSN 1546-1718.

[130] Juliane Perner, Sujath Abbas, Karol Nowicki-Osuch, Ginny Devonshire, Matthew D Eldridge, Simon Tavaré, et al. The mutREAD method detects mutational signatures from low quantities of cancer DNA. *Nature communications*, 11(1):3166, 2020.

[131] John G. Tate, Sally Bamford, Harry C. Jubb, Zbyslaw Sondka, David M. Beare, Nidhi Bindal, et al. COSMIC: the Catalogue Of Somatic Mutations In Cancer. *Nucleic Acids Research*, 47 (D1):D941–D947, 2019.

[132] Wei Jiao, Gurnit Atwal, Paz Polak, Rosa Karlic, Edwin Cuppen, Fatima Al-Shahrour, et al. A deep learning system accurately classifies primary and metastatic cancers using passenger mutation patterns. *Nature Communications*, 11(1):728, 2020. ISSN 2041-1723.

[133] Paz Polak, Rosa Karlić, Amnon Koren, Robert Thurman, Richard Sandstrom, Michael S Lawrence, et al. Cell-of-origin chromatin organization shapes the mutational landscape of cancer. *Nature*, 518(7539):360–364, 2015.

[134] Nicholas J Haradhvala, Paz Polak, Petar Stojanov, Kyle R Covington, Eve Shinbrot, Julian M Hess, et al. Mutational strand asymmetries in cancer genomes reveal mechanisms of DNA damage and repair. *Cell*, 164(3):538–549, 2016.

[135] Sandro Morganella, Ludmil B Alexandrov, Dominik Glodzik, Xueqing Zou, Helen Davies, Johan Staaf, et al. The topography of mutational processes in breast cancer genomes. *Nature Communications*, 7:11383, 2016.

[136] Christos Boutsidis and Efstratios Gallopoulos. Svd based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition*, 41(4):1350–1362, 2008. ISSN 0031-3203.

[137] Serena Nik-Zainal, Helen Davies, Johan Staaf, Manasa Ramakrishna, Dominik Glodzik, Xueqing Zou, et al. Landscape of somatic mutations in 560 breast cancer whole-genome sequences. *Nature*, 534(7605), 2016. ISSN 1476-4687.

[138] Johan Staaf, Dominik Glodzik, Ana Bosch, Johan Vallon-Christersson, Christel Reuterswärd, Jari Häkkinen, et al. Whole-genome sequencing of triple-negative breast cancers in a population-based clinical study. *Nature Medicine*, 25(10):1526–1533, 2019. ISSN 1078-8956.

[139] Xiaoqing Huang, Damien Wojtowicz, and Teresa M Przytycka. Detecting presence of mutational signatures in cancer with confidence. *Bioinformatics*, 34(2):330–337, 2018.

[140] Ludmil B Alexandrov, Philip H Jones, David C Wedge, Julian E Sale, Peter J Campbell, Serena Nik-Zainal, et al. Clock-like mutational processes in human somatic cells. *Nature Genetics*, 47(12):1402–1407, 2015. ISSN 1061-4036.

[141] Alison J. Coffey, Felix Kokocinski, Maria S. Calafato, Carol E. Scott, Priit Palta, Eleanor Drury, et al. The GENCODE exome: sequencing the complete human exome. *European Journal of Human Genetics*, 19(7):827–831, 2011.

[142] Elena Bushmanova, Dmitry Antipov, Alla Lapidus, and Andrey D Prjibelski. rnaSPAdes: a de novo transcriptome assembler and its application to RNA-Seq data. *GigaScience*, 8(9), 09 2019. ISSN 2047-217X. doi: 10.1093/gigascience/giz100. URL `https://doi.org/10.1093/gigascience/giz100`. giz100.

[143] Manfred G Grabherr, Brian J Haas, Moran Yassour, Joshua Z Levin, Dawn A Thompson, Ido Amit, et al. Full-length transcriptome assembly from RNA-Seq data without a reference genome. *Nature Biotechnology*, 29(7):644–652, 2011. ISSN 1087-0156. doi: 10.1038/nbt.1883.

[144] Migun Shakya, Chien-Chi Lo, and Patrick S. G. Chain. Advances and challenges in metatranscriptomic analysis. *Frontiers in Genetics*, 10:904, 2019. ISSN 1664-8021. doi: 10.3389/fgene.2019.00904. URL `https://www.frontiersin.org/article/10.3389/fgene.2019.00904`.

[145] Simon Anders, Paul Theodor Pyl, and Wolfgang Huber. Htseq—a python framework to work with high-throughput sequencing data. *Bioinformatics*, 31(2):166–169, 2015.

[146] Yang Liao, Gordon K Smyth, and Wei Shi. featurecounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, 30(7):923–930, 2014.

[147] Charlotte Soneson, Michael I. Love, and Mark D. Robinson. Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. *F1000Research*, 4, 2015. doi: 10.12688/f1000research.7563.1.

[148] Hui Jiang and Wing Hung Wong. Statistical inferences for isoform expression in RNA-seq. *Bioinformatics*, 25(8):1026–1032, 2009.

[149] Ernest Turro, Shu-Yi Su, Ângela Gonçalves, Lachlan JM Coin, Sylvia Richardson, and Alex Lewin. Haplotype and isoform specific expression estimation using multi-mapping RNA-seq reads. *Genome biology*, 12(2):1–15, 2011.

[150] Bo Li and Colin N Dewey. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):323, 8 2011. doi: 10.1186/1471-2105-12-323.

[151] Peter Glaus, Antti Honkela, and Magnus Rattray. Identifying differentially expressed transcripts from RNA-seq data with biological variation. *Bioinformatics*, 28(13):1721–1728, 2012.

[152] James Hensman, Panagiotis Papastamoulis, Peter Glaus, Antti Honkela, and Magnus Rattray. Fast and accurate approximate inference of transcript expression from RNA-seq data. *Bioinformatics*, 31(24):3881–3889, 08 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv483. URL `https://doi.org/10.1093/bioinformatics/btv483`.

[153] Naoki Nariai, Osamu Hirose, Kaname Kojima, and Masao Nagasaki. TIGAR: transcript isoform abundance estimation method with gapped alignment of RNA-Seq data by variational Bayesian inference. *Bioinformatics*, 29(18):2292–2299, 07 2013. ISSN 1367-4803. doi: 10.1093/bioinformatics/btt381. URL `https://doi.org/10.1093/bioinformatics/btt381`.

[154] Naoki Nariai, Kaname Kojima, Takahiro Mimori, Yosuke Kawai, and Masao Nagasaki. A bayesian approach for estimating allele-specific expression from RNA-seq data with diploid genomes. In *BMC genomics*, volume 17, pages 7–17. BioMed Central, 2016.

[155] Naoki Nariai, Kaname Kojima, Takahiro Mimori, Yukuto Sato, Yosuke Kawai, Yumi Yamaguchi-Kabata, and Masao Nagasaki. Tigar2: sensitive and accurate estimation of transcript isoform expression with longer RNA-seq reads. *BMC genomics*, 15(10):1–9, 2014.

[156] Daniel C. Jones, Kavitha T. Kuppusamy, Nathan J. Palpant, Xinxia Peng, Charles E. Murry, Hannele Ruohola-Baker, and Walter L. Ruzzo. Isolator: accurate and stable analysis of isoform-level expression in rna-seq experiments. *bioRxiv*, 2016. doi: 10.1101/088765. URL `https://www.biorxiv.org/content/early/2016/11/20/088765`.

[157] Daniel C Jones and Walter L Ruzzo. Polee: RNA-Seq analysis using approximate likelihood. *NAR Genomics and Bioinformatics*, 3(2):lqab046, 2021. ISSN 2631-9268. doi: 10.1093/nargab/lqab046.

[158] Avi Srivastava, Laraib Malik, Hirak Sarkar, and Rob Patro. A Bayesian framework for inter-cellular information sharing improves dscRNA-seq quantification. *Bioinformatics*, 36 (Supplement_1):i292–i299, 2020.

[159] Peng Liu, Rajendran Sanalkumar, Emery H Bresnick, Sündüz Keleş, and Colin N Dewey. Integrative analysis with chip-seq advances the limits of transcript quantification from rna-seq. *Genome research*, 26(8):1124–1133, 2016.

[160] Zhenqiang Su, Paweł P Łabaj, Sheng Li, Jean Thierry-Mieg, Danielle Thierry-Mieg, Wei Shi, et al. A comprehensive assessment of RNA-seq accuracy, reproducibility and information content by the Sequencing Quality Control Consortium. *Nature Biotechnology*, 32(9):903–914, 2014. ISSN 1087-0156. doi: 10.1038/nbt.2957.

[161] Atif Rahman and Lior Pachter. CGAL: computing genome assembly likelihoods. *Genome Biology*, 14(1):R8, 2013. ISSN 1465-6906. doi: 10.1186/gb-2013-14-1-r8.

[162] Bo Li, Nathanael Fillmore, Yongsheng Bai, Mike Collins, James A Thomson, Ron Stewart, and Colin N Dewey. Evaluation of de novo transcriptome assemblies from RNA-Seq data. *Genome Biology*, 15(12):553, 2014. doi: 10.1186/s13059-014-0553-5.

[163] Richard Smith-Unna, Chris Boursnell, Rob Patro, Julian M. Hibberd, and Steven Kelly. TransRate: reference-free quality assessment of de novo transcriptome assemblies. *Genome Research*, 26(8):1134–1144, 2016. ISSN 1088-9051. doi: 10.1101/gr.196469.115.

[164] Scott C. Clark, Rob Egan, Peter I. Frazier, and Zhong Wang. ALE: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome assemblies. *Bioinformatics*, 29(4):435–443, 2013. ISSN 1367-4803. doi: 10.1093/bioinformatics/bts723.

[165] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL `https://www.sciencedirect.com/science/article/pii/0377042787901257`.

[166] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, March 2003. ISSN 1532-4435.

[167] Federick Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976. ISSN 0018-9219. doi: 10.1109/proc.1976.10159.

[168] Mohsen Zakeri, Avi Srivastava, Fatemeh Almodaresi, and Rob Patro. Improved data-driven likelihood factorizations for transcript abundance estimation. *Bioinformatics*, 33(14):i142–i151, 07 2017. ISSN 1367-4803. doi: 10.1093/bioinformatics/btx262. URL `https://doi.org/10.1093/bioinformatics/btx262`.

[169] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer New York, 2016.

[170] William A. Gale. Good-Turing smoothing without tears. *Journal of Quantitative Linguistics*, 2, 1995.

[171] Aaron T. L. Lun, Samantha Riesenfeld, Tallulah Andrews, The Phuong Dao, Tomas Gomes, John C. Marioni, and participants in the 1st Human Cell Atlas Jamboree. EmptyDrops: distinguishing cells from empty droplets in droplet-based single-cell rna sequencing data. *Genome Biology*, 20(1):63, Mar 2019. ISSN 1474-760X. doi: 10.1186/s13059-019-1662-y. URL `https://doi.org/10.1186/s13059-019-1662-y`.

[172] Leming Shi, Laura H Reid, Wendell D Jones, Richard Shippy, Janet A Warrington, et al. The MicroArray Quality Control (MAQC) project shows inter- and intraplatform reproducibility of gene expression measurements. *Nature Biotechnology*, 24(9):1151–1161, 2006. ISSN 1087-0156. doi: 10.1038/nbt1239.

[173] Shawn C Baker, Steven R Bauer, Richard P Beyer, James D Brenton, Bud Bromley, John Burrill, et al. The External RNA Controls Consortium: a progress report. *Nature Methods*, 2(10):731–734, 2005. ISSN 1548-7091. doi: 10.1038/nmeth1005-731.

[174] Woo Jin Kim, Jae Hyun Lim, Jae Seung Lee, Sang-Do Lee, Ju Han Kim, and Yeon-Mok Oh. Comprehensive analysis of transcriptome sequencing data in the lung tissues of COPD

subjects. *International Journal of Genomics*, 2015:206937, 5 2015. ISSN 2314-436X. doi: 10.1155/2015/206937. URL `https://doi.org/10.1155/2015/206937`.

[175] Alyssa C. Frazee, Andrew E. Jaffe, Ben Langmead, and Jeffrey T. Leek. Polyester: simulating RNA-seq datasets with differential transcript expression. *Bioinformatics*, 31(17): 2778–2784, 04 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv272. URL `https://doi.org/10.1093/bioinformatics/btv272`.

[176] Andrew D Yates, Premanand Achuthan, Wasiu Akanni, James Allen, Jamie Allen, Jorge Alvarez-Jarreta, et al. Ensembl 2020. *Nucleic Acids Research*, 48(D1):D682–D688, 11 2019. ISSN 0305-1048. doi: 10.1093/nar/gkz966. URL `https://doi.org/10.1093/nar/gkz966`.

[177] Johannes Rainer. *EnsDb.Hsapiens.v86: Ensembl based annotation package*, 2017. R package version 2.99.0.

[178] Steffen Durinck, Paul T Spellman, Ewan Birney, and Wolfgang Huber. Mapping identifiers for the integration of genomic datasets with the R/Bioconductor package biomaRt. *Nature Protocols*, 4(8):1184–1191, 2009. ISSN 1754-2189. doi: 10.1038/nprot.2009.97.

[179] Anqi Zhu, Avi Srivastava, Joseph G Ibrahim, Rob Patro, and Michael I Love. Nonparametric expression analysis using inferential replicate counts. *Nucleic Acids Research*, 47(18):e105–e105, 2019. ISSN 0305-1048. doi: 10.1093/nar/gkz622.

[180] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature Methods*, 9(4):357–359, Apr 2012. ISSN 1548-7105. doi: 10.1038/nmeth.1923. URL `https://doi.org/10.1038/nmeth.1923`.

[181] Adam Roberts and Lior Pachter. Streaming fragment assignment for real-time analysis of sequencing experiments. *Nature Methods*, 10(1):71–73, Jan 2013. ISSN 1548-7105. doi: 10.1038/nmeth.2251. URL `https://doi.org/10.1038/nmeth.2251`.

[182] Jason Fan, Jamshed Khan, Giulio Ermanno Pibiri, and Rob Patro. Spectrum preserving tilings enable sparse and modular reference indexing. In *Proceedings of the 27th Annual International Conference on Research in Computational Molecular Biology*, pages 21–40, 2023.

[183] Fatemeh Almodaresi, Mohsen Zakeri, and Rob Patro. PuffAligner: a fast, efficient and accurate aligner based on the Pufferfish index. *Bioinformatics*, June 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab408. URL `https://doi.org/10.1093/bioinformatics/btab408`. btab408.

[184] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016. ISSN 1087-0156. doi: 10.1038/nbt.3519.

[185] Rob Patro, Stephen M. Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature Biotechnology*, 32

(5):462–464, May 2014. ISSN 1546-1696. doi: 10.1038/nbt.2862. URL `https://doi.org/10.1038/nbt.2862`.

[186] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in BWT-runs bounded space. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 1459–1477, USA, 2018. Society for Industrial and Applied Mathematics. ISBN 9781611975031.

[187] Massimiliano Rossi, Marco Oliva, Ben Langmead, Travis Gagie, and Christina Boucher. Moni: A pangenomic index for finding maximal exact matches. *Journal of Computational Biology*, 29(2):169–187, 2022. doi: 10.1089/cmb.2021.0290. URL `https://doi.org/10.1089/cmb.2021.0290`. PMID: 35041495.

[188] Omar Ahmed, Massimiliano Rossi, Travis Gagie, Christina Boucher, and Ben Langmead. Spumoni 2: Improved pangenome classification using a compressed index of minimizer digests. *bioRxiv*, 2022. doi: 10.1101/2022.09.08.506805. URL `https://www.biorxiv.org/content/early/2022/09/11/2022.09.08.506805`.

[189] Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. In *Digital SRC Research Report*. Citeseer, 1994.

[190] Fatemeh Almodaresi, Hirak Sarkar, Avi Srivastava, and Rob Patro. A space and time-efficient index for the compacted colored de Bruijn graph. *Bioinformatics*, 34(13):i169–i177, 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty292.

[191] Daehwan Kim, Joseph M. Paggi, Chanhee Park, Christopher Bennett, and Steven L. Salzberg. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nature Biotechnology*, 37(8):907–915, Aug 2019. ISSN 1546-1696. doi: 10.1038/s41587-019-0201-4. URL `https://doi.org/10.1038/s41587-019-0201-4`.

[192] Erik Garrison, Jouni Sirén, Adam M. Novak, Glenn Hickey, Jordan M. Eizenga, Eric T. Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F. Lin, Benedict Paten, and Richard Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, Oct 2018. ISSN 1546-1696. doi: 10.1038/nbt.4227. URL `https://doi.org/10.1038/nbt.4227`.

[193] Ilia Minkin, Son Pham, and Paul Medvedev. TwoPaCo: an efficient algorithm to build the compacted de Bruijn graph from many complete genomes. *Bioinformatics*, 33(24):4024–4032, 09 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw609. URL `https://doi.org/10.1093/bioinformatics/btw609`.

[194] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 06 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw279. URL `https://doi.org/10.1093/bioinformatics/btw279`.

[195] Jamshed Khan and Rob Patro. Cuttlefish: fast, parallel and low-memory compaction of de Bruijn graphs from large-scale genome collections. *Bioinformatics*, 37(Suppl 1):i177–i186, 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab309.

[196] Jamshed Khan, Marek Kokot, Sebastian Deorowicz, and Rob Patro. Scalable, ultra-fast, and low-memory construction of compacted de Bruijn graphs with Cuttlefish 2. *Genome Biology*, 23(1):190, 2022.

[197] Barış Ekim, Bonnie Berger, and Rayan Chikhi. Minimizer-space de bruijn graphs: whole-genome assembly of long reads in minutes on a personal computer. *Cell Systems*, 12(10): 958–968.e6, 2021. ISSN 2405-4712. doi: https://doi.org/10.1016/j.cels.2021.08.009. URL `https://www.sciencedirect.com/science/article/pii/S240547122100332X`.

[198] Mikhail Karasikov, Harun Mustafa, Gunnar Rätsch, and André Kahles. Lossless indexing with counting de bruijn graphs. *Genome Res.*, 32(9):1754–1764, May 2022.

[199] Sebastian Schmidt and Jarno N. Alanko. Eulertigs: minimum plain text representation of k-mer sets without repetitions in linear time. *bioRxiv*, 2022. doi: 10.1101/2022.05.17.492399. URL `https://www.biorxiv.org/content/early/2022/07/20/2022.05.17.492399`.

[200] Giulio Ermanno Pibiri. On weighted k-mer dictionaries. *Algorithms for Molecular Biology*, 18(1):3, Jun 2023. ISSN 1748-7188. doi: 10.1186/s13015-023-00226-2. URL `https://doi.org/10.1186/s13015-023-00226-2`.

[201] Jarno N. Alanko, Simon J. Puglisi, and Jaakko Vuohtoniemi. Small searchable k-spectra via subset rank queries on the spectral burrows-wheeler transform. In *Proceedings of the SIAM Conference on Applied and Computational Discrete Algorithms*, pages 225–236, 2023.

[202] Francisco Claude and Gonzalo Navarro. The wavelet matrix. In *Proceedings of the 19th International Symposium on String Processing and Information Retrieval*, pages 167–179. Springer, 2012.

[203] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In *Proceedings of the 12th International Workshop on Algorithms in Bioinformatics*, pages 225–235. Springer, 2012.

[204] Giulio Ermanno Pibiri and Rossano Venturini. Techniques for inverted index compression. *ACM Computing Surveys*, 53(6):125:1–125:36, 2021.

[205] Uwe Baier, Timo Beller, and Enno Ohlebusch. Graphical pan-genome analysis with compressed suffix trees and the Burrows-Wheeler transform. *Bioinformatics*, 32(4):497–504, October 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv603. URL `https://doi.org/10.1093/bioinformatics/btv603`.

[206] Pranvera Hiseni, Knut Rudi, Robert C Wilson, Finn Terje Hegge, and Lars Snipen. HumGut: a comprehensive human gut prokaryotic genomes collection filtered by metagenome data. *Microbiome*, 9(1):1–12, 2021.

[207] Justin M. Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific Data*, 3(1):160025, June 2016. ISSN 2052-4463. doi: 10.1038/sdata.2016.25. URL `https://doi.org/10.1038/sdata.2016.25`.

[208] Joan Mas-Lloret, Mireia Obón-Santacana, Gemma Ibáñez-Sanz, Elisabet Guinó, Miguel L. Pato, Francisco Rodriguez-Moranta, Alfredo Mata, Ana García-Rodríguez, Victor Moreno, and Ville Nikolai Pimenoff. Gut microbiome diversity detected by high-coverage 16S and shotgun sequencing of paired stool and colon sample. *Scientific Data*, 7(1):92, March 2020. ISSN 2052-4463. doi: 10.1038/s41597-020-0427-5. URL `https://doi.org/10.1038/s41597-020-0427-5`.

[209] Alistair Moffat and Lang Stuiver. Binary interpolative coding for effective index compression. *Information Retrieval*, 3(1):25–47, 2000.

[210] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18): 3094–3100, 2018.

[211] Jason Fan, Noor Pratap Singh, Jamshed Khan, Giulio Ermanno Pibiri, and Rob Patro. Fulgor: A fast and compact k-mer index for large-scale matching and color queries. *bioRxiv*, 2023. doi: 10.1101/2023.05.09.539895. URL `https://www.biorxiv.org/content/early/2023/05/20/2023.05.09.539895`.

[212] Nathan LaPierre, Mohammed Alser, Eleazar Eskin, David Koslicki, and Serghei Mangul. Metalign: efficient alignment-based metagenomic profiling via containment min hash. *Genome Biology*, 21(1):242, Sep 2020. ISSN 1474-760X.

[213] Alexa B. R. McIntyre, Rachid Ounit, Ebrahim Afshinnekoo, Robert J. Prill, Elizabeth Hénaff, Noah Alexander, Samuel S. Minot, David Danko, Jonathan Foox, Sofia Ahsanuddin, Scott Tighe, Nur A. Hasan, Poorani Subramanian, Kelly Moffat, Shawn Levy, Stefano Lonardi, Nick Greenfield, Rita R. Colwell, Gail L. Rosen, and Christopher E. Mason. Comprehensive benchmarking and ensemble approaches for metagenomic classifiers. *Genome Biology*, 18 (1):182, Sep 2017. ISSN 1474-760X.

[214] Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, 15(3):1–12, 2014.

[215] Rachid Ounit, Steve Wanamaker, Timothy J Close, and Stefano Lonardi. CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics*, 16(1):1–13, 2015.

[216] Derrick E. Wood, Jennifer Lu, and Ben Langmead. Improved metagenomic analysis with Kraken 2. *Genome Biology*, 20(1):257, Nov 2019. ISSN 1474-760Xa.

[217] Wei Shen, Hongyan Xiang, Tianquan Huang, Hui Tang, Mingli Peng, Dachuan Cai, Peng Hu, and Hong Ren. KMCP: accurate metagenomic profiling of both prokaryotic and viral populations by pseudo-mapping. *Bioinformatics*, 39(1), 12 2022. ISSN 1367-4811. btac845.

[218] N Tessa Pierce, Luiz Irber, Taylor Reiter, Phillip Brooks, and C Titus Brown. Large-scale sequence comparisons with sourmash. *F1000Research*, 8(1006), 2019. doi: 10.12688/f1000research.19675.1.

[219] Lorian Schaeffer, Harold Pimentel, Nicolas L Bray, Páll Melsted, and Lior Pachter. Pseudoalignment for metagenomic read assignment. *Bioinformatics*, 33(14):2082–2088, 02 2017. ISSN 1367-4803.

[220] Mark Reppell and John Novembre. Using pseudoalignment and base quality to accurately quantify microbial community composition. *PLOS Computational Biology*, 14(4):1–23, 04 2018.

[221] Tommi Mäklin, Teemu Kallonen, Sophia David, Christine J Boinett, Ben Pascoe, Guillaume Méric, David M Aanensen, Edward J Feil, Stephen Baker, Julian Parkhill, et al. High-resolution sweep metagenomics using fast probabilistic inference. *Wellcome open research*, 5(14), 2021.

[222] Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. *Genome biology*, 21(1):1–20, 2020.

[223] Mikhail Karasikov, Harun Mustafa, Amir Joudaki, Sara Javadzadeh-No, Gunnar Rätsch, and André Kahles. Sparse binary relation representations for genome graph annotation. *J Comput Biol*, 27(4):626–639, December 2019.

[224] Jarno N Alanko, Jaakko Vuohtoniemi, Tommi Mäklin, and Simon J Puglisi. Themisto: a scalable colored k-mer index for sensitive pseudoalignment against hundreds of thousands of bacterial genomes. *Bioinformatics*, 39(Supplement_1):i260–i269, 06 2023. ISSN 1367-4811. doi: 10.1093/bioinformatics/btad233. URL `https://doi.org/10.1093/bioinformatics/btad233`.

[225] Dongze He, Mohsen Zakeri, Hirak Sarkar, Charlotte Soneson, Avi Srivastava, and Rob Patro. Alevin-fry unlocks rapid, accurate and memory-frugal quantification of single-cell RNA-seq data. *Nature Methods*, 19(3):316–322, 2022.

[226] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6–es, 2006.

[227] Camille Marchet, Christina Boucher, Simon J Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 31(1):1–12, 2021.

[228] Andrea Cracco and Alexandru I Tomescu. Extremely-fast construction and querying of compacted and colored de Bruijn graphs with GGCAT. In *Proceedings of the 27th Annual International Conference on Research in Computational Molecular Biology*, pages 208–210. Springer, 2023.

[229] Giulio Ermanno Pibiri and Roberto Trani. PTHash: Revisiting FCH minimal perfect hashing. In *Proceedings of the 44th international ACM SIGIR conference on Research & Development in Information Retrieval*, pages 1339–1348, 2021.

[230] Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18): 3363–3369, 2004.

[231] Sebastiano Vigna. Broadword implementation of rank/select queries. In *Proceedings of the 7th International Workshop on Experimental and Efficient Algorithms*, pages 154–168, 2008.

[232] Giulio Ermanno Pibiri and Shunsuke Kanda. Rank/select queries over mutable bitmaps. *Information Systems*, 99(101756), 2021.

[233] Peter Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.

[234] Peter Elias. Efficient storage and retrieval by content and address of static files. *Journal of the ACM*, 21(2):246–260, 1974.

[235] Robert Mario Fano. On the number of bits required to implement an associative memory. *Memorandum 61, Computer Structures Group, MIT*, 1971.

[236] Samy Chambi, Daniel Lemire, Owen Kaser, and Robert Godin. Better bitmap performance with roaring bitmaps. *Software: Practice and Experience*, 46(5):709–719, 2016.

[237] Giuseppe Ottaviano and Rossano Venturini. Partitioned Elias-Fano indexes. In *Proceedings of the 37th International ACM SIGIR conference on Research & Development in Information Retrieval*, pages 273–282, 2014.

[238] Ilya Y Zhbannikov, Samuel S Hunter, Matthew L Settles, and James A Foster. SlopMap: a software application tool for quick and flexible identification of similar sequences using exact k-mer matching. *Journal of data mining in genomics & proteomics*, 4(3), 2013.

[239] Grace A. Blackwell, Martin Hunt, Kerri M. Malone, Leandro Lima, Gal Horesh, Blaise T. F. Alako, Nicholas R. Thomson, and Zamin Iqbal. Exploring bacterial diversity via a curated and searchable snapshot of archived DNA sequences. *PLOS Biology*, 19(11):1–16, 11 2021. URL `https://doi.org/10.1371/journal.pbio.3001421`.

[240] Manuel Holtgrewe. Mason – a read simulator for second generation sequencing data. *Technical Report FU Berlin*, October 2010.

[241] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V. Bzikadze, Alla Mikheenko, et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022.

[242] Giulio Ermanno Pibiri. Fast and compact set intersection through recursive universe partitioning. In *Proceedings of the 2021 Data Compression Conference*, pages 293–302. IEEE, 2021.

[243] Fatemeh Almodaresi, Prashant Pandey, Michael Ferdman, Rob Johnson, and Rob Patro. An efficient, scalable, and exact representation of high-dimensional color information enabled using de Bruijn graph search. *Journal of Computational Biology*, 27(4):485–499, 2020. PMID: 32176522.

[244] Giulio Ermanno Pibiri and Rossano Venturini. Clustered elias-fano indexes. *ACM Transactions on Information Systems*, 36(1):1–33, 2017.

[245] Smruthi Karthikeyan, Joshua I. Levy, Peter De Hoff, Greg Humphrey, Amanda Birmingham, Kristen Jepsen, Sawyer Farmer, et al. Wastewater sequencing reveals early cryptic SARS-CoV-2 variant transmission. *Nature*, 609(7925):101–108, Sep 2022. ISSN 1476-4687. doi: 10.1038/s41586-022-05049-6. URL `https://doi.org/10.1038/s41586-022-05049-6`.

[246] Balamurugan Jagadeesan, Peter Gerner-Smidt, Marc W. Allard, Sébastien Leuillet, Anett Winkler, Yinghua Xiao, Samuel Chaffron, Jos Van Der Vossen, Silin Tang, Mitsuru Katase, Peter McClure, Bon Kimura, Lay Ching Chai, John Chapman, and Kathie Grant. The use of next generation sequencing for improving food safety: Translation into practice. *Food Microbiology*, 79:96–115, 2019. ISSN 0740-0020. doi: https://doi.org/10.1016/j.fm.2018.11.005. URL `https://www.sciencedirect.com/science/article/pii/S0740002018305306`.

[247] Beryne Odeny and Raffaella Bosurgi. Time to end parachute science. *PLOS Medicine*, 19 (9):1–3, 09 2022. doi: 10.1371/journal.pmed.1004099. URL `https://doi.org/10.1371/journal.pmed.1004099`.